HOPEX Power Studio

HOPEX V1

Customizing Documentation

CUSTOMIZING REPORTS (MS WORD)

This section presents customization of reports (MS Word) using report templates (MS Word). The generation mechanism is also presented.

The following points are covered here:

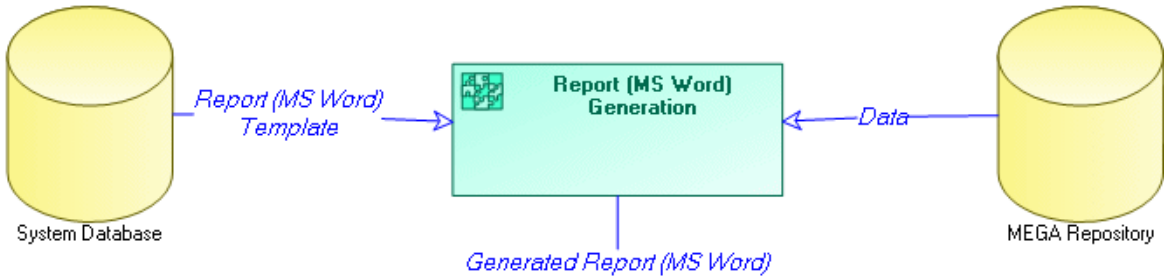
REPORT (MS WORD) CUSTOMIZATION FUNCTIONS

You can use the edit functions of **MEGA** to produce high quality reports using Microsoft Word for Windows™ (from now on referred to as Word).

Reports (MS Word) are created from report templates (MS Word), in the same way as ***MS Word*** documents are created from ***MS Word*** document templates.

Report (MS Word) storage

These report templates (MS Word) are stored in the **MEGA** system repository. This system repository contains all standard supplies that enable **MEGA** operation, for example report templates (MS Word), queries, Web site templates, etc.



Explained here is how you can create and customize report templates (MS Word) to produce reports (MS Word) matching your company standards.

Use of MS Word

To effectively handle reports (MS Word), you do not need to have an in-depth knowledge of ***MS Word***. However, if you want to modify the content or format of reports (MS Word) and report templates (MS Word), you must be familiar with the styles, document templates, and fields as used in MS Word.

The purpose of this guide is not to teach you all about ***MS Word***, but to offer a few guidelines on this software.

> ☛ *You can use **MEGA** with word processing software other than MS Word, but you will be able to execute RTF descriptors only (see "Customizing RTF Descriptors", page 41).*

Use of of reports in MEGA Web Front-End

If you use **MEGA** via the Web interface to generate your reports (MS Word), you must first convert report templates (MS Word) of the environment to RTF format. Microsoft does not recommend use of ***MS Word*** in server mode.

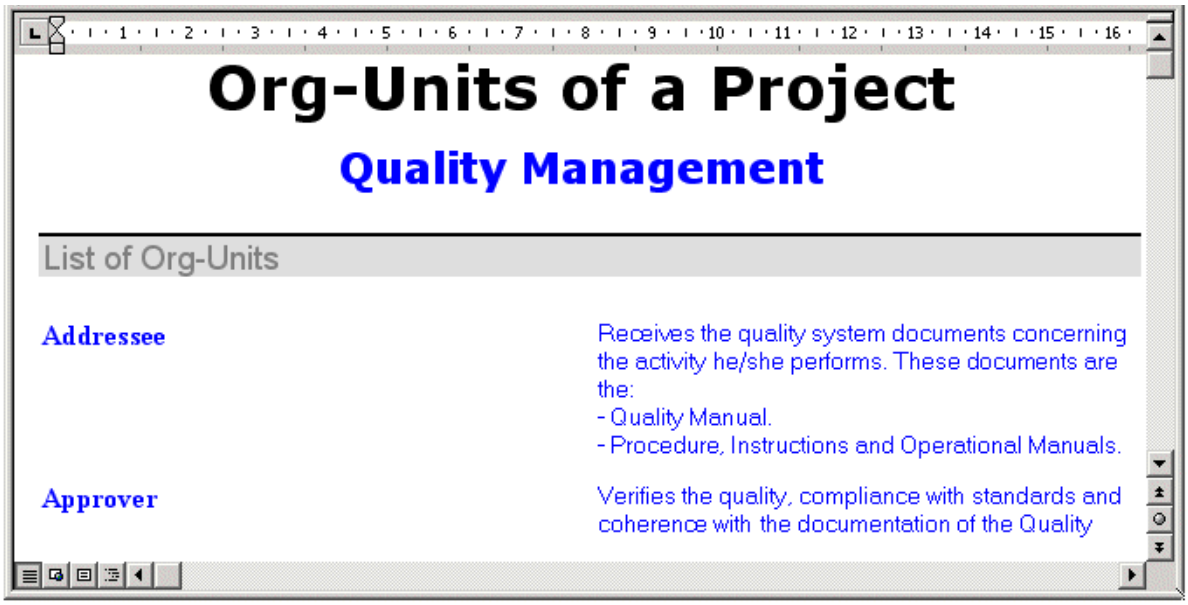
> ☛ *For more details, see "Converting Report Templates (MS Word) to RTF Format", chapter "Managing Environments" in the **MEGA Administration - Supervisor** guide.*

RTF reports can then be produced and packaged in .docx format.

Contents of Generated Reports (MS Word)

Generated reports (MS Word) consists of two parts:

- Texts that you can modify directly in MS Word.
- Description of the objects in the **MEGA** repository. This text is in blue.



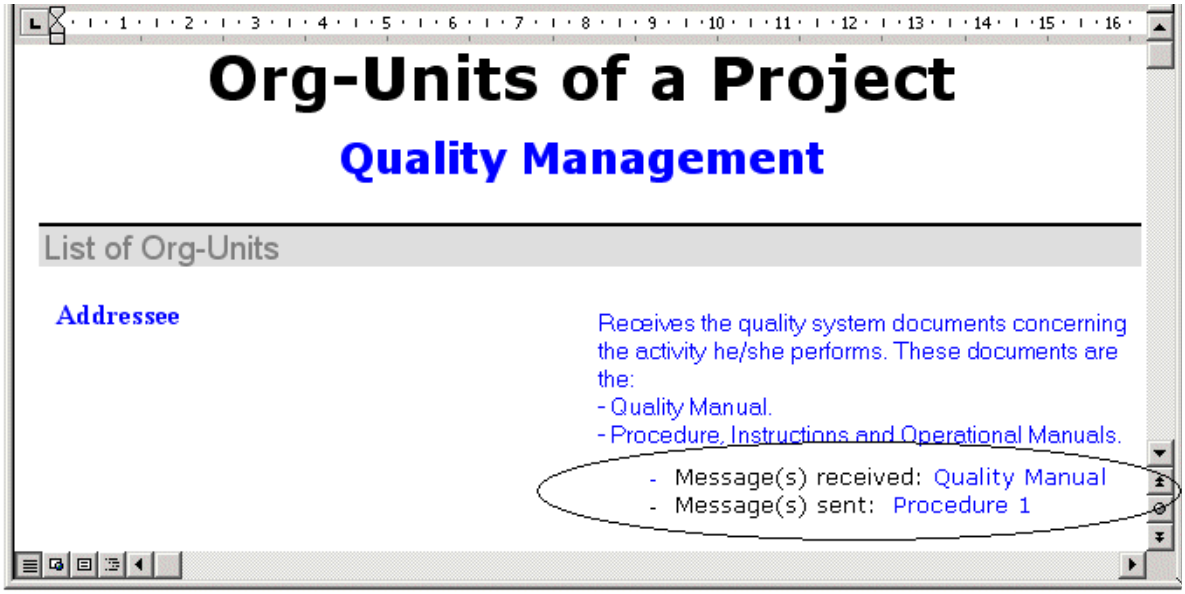
To insert descriptions of **MEGA** repository objects into documents:

1. Select objects to be inserted in the report (MS Word).
2. Select the format for describing these objects.

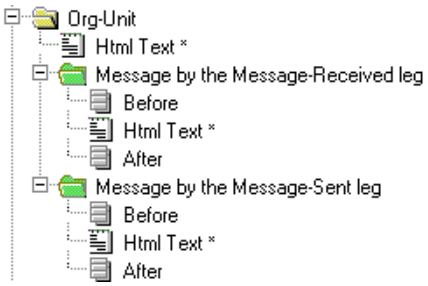
In the above example, the selected objects are the various org-units involved in the "Order Management" project.

Each org-unit name is in bold characters and its comment appears next to it. The text is formatted using ***MS Word*** styles.

An object description can include other objects:

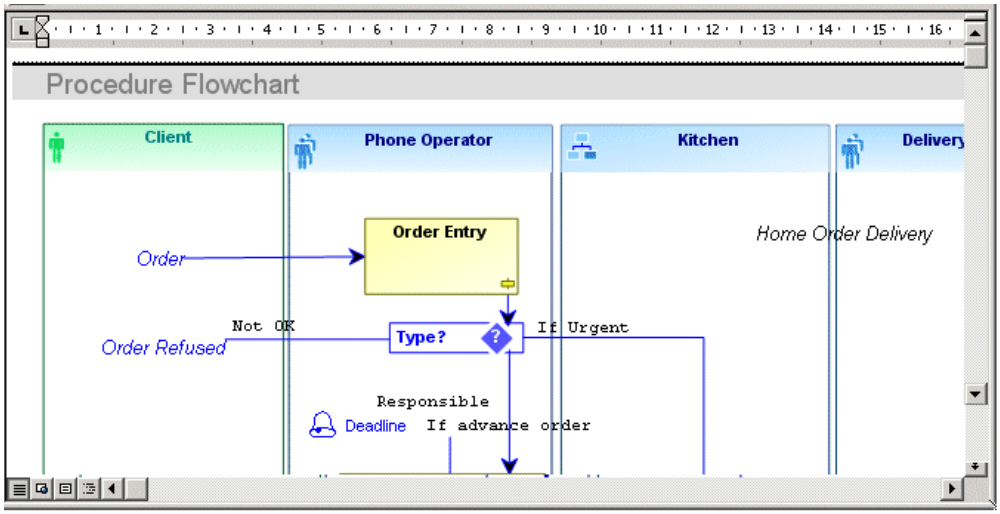


In the above example, the description of the org-unit contains its name and comment plus the messages the org-unit sends and receives.



In this case, the object description, or descriptor, is represented by a menu tree structure with text that explains each included object.

An object descriptor can also contain a drawing.



Report (MS Word) Generation Steps

For each main object type in **MEGA** (business process, organizational process, application, database, etc.), one or several report templates (MS Word) are available.

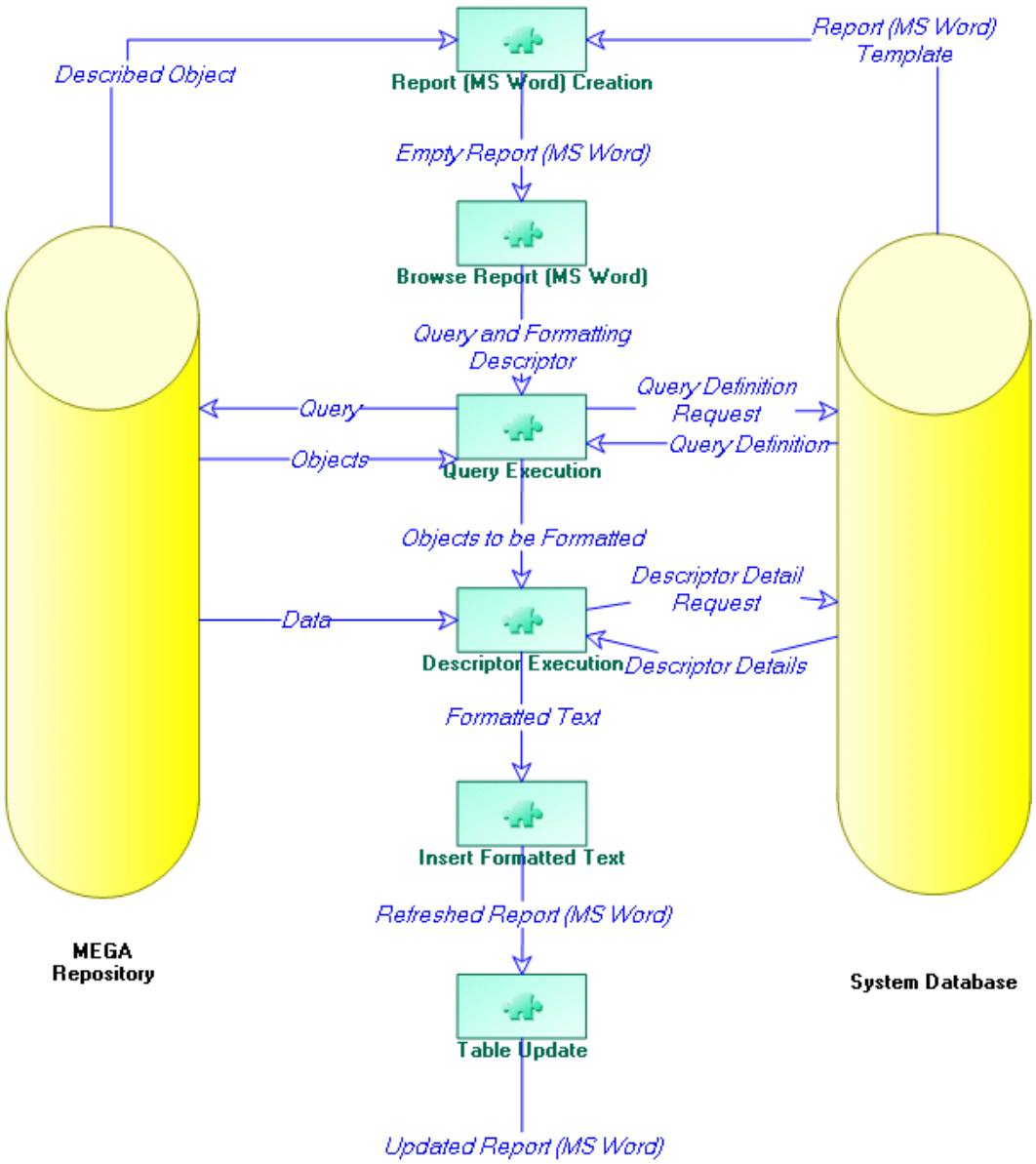
```
For a process for example, there is a general report
template (MS Word), one specific to risk analysis and one
specific to process simulation.
```

The generation of a report (MS Word) in **MEGA** is as follows:

1. **MEGA** searches for report templates (MS Word) to be proposed as a function of the type of object to be described and the products available to the user.
2. If several report templates (MS Word) exist, **MEGA** asks the user to select the appropriate report template (MS Word).
3. The report (MS Word) is automatically created from a copy of the selected report template (MS Word).
4. **MEGA** browses the report (MS Word) to find queries specified in the report template (MS Word).
5. For each query found, the program searches for its definition in the system repository, then executes this in the **MEGA** repository.
6. **MEGA** formats each query result object using the descriptor specified in the report template (MS Word). This descriptor is found in the system repository.
7. The formatted text is inserted in the report (MS Word).
8. The program processes the next query.

9. When all queries have been processed, the program updates the table of contents and index if these exist.

   ☛ *Note that for .docx reports generated on **MEGA Web Front-End**, the table of contents need to be updated manually.*



CREATING A REPORT TEMPLATE (MS WORD)

You can create as many reports (MS Word) as you want, and modify them as many times as you want.

However, if you want to produce several reports (MS Word) of the same type, it is recommended that you use a report template (MS Word) to create each report (MS Word).

A report template (MS Word) provides the framework of a report (MS Word). it is fleshed out with data from the repository when creating a report (MS Word). It contains texts and report template (MS Word) elements which allow you to rapidly create reports (MS Word) associated with an object.

You can create a report template (MS Word) even if you do not have the **HOPEX Studio** technical module.

> ☛ *The standard report templates provided are protected. You must duplicate them if you wish to modify or create new templates from those proposed.*

In the following example you will create a new report template (MS Word) describing the list of org-units in a project with their comments.

To create a report template (MS Word) from an existing one:

1. From **MEGA**, select the **Utilities** navigation window.

   > ☛ *To access the **Utilities** window, select **View > Navigation Windows > Utilities**.*

2. From the **Report Templates (MS Word)** folder, right-click the report template (MS Word) from which you want to create your new template and select **Duplicate**.

   ```
   Example: "Standard".
   ```

   - The "Standard" report template (MS Word) contains style and macro definitions only. Use this template to create a new blank report template (MS Word).
   - If you select a different report template (MS Word), your new report template (MS Word) will start with the framework of this template. You can then modify the new report template (MS Word) as desired.

   The **Duplicate an Object (Report Template (MS Word))** dialog box appears.
3. (Optional) Modify the **Name** of the report template (MS Word) to be created.
4. In the **Strategy** field, select **Reuse descriptors and queries**.

   > ☛ *For more details on duplication of report templates (MS Word), see "Duplicating Descriptors", page 56.*

5. Click **OK**.
   When duplication has been completed, the element corresponding to the new report template (MS Word) is created.

To open the report template (MS Word) you created:

- Right-click the report template (MS Word) you created and select **Open**.
  The MS-Word application is displayed in the foreground.

Modifying a Report Template (MS Word)

Report template (MS Word) modifications can apply to page format, headers and footers and accompanying text entered in MS Word: in this case they concern only MS Word, and are made as in normal use of this software.

☛ *See "Entering text in your MS Word report template", page 26*

To display the name of a project, that is to say the name which should be instanced in each report (MS Word), you need to insert a "report template (MS Word) element".

☛ *See "Inserting Report Template (MS Word) Elements", page 26*

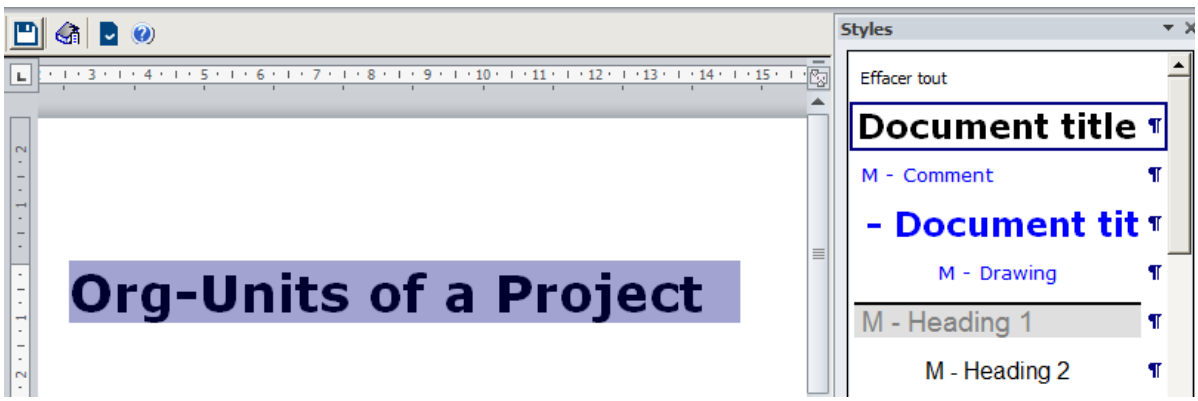
Entering text in your MS Word report template

To add a title:

1. In the report template, enter the title.

   ```
   Example: "Org-Units of a Project"
   ```

2. Select the title (for example: "org-Units of a Project") and in the MS Word **Styles** apply "Document title" style.



Inserting Report Template (MS Word) Elements

Report template (MS Word) elements query the repository; they consist of:

- a descriptor (see "Customizing RTF Descriptors", page 41)
- a query (see the **MEGA Common Features** guide).

Report template (MS Word) elements can be inserted in report templates (MS Word): they are transformed into report (MS Word) elements when creating a report (MS Word).

To insert report template (MS Word) elements in a report template (MS Word):

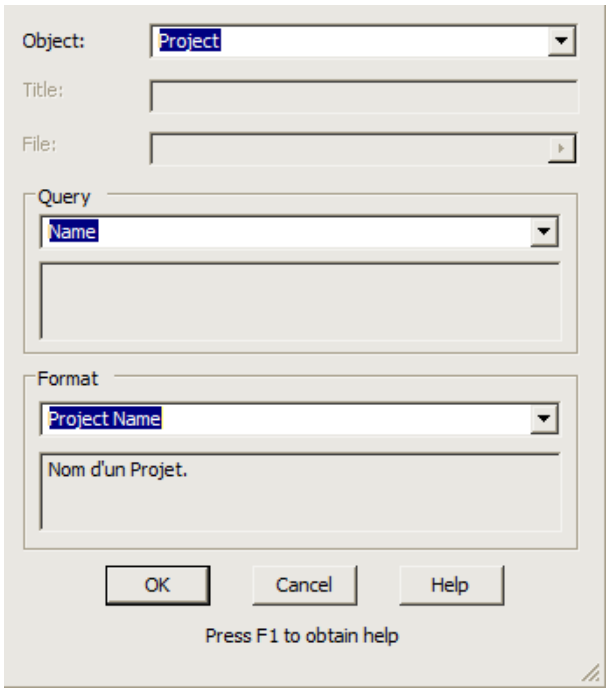
1. Position the cursor where you want to insert the "report template (MS Word) element.

2. In the edit toolbar, click **Insert a Report template (MS Word) Element**  .
   The **Insertion of Report Template (MS Word) Element** dialog box opens.
3. In the **Object** field, select the type object to which the report template (MS Word) element relates.

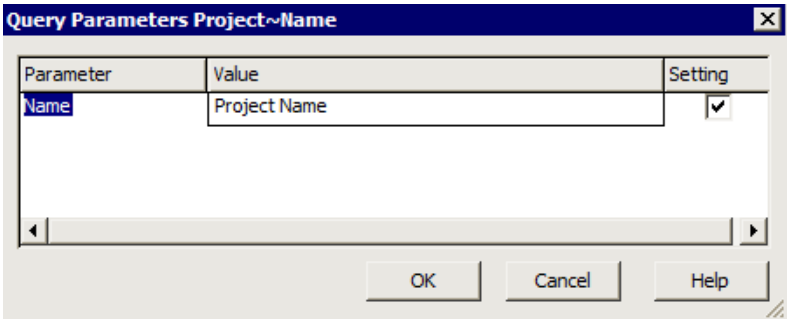
   `Example: “Project”.`
4. In the **Query** field, select the query criterion for of the desired project.

   `Example: "Name" which will search for projects by name.`
5. In the **Format** field, select the format of the result with a description.

   `Example: The "Project Name" descriptor enables display of the project name.`



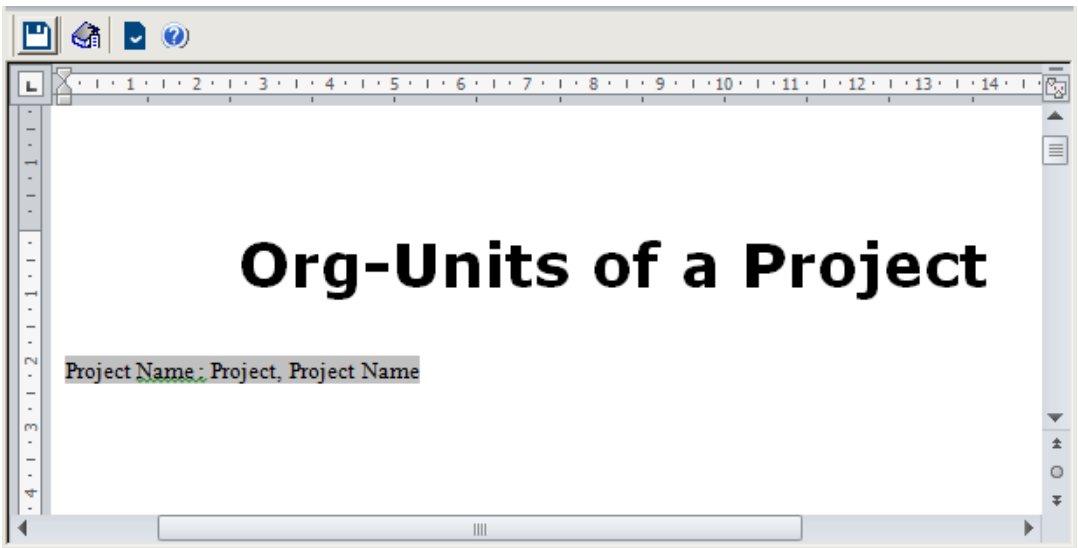
6. Click **OK**.
   An intermediate dialog box appears where input is required in our example:



   The software will request a value for this setting when it creates a report (MS Word) based on this template.
7. In the **Value** filed, enter the value.

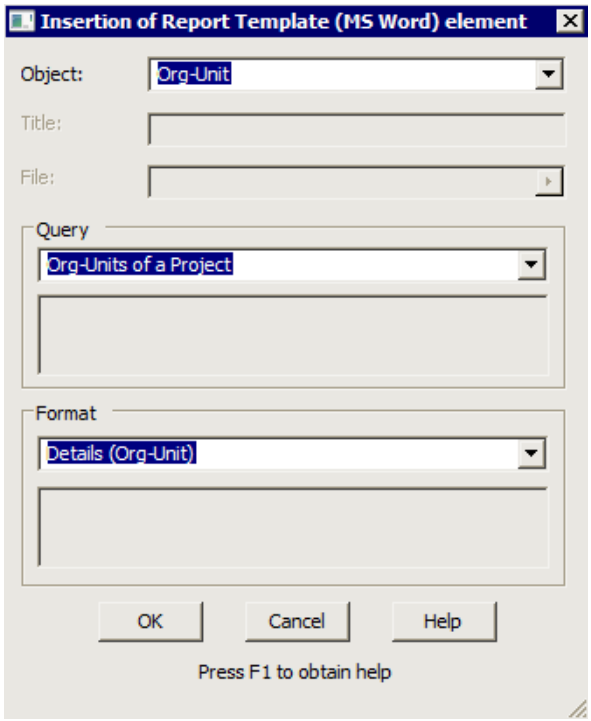
   `Example: "Project Name".`

   ☺ *If you clear the **Setting** check box, the value you enter will be used as the setting value when the report (MS Word) is created.*
8. Click **OK**.
   The first report template (MS Word) element has been inserted in the document template.



9. Enter the title of the next paragraph (for example: "List of Org-Units"), and apply the "Heading 1" style.

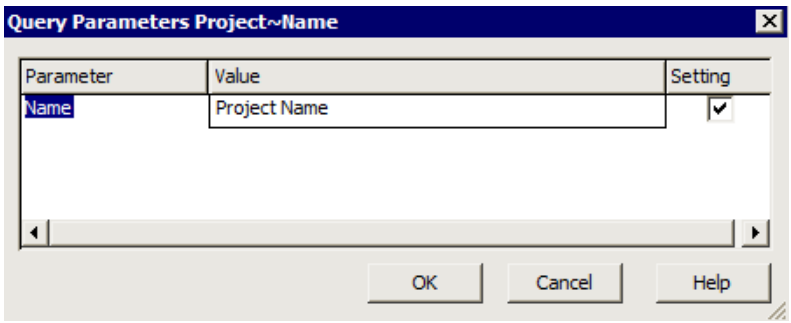
10. Insert a second report template (MS Word) element, which will present the "Org-Units of a Project" with the "Details (Org-Unit)" descriptor.



11. Complete the **Query** and **Format** fields and click **OK**.
12. In its project text box, enter the name of the setting value which will be requested when the report is processed.

> ☛ *Several report template (MS Word) elements can use the same setting. For this to be possible, give the same name to the settings associated with each of these elements.*

```
Example: enter "Project Name".
```

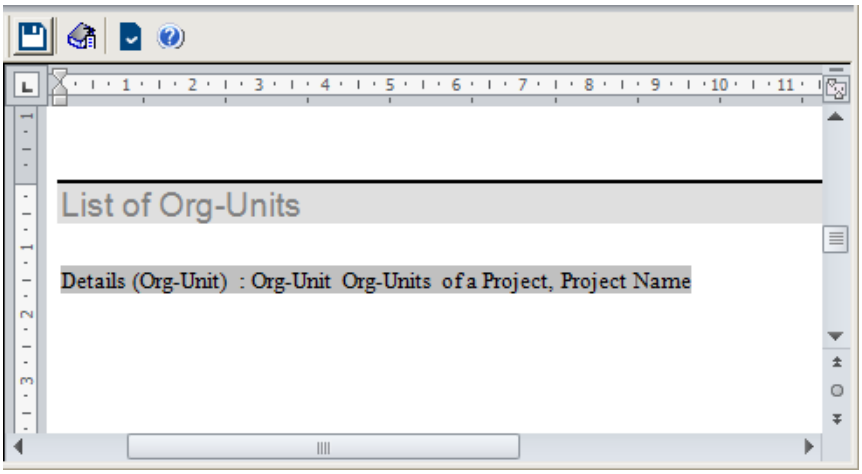


13. Click **OK**.

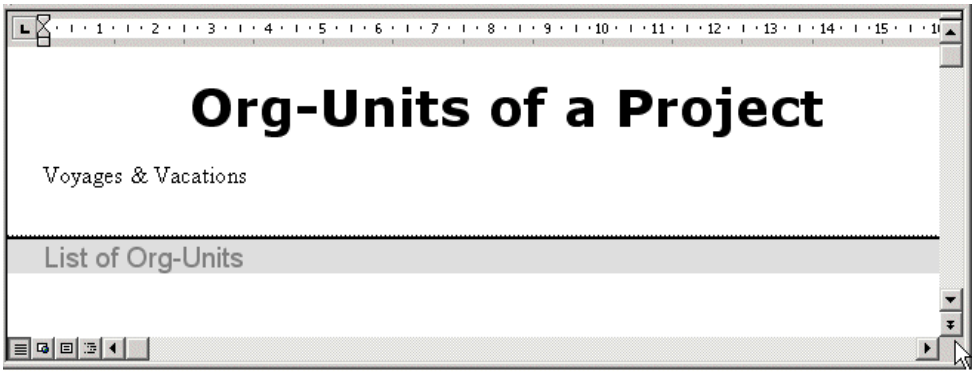
> ☛ *Insertion of a report template element is by insertion of a report element in MS-Word, followed by the name of the descriptor used (here,*

"Details (Org-Unit)") followed by the names of the query and the setting (in this example, "Org-Units of a Project", Project Name"):

Report template (MS Word) element 'Details (Org-Unit): Org-Units of a Project, Project Name'



14. In the edit toolbar, click **Save** .
    You can now create reports (MS Word) from your new report template (MS Word). T



Deleting a Report Template (MS Word) Element

To delete of a report template (MS Word) element:

- Delete the corresponding line.

Replacing a Report Template (MS Word) Element

You can replace a report template (MS Word) element by a new element.

☛ *The standard report templates provided are protected. If you want to modify them, you must first duplicate them. The same applies for descriptors.*

To delete a report template (MS Word) element and insert the new element:

1. Open the report template (MS Word).
2. Delete the report template (MS Word) element by deleting its code.
3. From the document edit toolbar, click  to insert a new report template (MS Word) element.

Backing Up a Report Template (MS Word)

To create a backup copy for use in another environment, you need to extract the report template (MS Word) together with the queries and descriptors it uses.

To extract a report template (MS Word):

1. In the navigator, right-click the report template and select **Properties**.

   For example: "Org-Units of a Project" (MS Word)

2. Select the **Documented object** tab.
3. Make sure that the documented **Object type** appears.

   Example: "Project"

   The **Refresh** button updates the links between the report template (MS Word) and its components (queries and descriptors).

   ☺ *If queries or descriptors have been modified or renamed, we recommend that you click the **Refresh** button before extraction. When in doubt, always click this button before extraction.*

4. If necessary, specify the **Category** to which the object type is attached (a category is a set of objects of the same type).

   ☛ *A category is a group of objects of the same type. For example, "Organizational Chart" is a category of the "Diagram" type. A type may have several categories.*

5. Right-click the report template (MS Word) and select **Manage > Export**. The **Export MEGA objects** dialog box opens.
6. Click **Export**.

PRIVATE AND PUBLIC REPORT TEMPLATES (MS WORD)

At the time of its creation, a new report template (MS Word) is private and is therefore visible only to its creator.

This report template (MS Word) can be made public so that it will be available to other users.

> ☛ *For more details, see "To make a private report template (MS Word) public:", page 33.*

To access report templates (MS Word):

1. In **MEGA**, open the **Utilities** navigation window.
2. Expand folders **Report Template (MS Word)** and **Private**.
   The **Private** folder contains private report templates (MS Word) of the current user.

   A private report template (MS Word) is owned by the user that created it. You can see this by exploring the report template (MS Word).

To explore the report template (MS Word) and see its creator:

1. Right-click the report template (MS Word) and click **Explore**.
2. In the window that opens, expand the **User** folder.
   The Report Template (MS Word) creator is displayed in the right pane of the window.

> ☛ *To see this folder, you must have "Advanced" access to the metamodel.*

Making report templates (MS Word) available

Making a private report remplate (MS Word) public

To make a private report template (MS Word) public:

1. In the **Utilities** navigation window, expand the **Report Template (MS Word)** and **Private** folders.
2. Right-click the desired report template (MS Word) and select **Manage > Make Public**.
   The report template (MS Word) is moved to the global list of report templates (MS Word).

> ☛ *A report template (MS Word) that has been made public can be returned to private status.*

Making a report remplate (MS Word) private

To make a report template (MS Word) private:

- Select the report template (MS Word) and drag it into the **Private** folder.
  When a report template (MS Word) has been made public, other users of the environment can reuse or modify this report template (MS Word).

Deleting Private Report Templates (MS Word)

Private report templates (MS Word) and descriptors can be deleted at repository cleanup.

To delete all private report templates (MS Word) and descriptors:

1. From **MEGA** menu bar, select **File > Properties**.
2. In the **Characteristics** tab, click **Repository Cleanup**.
   The repository cleanup dialog box presents groups of elements that can be deleted. The private report templates (MS Word) and descriptors proposed are those of the current user.

   ☺ *To display the list of private report templates (MS Word) and descriptors, click **View**.*
3. Click **OK**.
   Selected elements are deleted.

To delete private report templates (MS Word) one-by-one:

1. Right-click the report template (MS Word) name and select **Delete**.
   A dialog box asks you to confirm deletion.
2. Click **Delete**.
   The report template (MS Word) is deleted.

CONVERTING REPORT TEMPLATES (MS WORD) FOR THE WEB

To generate report templates (MS Word) from the **MEGA Web Front-End**, you must first convert the report templates (MS Word) of the environment to RTF format. Microsoft does not recommend use of MS Word in server mode, which is why it is necessary to carry out this conversion.

The conversion applies to:

- report templates (MS Word)
- environment style sheets

When conversion is completed, the new report templates (MS Word) are created in RTF format and are packaged in .docx format.

> ☛ *Reports (MS Word) generated from these report templates (MS Word) are in the new format not only for users of **MEGA Web Front-End** but also for users of **MEGA Windows Front-End**.*

Converting Report Templates (MS Word)

Converting report templates (MS Word) of an environmrent

For more details on how to convert report templates (MS Word) to RTF format, see the **MEGA Administration - Supervisor** guide, "Managing Environments".

Converting report templates (MS Word) separately

You may need to convert a report template (MS Word) separately, for example if it is recovered from another environment via a .mgr file.

To convert a report template (MS Word) separately:

1. Right-click the report template (MS Word) and select **Convert to RTF format**.

   > ☛ *The command is only available in an environment in which report templates (MS Word) have already been converted to RTF format.*

   The report template (MS Word) and associated style sheet are converted to RTF format.

Converting a Style Sheet

When you convert report templates (MS Word) of an environment to RTF format, the associated style sheets are converted at the same time.

You may however have to convert a style sheet later (for example if the style sheet has been associated with a report template (MS Word) after its conversion).

To convert a style sheet:

- Right-click the report template (MS Word) and select **Manage > Convert style sheet to Web mode**.
  The style sheet is stored in the **Mega_Usr** folder of the environment.

CUSTOMIZING REPORT (MS WORD) SENDING

MEGA enables sending of reports (MS Word) for review or validation. This function is accessible via the commands **Send for Review** and **Send for Validation** in the report (MS Word) properties dialog box.

Information concerning recipients is based on content of the **Distribution** tab of the report (MS Word) properties dialog box.

> ☛ *Rules concerning the recipients list are described in the **MEGA Common Features** guide, chapter "Generating documentation", paragraph "Distributing a report (MS Word) for review or validation".*

To customize this function (for example, configure object content, recipients, etc.), you should override the C++ code in **MEGA**.

To intervene at this level, you must have "Advanced" access to the metamodel.

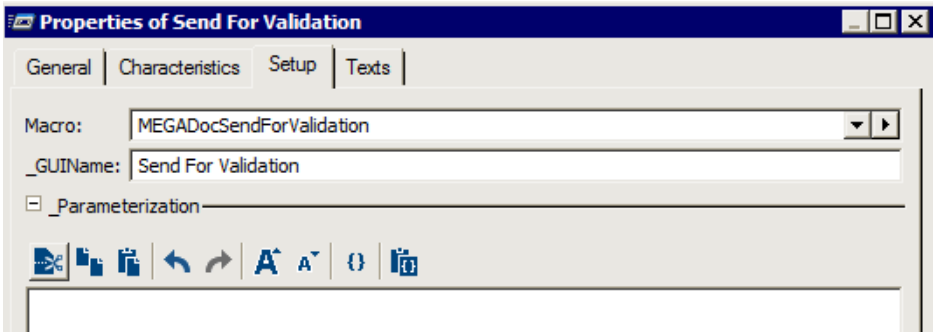
> ☛ *Access to the metamodel is defined in the options: from MEGA menu bar, **Tools > Options > Repository**, in the right pane for **Metamodel Access** option select "Advanced".*

To customize report (MS Word) sending:

1. Explore the **Report (MS Word)** MetaClass.
2. Right-click the Command (for example: "Send for Validation") and select **Properties**.
3. From the **Setup** tab, you can access macros.
   The macro "MEGADocSendForValidation" calls C++ code that executes standard code.
4. In the **Macro** field, click the right-oriented arrow and select **Create** to create a new macro.

   `For example: "MyDocSendForValidation".`

   > ☛ *In the option (**Tools > Options > Repository**), the **Authorize MEGA data modification** option value must be "Authorize".*



5. Open the **Properties** of this new macro.
6. Update the script in the **VB Script** tab of the macro properties and click **OK**.

   > ☛ *For more information on these functions, see the **All about Starting with APIs** technical article.*

   > ☛ *To call a VB script of this type, you must use a .dll (cdo.dll) and deploy it on all workstations. The user must have adequate*

administration rights for a script to be executed automatically on his or her workstation.

CUSTOMIZING RTF DESCRIPTORS

You can customize the *descriptors* provided at installation to suit your specific needs if you have the **HOPEX Studio** technical module.

You can also create new descriptors to:

- produce files in RTF format that can be used by most word processors.
- insert the produced files in *report (MS Word) elements* or *report template (MS Word)* elements used in your reports (MS Word) and report templates (MS Word).

✓ "Creating RTF Descriptors", page 42
✓ "Defining descriptors", page 44
✓ "Executing Descriptors", page 51

CREATING RTF DESCRIPTORS

To display descriptors you need to be in "Advanced" metamodel access.

☛ *Access to the metamodel is defined in the options: from **MEGA** menu bar, **Tools > Options > Repository**, in the right pane for **Metamodel Access** option select "Advanced".*

RTF descriptor categories

Descriptors are either:

- ***Contextual*** descriptors, which are designed to be used in a report template (MS Word).
- ***Elementary*** descriptors, which can be used in a report template (MS Word) or processed alone.
- ***Customized*** descriptors that you create yourself to meet your specific needs.
- ***Private*** descriptors

Creating a Descriptor

☛ *To create a descriptor from an existing descriptor, see "Duplicating Descriptors", page 56.*

To create a descriptor:

1. From **MEGA**, open the **Utilities** navigation window.

   ☛ *To access the **Utilities** window, select **View > Navigation Windows > Utilities**.*

2. Expand the **Descriptors** and **Rtf Format** folders.
3. Right-click the descriptor category folder corresponding to the descriptor category you want to create and select **New > Descriptor**.

   ☛ *See "RTF descriptor categories", page 42.*

   `Example: Customized category.`

   The **Create Descriptor** dialog box opens.
4. In the **Name** field, enter the name of your descriptor.
5. In the **Described object** field, select the described object, for example Org-Unit.
   The descriptor can also relate to an abstract MetaClass.

   ☛ *For more details on the abstract metamodel, see **HOPEX Studio**, "Managing the Metamodel", "Abstract Metamodel".*

   ☛ *If the name you entered is already used by another descriptor, **OK** is unavailable (gray).*
6. Click **OK**.

Descriptors and inherited objects

If the global option concerning variations is activated (**Activate variations** options in **Options > Business Process and Architecture Modeling**) you can take account of inheritance.

> ☛ *For more details on inheritance and object variations, see the **MEGA Common Features** book "Handling Repository Objects", "Object Variations".*

When inheritance is taken into account repository paths specified in descriptors (via queries or MetaAssociationEnds) also include inherited objects.

You can define to take account of inheritance:

- globally at descriptor level
- specifically at the level of a given path

To take account of inheritance:

1. Right-click the descriptor, and select **Properties**.
2. In the **Characteristics** tab, select the **Take account of inherited objects** check box.
   Inheritance is now taken into account in the descriptor, except if you specify a different behavior on certain elements of the structure.

DEFINING DESCRIPTORS

The object descriptor structure is made up of several linked groups organized in a tree. Each group relates to an object and specifies the query or link, which enables passage from the previous object to this object.

Text is the basic element used to define what is displayed for each of the objects in a generated report (MS Word).

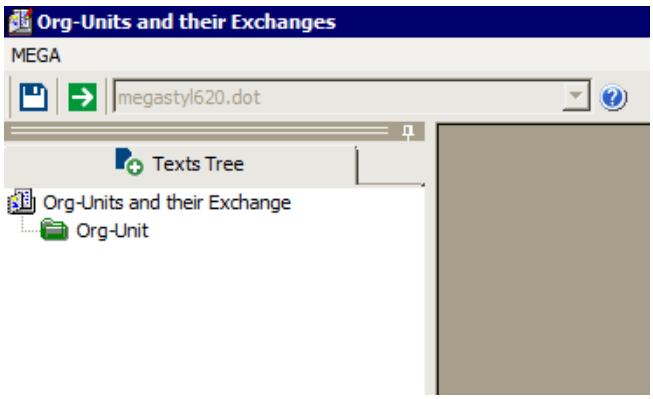
Accessing the descriptor structure

To access the descriptor structure:

1. From the **Utilities** navigation window, double-click the created descriptor.

   ☛ *See "Creating a Descriptor", page 42.*

   The descriptor editing window appears and displays the **Texts Tree** tab.



To create the descriptor structure, see:

- "Adding Text to a Descriptor", page 44
- "Creating Groups", page 46

Adding Text to a Descriptor

Text is the basic element used to define what is displayed for each of the objects in a generated report (MS Word).

```
Example: you can add text to the "Org-Unit" object to edit
its name and comment.
```

To add text to a descriptor:

1. Access the descriptor structure.

   ☛ *See "Accessing the descriptor structure", page 44.*

2. Right-click the folder representing the described object (Example: **Org-Unit**) and select **New > Text**.

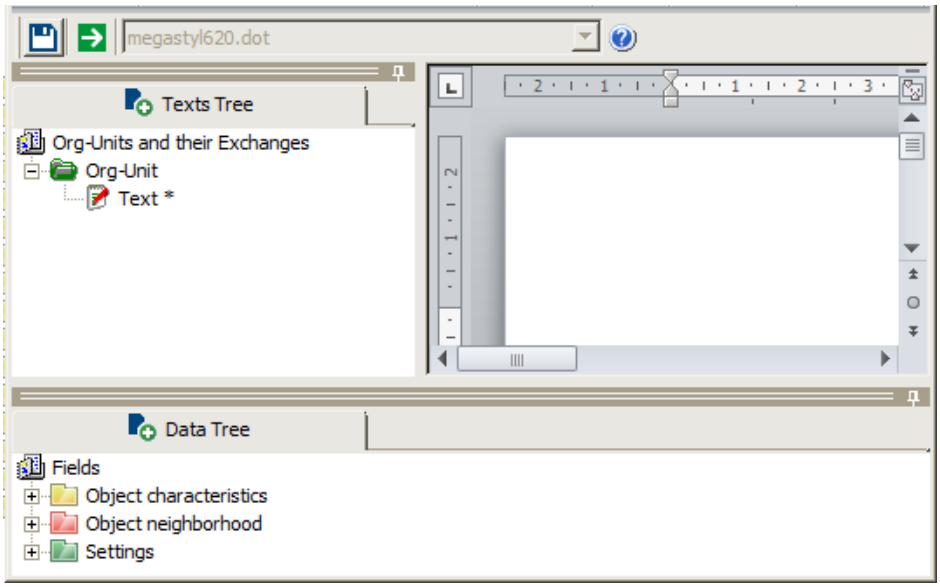
   `Example: Org-Unit.`

   Text is added in the **Texts Tree** tab.

   ☛ *The icon indicates which text is currently being edited.*

   The following panes appear in the descriptor editing window:

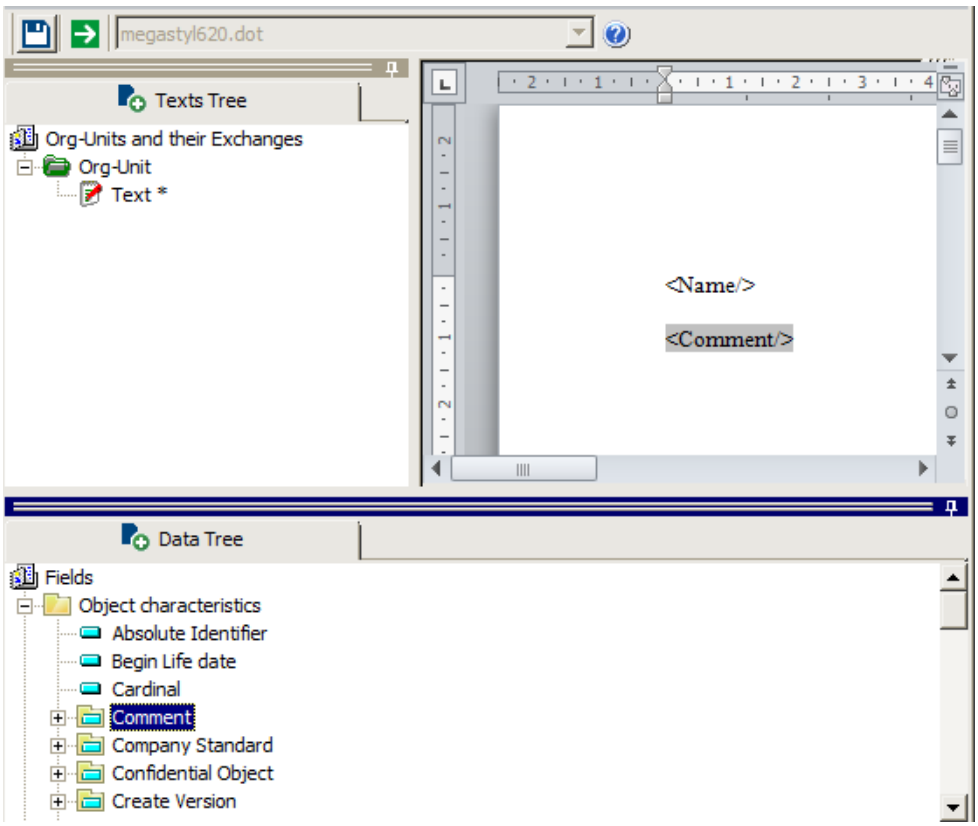
   - an MS Word pane (top right pane) to edit the text.
   - a **Data Tree** tab (bottom pane), which enables you to select a field and drag and drop it for edition in MS Word pane.



3. In the **Data Tree** tab select the field you want to insert in the text.
4. Drag and drop the field in the top-right pane.

   `Example: to insert the Name and Comment fields in the text area, in the Object characteristics folder, successively`

select the **Name** and **Comment** fields and drag-and-drop them into the desired location in the text area.



5. (Optional) Customize your document format by applying Word styles.

   ☛ *These styles have the prefix M- and are based on the M-Normal style that is similar to the Word Normal style. Note that the M-Normal style text is in blue.*

   ☛ ***Fields** from **Object characteristics** folder cannot be manually edited. You must insert these in the text by drag-and-drop as explained above, or by copy/paste.*

6. Click **Save**.
7. In the **Texts Tree** tab, right-click **Text** and select **Close**.
   The text is automatically saved and the icon returns to its original form.

Creating Groups

The object descriptor structure is made up of several linked groups organized in a tree. Each group relates to an object and specifies the query or link, which enables passage from the previous object to this object.

Example: You can display the messages sent by each org-unit.

To create a group:

1. Right-click the folder representing the described object (Example: **Org-Unit**) and select **New** > **Group**.
   The **Group Properties** dialog box opens.
2. (Optional) In the **Title** field, enter a title.

   ☛ *In most cases, you may choose not to give a **Title** to the group, since the object and query names provide sufficient information to identify the group.*
3. You can select a registered query/MetaAssociationEnd or embed a query.
   Select either:
   - **Query or MetaAssociationEnd** to select a registered query.

     🕮 *A MetaAssociationEnd is the extremity of a MetaAssociation linking the MetaAssociation to a MetaClass.*

     ☛ *Concrete links as well as generic links are proposed in the list of MetaAssociationEnds.*

     Example: querying organizational processes linked to an operation, you will see all links connecting these two object types, including the "Behavior" generic link that will find the organizational process describing behavior of the operation.

     In the **Described object** field, select the described object.

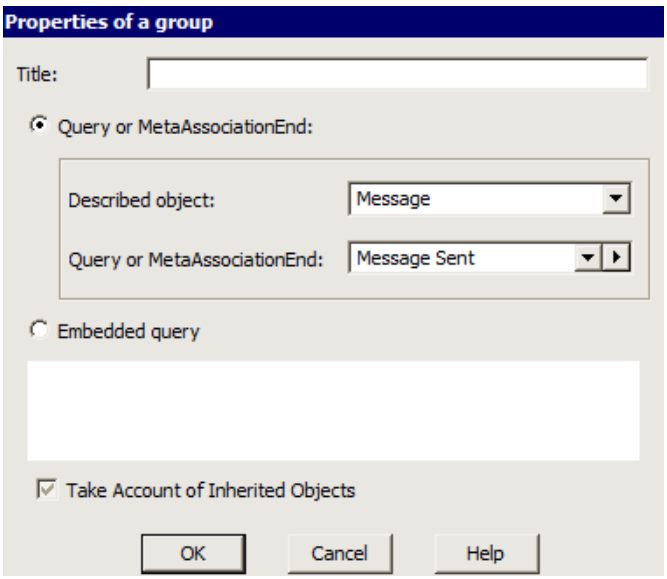
     Example: "Message" object

     In the **Query or MetaAssociationEnd** field, select the query/MetaAssociationEnd.

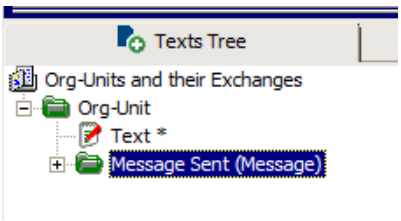
     Example: “Message-Sent”.
   - **Embedded query** to embed a query that you do not want to save in the repository. If you have a specific need to write a query that will never be reused elsewhere. Enter your query code directly in the corresponding frame.

     ☛ *For more details on queries, see **MEGA Common Features**.*

4. If the general option on variations is activated, the **Take Account of Inherited Objects** option is displayed.
   You can choose to:
   - follow default behavior defined at descriptor level (see "Descriptors and inherited objects", page 43): keep the check box grayed: ☑ (or ■ according to the selected Windows theme).
   - activate or deactivate the option at group level (by selecting or clearing the check box).



5. Click **OK**.
   The created group is displayed under the described object folder.



Adding Text to a Group

You can add text:

- before the object
- to the object
- after the object

To add text to a group:

1. Expand the group folder.

   ☛ *See "Creating Groups", page 46.*

2. Right-click **Before** (/**After** according to where you want to add text) and select **Text** to add text at the beginning (/end).

   Example: select **Before** to add text at the beginning of the list of messages sent by an org-unit.

3. In MS Word pane, enter the text.

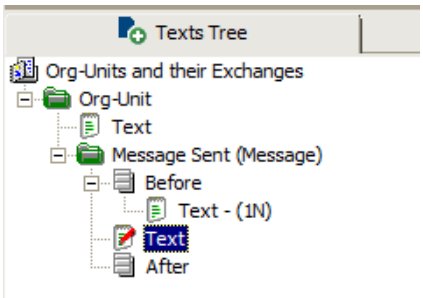
   Example: "Messages sent" that you want to display before the list of messages.

4. Apply a style to your text.

   Example: "M-List"

   See "Managing Formats and Styles", page 77 for more information on styles.

5. Right-click **Text** and select **Close** to close the Text editor panes.
6. To add text to the object (Example: "Message sent"), right-click the object and select **New > Text**.



   Another **Text** icon appears in the tree.

7. From the **Data Tree** tab, in the **Object characteristics** folder, select **Name** and drag and drop it into MS Word pane.

   Example: Add **Name** and **Comment** object characteristics

8. (optional) Customize the layout of your document using the styles provided by **MEGA** and MS Word.
9. Right-click **Text** and select **Close** to close the Text editor panes.
10. (optional) Create other groups.

    Example: Create a group that displays the messages received by an org-unit.

    ☛ *Do not forget to save your work from time to time (select **Descriptor > Save**).*

Specifying text execution conditions

You can define execution conditions on a "Before" text. You can vary the text according to the number of objects resulting from execution of the query.

Execution condition can be:

- **No object**: the text is displayed if there is no object

  Example: "No message sent"
- **A single object**: the text is displayed if there is only one object

  Example: "Message sent"
- **One object or more**: the text is displayed if one or more objects

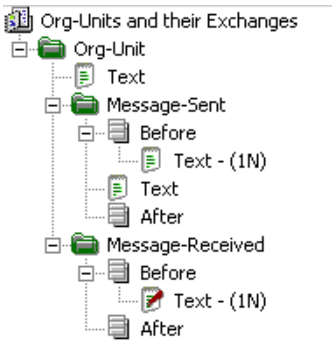
  Example: "Message(s) sent"
- **More than 1 object**: The text is displayed when there are several objects

  Example: "Messages sent"

To define text execution conditions:

1. From the descriptor editing window, under the **Before** folder, right-click **Text** and select **Properties**.
2. Select the **Execution** tab.
3. Select the execution condition.
   The text is edited depending on the number of objects found on execution of the query.

   Example: The text is edited depending on the number of messages sent by the org-unit



☛ *To reorder the descriptor text and groups see "Defining the Object Order in Reports (MS Word)", page 65.*

EXECUTING DESCRIPTORS

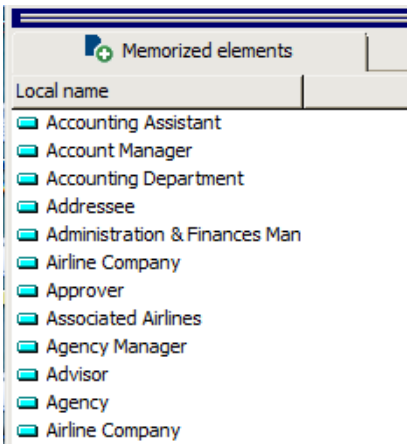
Executing a descriptor

To execute a descriptor:

1. Access the descriptor editing window.

   ☛ *See "Accessing the descriptor structure", page 44.*

2. From the descriptor editing window, select **MEGA > Process**.
   The query dialog box appears.
3. Click Find.
4. In the dialog box that opens, select the objects that interest you and click **OK**.
   A message box shows the progress of descriptor processing.
   Objects selected are memorized and displayed in the **Memorized elements** tab.



The result is displayed in an MS Word window:

Re-executing the descriptor from memorized objects

MEGA memorizes objects used in the last execution of the query. These objects appear in the **Memorized Elements** tab (see "Executing a descriptor", page 51).

To re-execute a descriptor from memorized object:

1. (optional) If needed, from the descriptor editing window, delete an object from the **Memorized Elements** tab list (right-click the object and select **Remove**).
   This set of objects enables time saving when executing and testing the descriptor.

2. Click **Runs descriptor**  and select **Execute on all memorized elements**.
   The reports execute and the RTF format document is regenerated.

ADVANCED FEATURES OF RTF DESCRIPTORS

Previous chapters detail the basic principles of customizing reports (MS Word) and RTF descriptors. This chapter introduces advanced functions that you can use to refine your descriptors.

☛ *These descriptor customization functions are available only with the **HOPEX Studio** technical module.*

The following points are covered here:

Modifying Descriptors

Duplicating Descriptors

☛ *A number of standard descriptors are created by the installation program. These standard descriptors are read-only so as to avoid conflicts when you upgrade your version of **MEGA**. If you want to make changes to a standard descriptor, you must duplicate it.*

To duplicate a descriptor:

1. From **MEGA**, open the **Utilities** navigation window.

   ☛ *To access the **Utilities** window, from **MEGA** menu bar select **View > Navigation Windows > Utilities**.*

2. In the **Descriptors** folder, right-click the descriptor you want to duplicate and select **Duplicate**.
   The **Duplicate an Object (Descriptor)** dialog box appears.
3. In the **Strategy** field, keep **Duplicate** value if you want to modify descriptor content (example: macro or query).

   ☛ *Select **Reuse descriptors and queries** if you only want to reuse the descriptor content.*

4. (Optional) In the **Prefix**/**Suffix** field, modify the prefix/suffix used by default to create the name of duplicated descriptors.

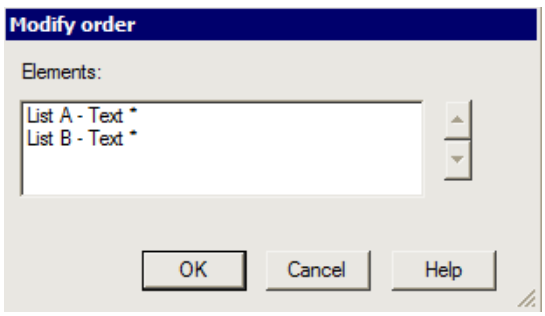
   ☛ *By default: **(Duplicate)** suffix.*

5. (Optional) Modify the **Name** of the duplicated descriptor.
6. (If needed) In the **Library** pane, select **Create duplicates in another library (for objects stored in a library)** and select the library in the drop down list.

Modifying the Order of Groups and Texts in Descriptors

To move groups and texts within the descriptor structure:

1. From the **Utilities** navigation window, right-click the descriptor item you want to reorganize (example: a group) and select **Reorganize**.
   The **Modify order** dialog box appears.



2. Select the element to be moved.

   ```
   Example: "List B - Text"
   ```

3. Use the up and down arrows to move the selected element.
4. Click **OK**.
   The item order is modified.

USING BLOCK TYPES

The followings points are covered here:

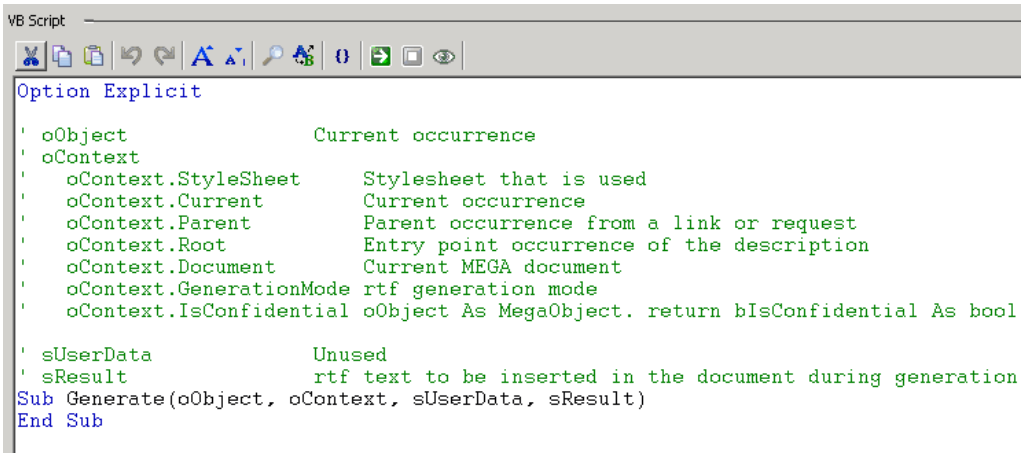
Using Macros

MEGA enables addition of macros in descriptors. You can therefore communicate with Word via VB Script and use APIs to access the repository.

Adding a macro

To add a macro:

1. Right-click a group and select **New > Macro**.
   The **Create Macro** dialog box opens.
2. Enter the macro name and click **OK**.
   A VB Script tab appears.



Macro principle

The macro is called by the "Generate" method from the following code:

```
Explicit Option

Sub Generate(oObject, oContext, sUserData, sResult)

End  Sub
```

- oObject: the current object in the descriptor
- oContext: macro generation context which supports the following properties:
  - StyleSheet: report (MS Word) style sheet address
  - Current: current object on which macro executes
  - Parent: parent object
  - Root: descriptor entry point object
  - GenerationMode: rtf generation mode
  - IsConfidential: returns a boolean
- sResult: string in which RTF generated by the macro is written

Macro availability

A macro can relate to one, several or all the MetaClasses.

The MetaClass to which the macro relates should appear in the **Characteristics** tab of the macro properties.

☛ *If no MetaClass is indicated in the macro properties, this means that the macro relates to all the MetaClasses.*

Macros example

Two macros are provided as standard. You can configure these by first duplicating them.

- ***External Reference Content***: manages insertion of content of external references in the report (MS Word). Various formats are proposed:
  - .txt, .ini, .log, .xml: files are read in text format.
  - .rtf, .wri, .doc, .xls, .bmp, .gif, .jpg, .png, .tif: files are read/converted to .rtf or image format.

    ☛ *In **MEGA Web Front-End**, you cannot reproduce the content of external references.*
- ***Business process hypothesis achievement matrix:*** enables creation of a matrix type table.

Macro structure

The following code gives the structure enabling opening of a Word session in VB, creating an empty report (MS Word) and saving its content in RTF. Word APIs then enable writing of text, tables, etc.

```
Sub RTFGenerate(mgoContext, strResultRtf)
  Dim oWordApplication
  Set oWordApplication = CreateObject("Word.Application")
      Dim oWordDocument
      Set oWordDocument = oWordApplication.Documents.Add
' Enter your code here
mgoContext.GetRoot.SaveWordDocumentAsRTF oWordDocument,
strResultRtf
  oWordApplication.Quit
End  Sub
```

SaveWordDocumentAsRtf is an _Operator that enables Word document content backup in RTF.

Taking inheritance into account in macros

Regarding inheritance, you can:

- explicitly request by code the inclusion of inherited objects
- specify a behavior for the macro by fixing its execution context

To specify behavior at macro level:

1. In the macro properties, **Characteristics** tab, in the **_ExecutionOptions** field, specify the execution options:
   - not specified: the macro does not take inheritance into account
   - "Variation Inheritance": the macro takes inheritance into account
   - "Inherits caller options": the macro behaves like the caller macro

When you choose to take inheritance into account, the getCollections and getSelections functions of the macro include inherited objects.

Behavior regarding inheritance is the same for the following two expressions:

- myApplication.getCollection("Service")
- myApplication.getCollection("Service", " @inherited ")

> ☛ *In macros, even if the **Variation Inheritance** execution option is selected, the "Inheriting" parameter (which shows objects that are inheriting) is deactivated. To take this parameter into account, it must be indicated specifically in the getCollection (see API documentation).*

Adding a Diagram

You can define:

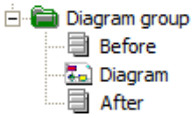
- Report (MS Word) diagram presentation
- Report (MS Word) diagram orientation

To add a diagram:

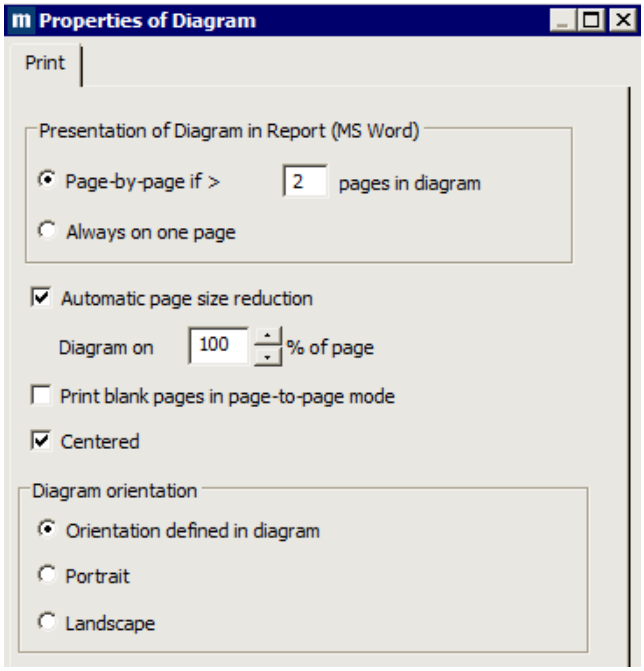
1. Add a new group.

   ☛ *See "Creating Groups", page 46.*

2. From the **Properties of a group** dialog box:
   - in the **Described object** field select "Diagram"
   - in the **Query of MetaAssociationEnd** select "Diagram".
3. Click **OK**.
4. Right-click the group and select **Insert > Diagram**.
   The icon representing a diagram appears in the tree.



5. Right-click the diagram icon and select **Properties**.



6. In the **Presentation of Diagram in Report (MS Word)** you can define if you want your diagram to be displayed on one or several pages in the generated report (MS Word):
   - By default, the **Page by page** option is selected if your drawing needs more than 2 pages. You can modify the setting for the number of pages required to activate this option.
   - The **All Diagram on one Page** option scales the entire drawing to fit on a single page. In this case the size is automatically reduced.
7. You can reduce the diagram size (in **% of page**).
8. Select the **Print blank pages in page-to-page mode** option so that unused pages are inserted in the diagram.

9. By default the diagram is centered, unselect **Centered** if you do not want the diagram to be centered.
10. In the **Diagram orientation** pane, define the orientation of the diagram at generation.
    - **Portrait** for a vertically oriented diagram
    - **Landscape** for a horizontally oriented diagram
    - **Orientation defined in diagram** for a diagram oriented as specified in the diagram editor (**Diagram > Page Setup**).

Adding existing descriptors

To add an existing descriptor to the descriptor you are currently building:

1. Right-click the group in which the descriptor is to be inserted, and select **Insert > Descriptor**.
   The dialog box proposes all descriptors that relate to the same object type.

   ```
   For example if you want to add an existing descriptor from
   the "Org-Unit" group, all descriptors with "Org-Unit" entry
   point can be selected.
   ```

2. Select a descriptor and click **OK**.
   The added descriptor appears in the descriptor tree.

   ☛ *In our example, we added a descriptor directly under "Org-Unit". If you want descriptors corresponding to another type of object to appear, such as "Message", you must first create a group corresponding to this object type and then add a descriptor under this group.*

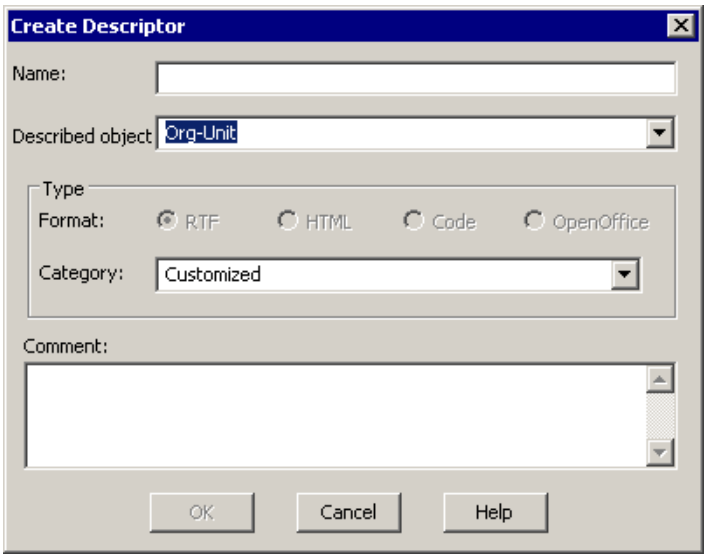
To disconnect it from your descriptor, right-click the added descriptor and select **Disconnect**.

☛ *Introduction (**Before**) and Conclusion (**After**) texts of a descriptor are not executed if the descriptor is called from another descriptor.*

Creating descriptors directly in the descriptor editor tree

To create a descriptor directly from the descriptor editor tree:

1. Right-click the descriptor root object type or group and select **New > Descriptor**.
   The **Create Descriptor** dialog box opens.
   The descriptor can relate to the root object (for example: "Org-Unit"), or to the abstract MetaClass from which the root object is derived.



For more details, see "Customizing RTF Descriptors", page 41.

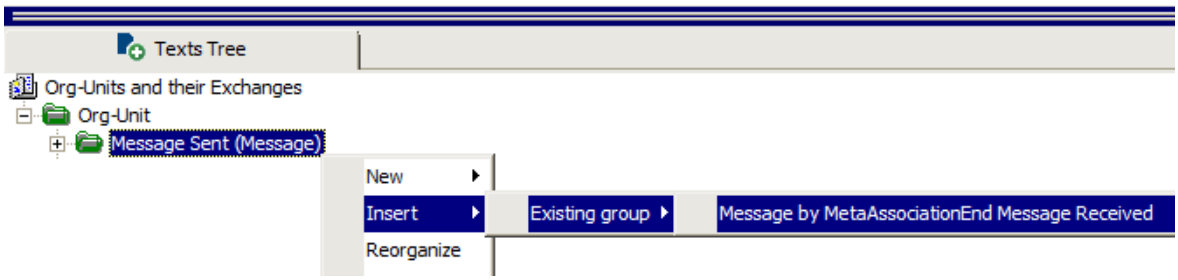
Using Groups

Reusing an existing group

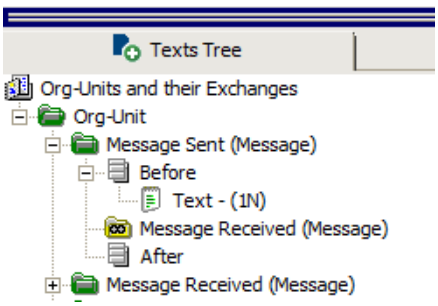
You can reuse an existing group in your descriptor if this group relates to the same object type.

To reuse an existing group:

1. Right-click the group that relates to the same object type as the group you want to add and select **Insert > Existing Group** then the name of the group you want to add.
   Groups relating to the same object type are proposed.



The group connected in this way remains linked to the reference group. An icon is added to the tree structure, but you cannot modify these elements (you can modify the group in which it was initially created).



Deleting a Group

To delete a group:

- Right-click the group and select **Delete**.
  All dependent elements are deleted, except for the connected groups and descriptors, which remain in their initial location.

  ☛ *When in doubt, you can begin by deleting the texts and then check the results before deleting the entire group.*

Modifying a group properties

To modify a group properties:

1. Right-click the group and select **Properties**.
2. In the group properties you can modify the query.
3. Click **OK**.

DEFINING THE OBJECT ORDER IN REPORTS (MS WORD)

To sort objects within a group ***in the generated report (MS Word)***:

1. In a group properties, click **Sort**.
   The Sort dialog box of the group opens.
   The default criteria are repository "Order" and "Short Name".
2. To add or remove a sort criterion:
   - in the **Characteristics** list select a characteristic and click **Add** to add a sorting criterion
   - in the **Criteria** list, select a characteristic and click **Remove** to remove a sorting criterion.
3. To order the sort criteria, right-click a characteristic and select **Up** or **Down** as required.
4. If needed, in the **Criteria** list, select an item and in the **Direction** field define the sorting direction ("Ascending" or "Descending").

Examples

- To sort the group objects in alphabetical order, place the "Name" criterion at the top of the **Criteria** list and select **Ascending** order.
- To group the objects by creator, add "Creator Name" characteristic in the **Criteria** list and place it at the top.

To modify the order of objects in reports (MS Word):

1. Check that the descriptor used in your report (MS Word) contains groups relating to a MetaAssociationEnd and not to a query.
   The group should be colored green (not blue).
2. In the diagram, click **Tools > Order**.

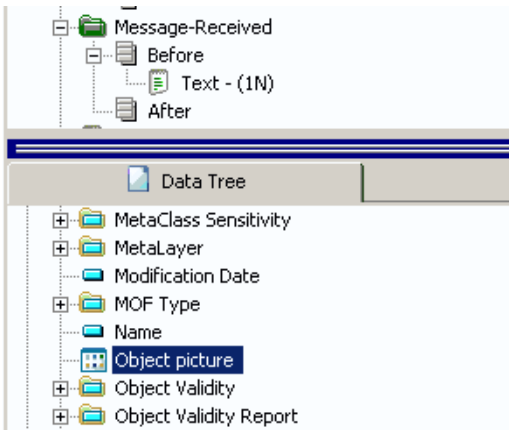
If you want to reproduce the previously defined order in a diagram, you must keep the "Order" criterion at the top of the list. The order can be kept only for green-colored groups, which relate to a MetaAssociationEnd (it cannot be kept for blue-colored groups, which relate to a query).

MODIFYING TEXT

The following points are detailed here:

Adding the Image of an Object

The "object image" characteristic allows you to obtain the object image in the generated report (MS Word). This characteristic is available for each object.



In the descriptor editor, the <Object image/> tag is used for this purpose.

Use of this tag simplifies multilingual management by making object type naming optional.

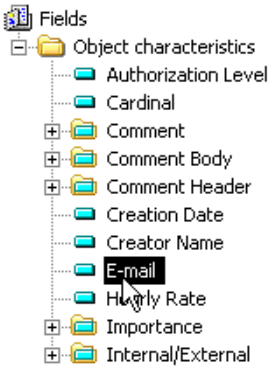
Showing Metamodel Names

You can show metamodel characteristic and link names and object type names in different languages.

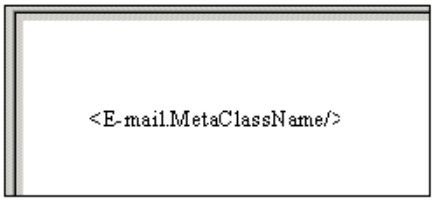
Showing a characteristic name

To show a characteristic name:

1. In the tree, select the icon corresponding to the object characteristic and simultaneously press the <Ctrl> key.



2. Holding the <Ctrl> key down, drag the characteristic name to the text area.
<Name.MetaClassName> appears in the text area. In the report (MS Word), "Name" appears in the current language.



Showing the object type

To show an object type name:

1. In the "Object Characteristics" folder, select "Object MetaClass Name" and simultaneously press the <Ctrl> key.
2. Holding the <Ctrl> key down, drag-and-drop "Object MetaClass Name" to the text zone.
<Object MetaClass Name/> appears in the text area. In the report (MS Word), the object type name will appear in the current language.

Summary regarding metamodel names

<Name>	returns object name
<Name.MetaClassName>	returns "Name" in current language
<MetaClassName>	returns object (MetaClass) type name in current language

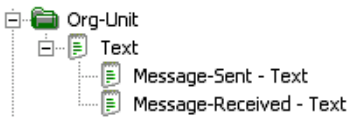
Use of names derived from the metamodel simplifies multilingual management. It avoids multiplication of texts, simplifying maintenance and reducing repository size.

Component Drag-and-Drop

From the descriptor editor you can drag-and-drop object components so as to obtain sub-texts carrying the MetaAssociationEnd name.

To create sub-texts:

1. Drag-and-drop components for example "Message-Sent" and "Message-Received" into the text body of the "Org-Unit" group.



2. In each of the sub-texts, drag-and-drop the "Name" characteristic.
3. Format the text under the group, as below:

<Name/>

<Comment/>

<Message-Sent.MetaClassName/>	<Message-Received.MetaClassName/>
<Message-Sent><Name/></Message-Sent>	<Message-Received><Name/></Message-Received>

☛ *In the properties dialog box of the component texts created, select the **Format** tab, then the **New Line** option.*

This method can prove simpler than adding groups, as described in "Using Groups", page 63, but it does have limitations. For example, it is not possible to specify text execution conditions as described in "Text editing conditions", page 69.

Taking inheritance into account in texts with components

If the general option of variations is activated, you can choose to:

- follow default behavior of the descriptor
- or activate/deactivate inheritance

To specify behavior of text regarding inheritance:

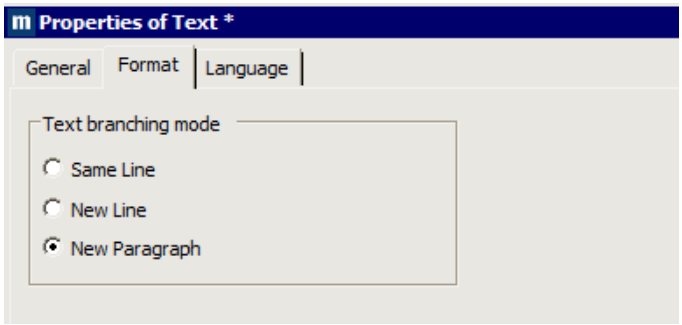
- Select or clear the corresponding check box in the text properties.

  ☛ *The box should remain grayed to follow descriptor default behavior.*

Defining Text Formats

To define a text format:

1. From the descriptor, right-click the text and select **Properties**.
2. Click **Format** tab.



3. Specify how sequentially linked texts should appear ("text branching mode"), select:
   - **Same Line** for the selected text to appear on the same line as the previous text.
   - **New Line** for the selected text to start a new line without insertion of paragraph spacing between the selected text and the previous text.
   - **New Paragraph** (default) for the selected text to start a new paragraph. This option inserts paragraph spacing between the selected text and the previous text.

Specifying Text Language (Multilingual Context)

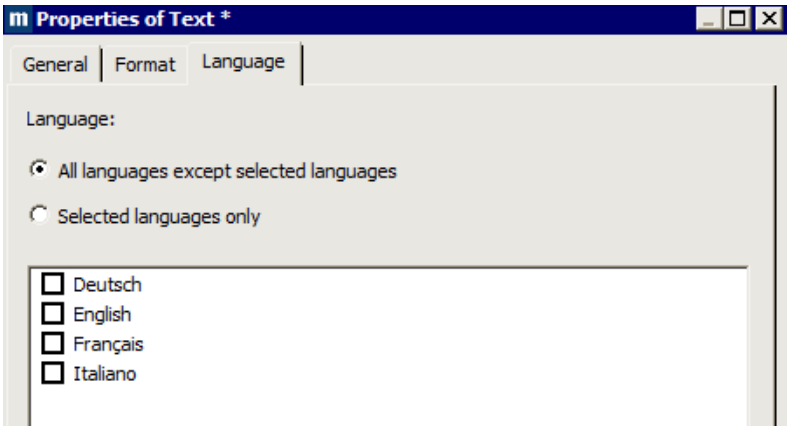
You can edit texts in several languages.

Text editing conditions

To specify in which language text will be edited:

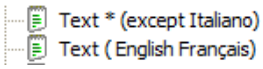
1. From the descriptor, right-click the text and select **Properties**.
2. Select the **Language** tab.
   By default, a text is executed in all languages.

3. Select or remove one or several of the environment languages.



4. To restrict or extend the choice of languages, you can:
   - specify the languages you want (**Selected languages only** option).
   - exclude unwanted languages (**All languages except selected languages** option).

   In the descriptor menu tree, the languages applied to text are indicated alongside its icon. "*" indicates that all languages have been selected.



☺ *If you edit the same text in several languages, it can be useful to provide a text with the property **All languages except**. If you add languages at a later stage, all the languages can thus be processed without the risk of any being forgotten.*

Data derived exclusively from MEGA

When you edit texts that contain **MEGA** data exclusively (not texts that you yourself have entered), it is of no great value to duplicate these texts for each language.

```
Example: enter <&Name&> to edit the name in the current
language (which is not the case if you enter "&Name
(English)&").
```

☛ *See also "Showing Metamodel Names", page 66.*

Inserting Tables into Text

If you want your descriptor to appear in tabular form, insert a table into all descriptor texts.

To do this:

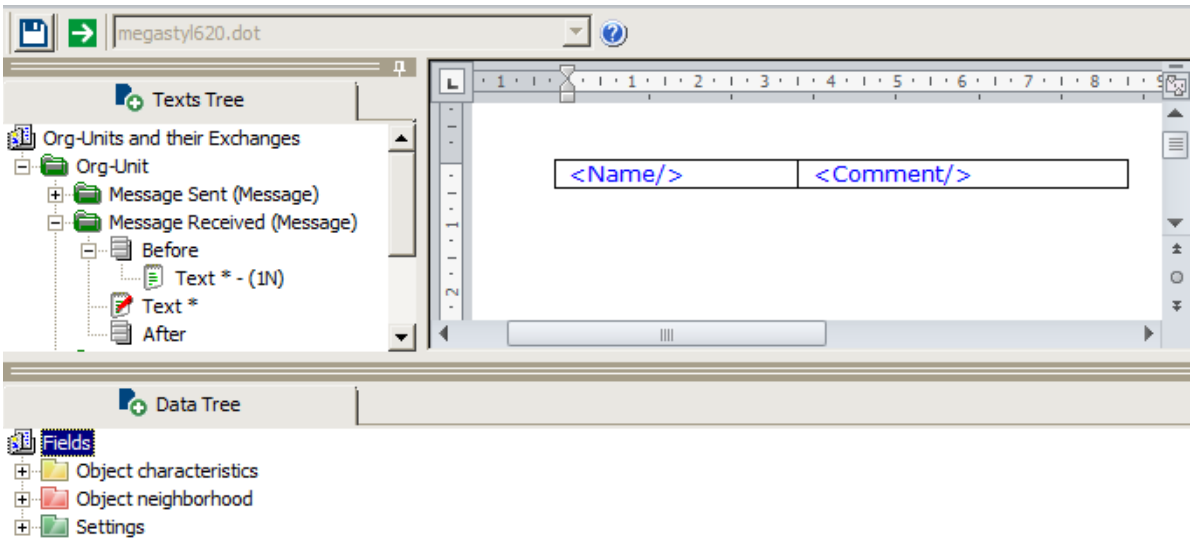
1. Access the descriptor structure.

   ☛ *See "Accessing the descriptor structure", page 44.*

2. Edit the group text.

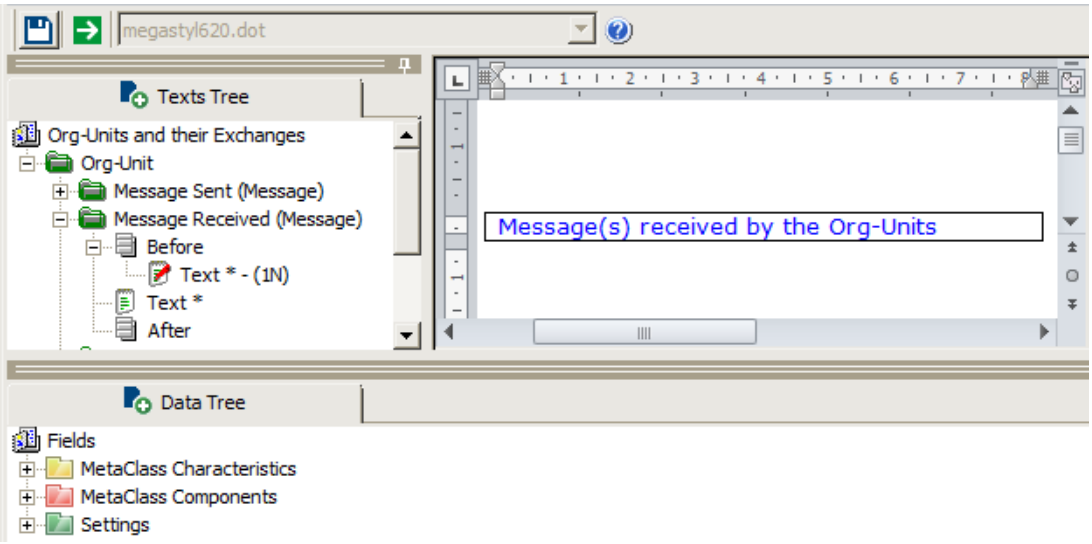
   ```
   Exemple: "Message-Received" group
   ```

3. From Word menu bar, insert a table and set up the text fields in table format.
4. Format this text with the “M-Table” style.



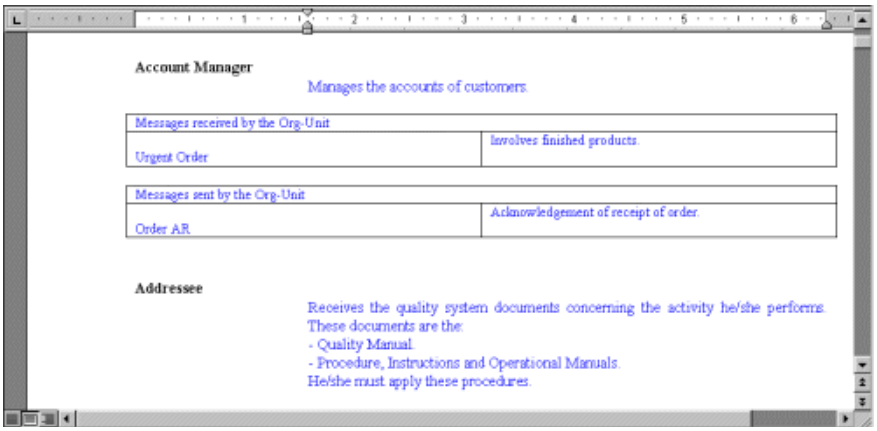
5. Create a table for the "Before" text. Insert a one-row table and enter text describing the contents of the table and format this text with the "M-Table" style.

   `Example: text such as "Message(s) received by the Org-Unit"`



6. Right-click the "Text" you modified previously, and select **Properties** to specify the text branching mode.
7. Select the **Format** tab and select **Same Line**, so that there is no break between the tables.
8. Proceed similarly with "Message by the Message-Sent MetaAssociationEnd" group, but copy the tables you just created so that presentation is consistent.

After processing the descriptor, a result similar to the following is displayed:



☛ *By default, Word 2000 adjusts the columns of the table to fit their contents. For improved legibility of tables generated using **MEGA**, create the first line of the table in the descriptor editor, then clear the **Automatically resize to fit contents** check box in the table options*

*(in Word, menu **Table > Table Properties**, **Table** tab, **Options** button).*

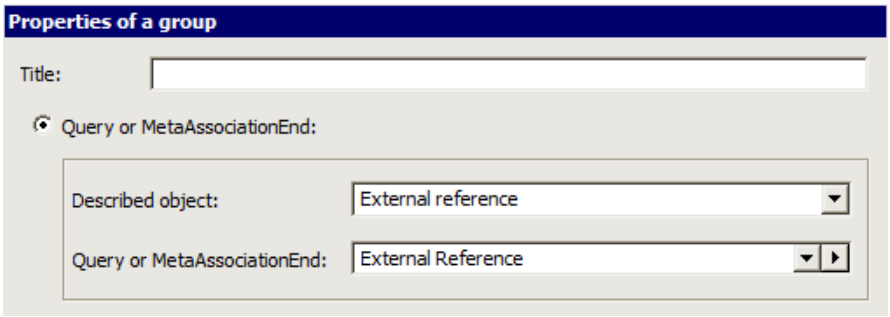
Adding External References in a Descriptor

If you have created external references for certain repository objects, you can include fields corresponding to these references in your descriptors to quote them in generated reports (MS Word).

```
In the "Org-Units and their exchanges" descriptor that you
created earlier, add a group enabling creation of a link to
external references associated with the messages sent.
```

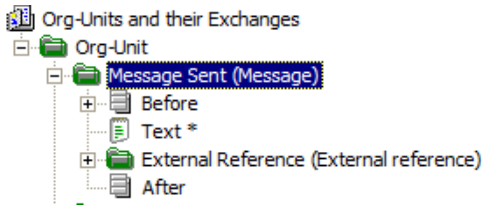
To do add an external reference in a descriptor:

1. in the group (example: "Message-Sent") add a new group.
   The **Group Properties** dialog box opens.



2. Select **Repository query or leg** and:
   - In the **Described object** field select "External reference".
   - In the **Query or leg** field select "External reference".

     ☛ *Else select **Embedded query** and enter: Select [External reference] where [Message]= "&Message"*
3. Click **OK**.
   An **External Reference** folder is added in the tree.

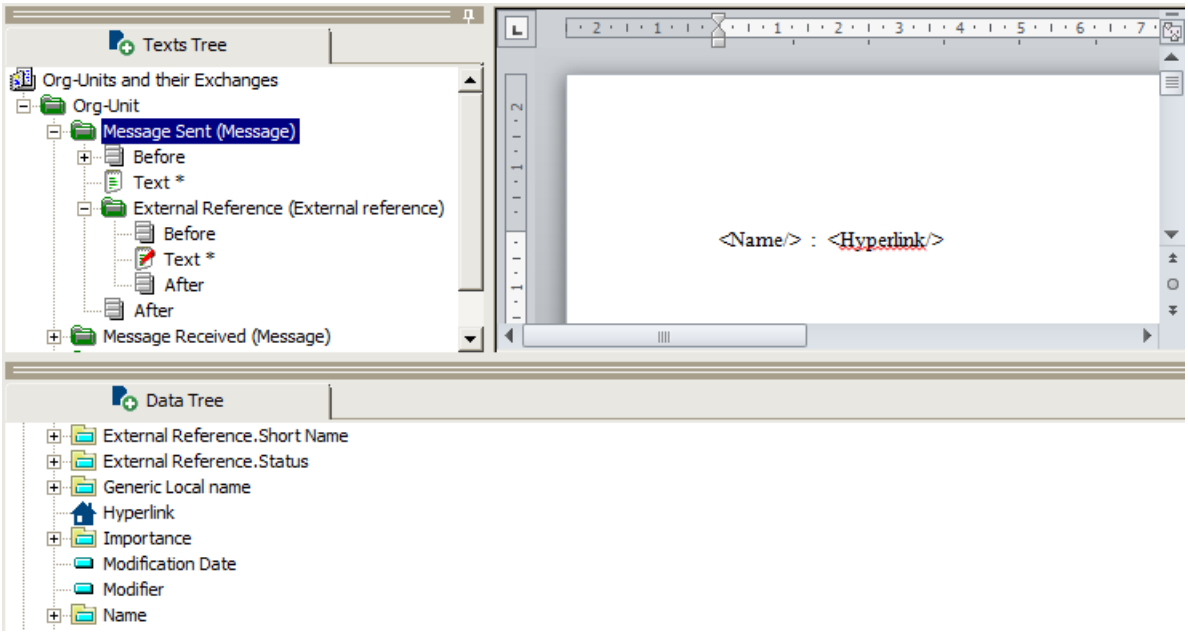


4. Add a text to this group.

   ☛ *See "Adding Text to a Descriptor", page 44.*

   An empty report (MS Word) appears.

5. Add the "Name" and "Hyperlink" fields (drag and drop).



In the report (MS Word) obtained after processing the descriptor:

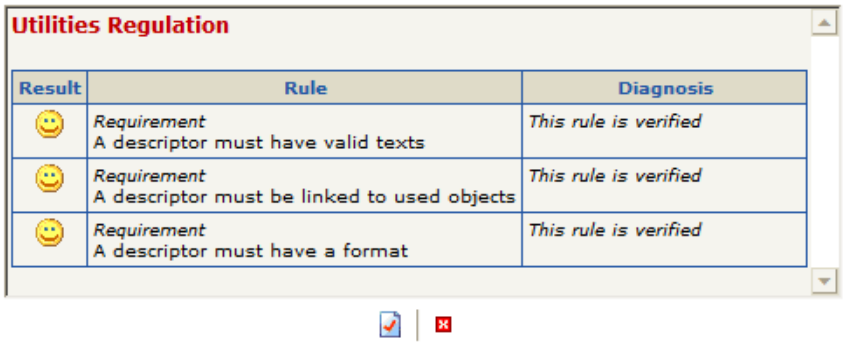
- The "Name" field contains the name you gave to the external reference.
- The "Hyperlink" field indicates either the file path or the URL address, depending on the type of external reference.

> ☛ *The content of the external reference cannot be displayed using this method. To do this, you must use a macro. For more details, see "Using Macros", page 58.*

CHECKING DESCRIPTOR VALIDITY

To check validity of a descriptor

1. In the descriptor editor menu, select **MEGA > Tools > Check**.
2. Select the **Modeling Regulation**.
3. Click **OK**.
   Errors are displayed. The incorrect field is indicated for each text concerned.



☛ *The **Check** menu is available on descriptors, report templates (MS Word) and reports (MS Word).*

4

MANAGING FORMATS AND STYLES

The following points are covered here:

☛ *The description of styles in this chapter is useful for those who have the **HOPEX Studio** technical module.*

REPRODUCING RTF TEXT FORMATS

As a reminder, **MEGA** RTF texts enable formatting of:

- characters (bold, underline, color...)
- paragraphs (bullets, indents, etc.).

When you generate a report (MS Word) or Web site, you generally seek to obtain a consistent result (same font, same character size, same paragraph justification, etc.) throughout the report (MS Word) or HTML page, or between different documents.

Similarly, you will usually want to standardize formatting of comments entered by different users.

You may however want to reproduce texts exactly as they have been entered by users in the RTF text editor.

Permissive and restrictive policies

For these reasons, **MEGA** proposes two policies:

- a restrictive policy (default policy), which enables standardization of formatting. The aim is to ensure consistency of documentation.

  ☛ *For more details, see "Standardizing Formatting", page 79.*

  In short, styles take precedence over formats introduced in the RTF text editor.
- a permissive policy, which enables to keep formats specified in the RTF text editor when entering texts.

  ☛ *For more details, see "Keeping Formatting Applied in the Text Editor", page 80.*

Recommendations

MEGA recommends that formats be standardized (restrictive policy).

For more information on how to configure reproduction of formatting, see:

- "Standardizing formatting", page 80
- "How to keep RTF text formatting", page 81.

Reproducing RTF text formats

The following tables indicate how text formats are reproduced in the RTF text editor when:

- outputs are standardized
- RTF text formats are kept.

Standardizing Formatting

Reproducing character formatting

The following table indicates how each format is reproduced when generating a report (MS Word) or Web site.

Formatting	Reproduction in report (MS Word)	Reproduction in Web site
Bold	Yes	Yes
Italic	Yes	Yes
Underline	Yes	Yes
Strikethrough	Yes	Yes
Font	No	No
Font size	No	No
MEGA fields	Object name in bold	If objects produce pages, fields are reproduced in the form of links. If not, only the object name is reproduced.

Standardizing formatting - characters

☛ *Formatting of texts generated in tables is ignored in reports (MS Word).*

Reproducing paragraph formatting

Paragraph formatting applies to the complete RTF text, unlike character formatting.

Formatting	Reproduction in report (MS Word)	Reproduction in Web site
Alignment	No	No
Indent	The indent is added to that of the style	The indent is added to that of the style
Bullets	Yes	Yes

Standardizing formatting - paragraphs

☛ *Formatting of texts generated in tables is ignored in reports (MS Word).*

Standardizing formatting

You must indicate the generation mode selected for each Web site template and report template (MS Word).

To standardize formatting in report (MS Word):

1. Access the Report Template (MS Word) properties.
2. From the **Report Template (MS Word)** tab, in the **Text Generation Mode** field, select "Restrictive policy".

To standardize formatting in Web sites:

1. Access the Web Site Template properties.
2. In the **Generation** tab, clear the **RTF font enabled** option.

Keeping Formatting Applied in the Text Editor

You can keep formatting applied to comments entered in the RTF text editor by the user. This relates to formatting applied via the toolbar in the object comment dialog box (bold, italic, underlined, etc.)

This method is not however recommended. Distortions may appear in the ergonomics of inputs when this formatting is added to styles defined in the formatter.

The following table indicates if RTF text formatting is reproduced:

Character formatting reproduction

Formatting	Reproduction in report (MS Word)	Reproduction in Web site
Bold	Yes	Yes
Italic	Yes	Yes
Underline	Yes	Yes
Strikethrough	Yes	Yes
Font	Yes	Yes
Font size	Yes	No
MEGA fields	Object name in bold	If objects produce pages, fields are reproduced in the form of links.

Retaining formatting - characters

☛ *Formatting of texts generated in tables is ignored in reports (MS Word).*

Reproducing paragraph formatting

Formatting	Reproduction in report (MS Word)	Reproduction in Web site
Alignment	Yes	Yes
Indent	The indent is added to that of the style	The indent is added to that of the style
Bullets	Yes	Yes

Retaining formatting - paragraphs

☛ *Formatting of texts generated in tables is ignored in reports (MS Word).*

How to keep RTF text formatting

You must indicate the generation mode selected for each Web site template and report template (MS Word).

To keep RTF text formatting in report templates (MS Word):

1. Access the Report Template (MS Word) properties.
2. In the **Text Generation Mode**, select "Permissive policy".

To keep RTF text formatting in Web sites:

1. Access the Web Site Template properties.
2. in the **Generation** tab, select the **RTF font enabled** option.

Formatting and Objects Not Supported by MEGA

The following formats are not supported by **MEGA**, whether in RTF descriptors executed, reports (MS Word), or Web sites:

- Tables
- Pictures
- Word styles (obtained by copy/paste of a Word document)

CUSTOMIZING STYLES

Presentation

Word styles allow you to format object descriptors and reports (MS Word). The style names indicated in object descriptors are included when inserting the descriptors into reports (MS Word) or report templates (MS Word).

Styles

A style is a combination of formats that you can apply to a paragraph of text in a word processor. Styles allow you to systematically apply formats such as fonts, margins, and indents.

MEGA reports (MS Word) include styles. The names of these styles are prefixed with "M-". These styles are based on the "M - Normal" style, which is identical to the MS-Word "Normal" style. The only difference is the character color, which is blue.

☺ *To modify the font used for all the "M -" styles, modify the font of "M - Normal".*

Style sheet

The styles (character and paragraph format, etc.) are defined in a style sheet (which is a Word document template).

Default style sheet

The default style sheet used by report templates (MS Word) and object descriptors is "Megastyl620.dot". It is located in the Mega_Std folder. It is recommended to use this style sheet.

☛ *For more information, see "Customizing the default style sheet", page 84.*

Specifying a style sheet

To change the default style sheet:

1. From **MEGA** menu bar, select **Tools > Options**.
2. Expand the **Documentation** folder.
3. Select **Reports (MS Word)** and in **MEGA style sheet** field indicate the name of the document template you want to use as the document template style sheet.

Use of the same style sheet is recommended for descriptors, report templates (MS Word) and reports (MS Word).

Structure of MEGA Styles

MEGA descriptor styles are most of the time organized logically. The style names indicate their purpose, independently of their format.

```
For example, diagrams are style "M-Drawing".
```

To print the description of styles:

1. in Word, open the document template used (default: “Megastyl620.dot” in the MEGA_STD folder).
2. From the **File** menu, select **Print**.
3. In the **Settings / Document properties** section, select **Styles**. The styles used in the style sheets will be printed.

Creating and Modifying Styles

If a style format does not suit you, instead of applying another style you should modify the style provided. This way you will not have to open all the descriptors to change the style name.

Modifying a style

To modify a style:

1. Open the style sheet ("xx.dot", the default being "Megastyl620.dot")
2. Save it in the “MEGA_USR” folder of the environment.
3. Modify the style.

Remarks on modification of styles

The default style sheet “Megastyl620.dot” is write-protected in the MEGA_STD folder. You can save another style sheet with the same name or create a new one in the environment MEGA_USR folder.

> ☛ *Copy a style sheet to the “MEGA_STD” folder only if you need to share it between several environments.*

Styles are defined hierarchically.

```
For example, “M-Sublevel 2” is based on “M-Sublevel 1”.
```

This is why modifications made to one style may impact other styles.

You can modify a style without having to modify the object descriptors; all descriptors whose text uses this style will include the modification. To ensure that modifications to a style are taken into account, they have to be included in the style sheet defined in the user options. All reports (MS Word) defined using report templates (MS Word) or descriptors based on the style sheet will then be modified accordingly.

To modify the orientation (portrait/landscape) or the margins of generated reports (MS Word):

- Modify the document template directly.

Parameterizing the report templates (MS Word) allows you to define which style sheet should be used when creating a report (MS Word) based on this report

template (MS Word). The user can modify the template using standard MS-Word template and style manipulation functions.

Use of frames is not recommended in definition of styles.

Creating a new style

You can create a new style that can be used in the texts of object descriptors. When you create a new style, you have to add it to the "Megastyl620.dot" style sheet. It then becomes available to the object descriptors.

Never insert the style directly into the descriptor. To ensure that the style is properly defined in generated reports (MS Word), it must be included in the style sheet specified in the user options. Avoid using the same style in the body of the text and in a table. If you later reduce the size of this style, cells using this style could become so small they are illegible.

Attaching a style sheet

You can attach a style sheet to an environment or to a user:

- The style sheet attached to the environment will be the default for each user you create.
- The style sheet attached to a user will be the default when the user creates a new report template (MS Word) or descriptor.

It is not recommended to attach a style sheet to a report (MS Word), a report template (MS Word) or a descriptor. If ever you do so, you will have to open all texts of your descriptor and save them so that the new styles are taken into account.

☛ *After creation, the report (MS Word) is independent and modifications made to the style sheet are not automatically reflected in the report (MS Word)*

☛ *Styles taken into account in a report (MS Word) created from a report template (MS Word) are those contained in this report template (MS Word). Only the names of styles defined in a descriptor are included in the report (MS Word). The details for the styles defined in the descriptor are replaced by those inherited from the report template (MS Word).*

When a style used in a descriptor does not exist in the report template (MS Word) being used, it is replaced by the "Normal" style. For this reason, it is recommended that you create style sheets using "Megastyl620.dot" as the template.

You can define or modify a style by testing it in a descriptor. However, to make this new style or modification available to new report templates (MS Word), you should also define or modify the style in the style sheet attached.

Customizing the default style sheet

It is advised to use the Megastyl620.dot syle sheet provided.

To customize this style sheet:

1. Copy the reference style sheet ("Megastyl620.dot") to the MEGA_USR folder of your work environment.
2. Open the style sheet with MS Word, without running **MEGA**.
3. Make the desired style modifications or create a new style.

4. Using **MEGA**, attach the style sheet to the user, so that it becomes the default style sheet for descriptors.

   ☛ *Attach the style sheet by indicating this style sheet in the **Documentation > Reports (MS Word)** options.*

Description of Default Styles (MegaStyl620.dot)

Here are some of the styles provided in the default style sheet.

Titles

Title 1	For chapter titles, it is numbered and automatically inserts a page break before.
Title 0	Identical to Title 1, but does not include numbering or page break. For contents table and distribution list.
Title 2	Defines the various sub-chapters. It is centered and does not include numbering.
Title 3	Italic and does not include numbering.
M - Document Title	Used on front page. It is centered, in font 26. It takes the report (MS Word) title.
M - Document Title 2	Used on front page. It is centered, in font 22. It is used to supplement the title if necessary.
M - Sublevel ***M - Sublevel 2*** ***M - Sublevel 3***	Enable creation of additional paragraph titles that do not however appear in the contents table.

Texts

M - Comment	Text default format.
M - List	Style used for lists. A bullet precedes the beginning of the line.
M - Remark	Style used for a comment preceded by a special bullet. Can be used to indicate remarks in a report (MS Word).

Tables

M - Table	Default format for a cell.
M - Table Center	Format used to center text in cells.
M - TableHeading	Title cell format, centered with gray fill.

Drawing

M - Drawing Style for diagrams.

Contents table

TM1 Contents table line style connecting to "Title 1".

TM2 Contents table line style connecting to "Title 2".

Index

Index title Index title style

Index 1 Generated index data style.

9

CUSTOMIZING BUSINESS DOCUMENTS

MEGA lets you store, classify, reference and update documents whose contents are independent of the repository. Business documents follow particular storage and classification rules; this chapter presents how to customize these.

☛ *For a presentation of business documents, see chapter "Generating Documentation" in **MEGA Common Features** guide.*

MANAGING BUSINESS DOCUMENTS

Prerequisites

To use customization functions of business documents, you must have available certain products or options.
Below is a summary of functions available with each product or option.

Product/Option	Functions
HOPEX Productivity Pack	Creating and using document management categories and models
MEGA Administration - Supervisor	Accessing confidentiality areas
HOPEX Studio	Defining documentable objects Customizing macros
Products or solutions	Accessing documentable objects.

☛ *Solutions such as **HOPEX ERM** and **HOPEX Internal Audit** include the **HOPEX Productivity Pack** to manage attachments.*

Defining the Business Document Storage Directory

The files associated with different business document versions are stored by **MEGA** in the storage directory in ".dat" format.

📖 *The **Storage Directory** is the directory in which business documents of the application are stored.*

The .dat files correspond to contents of documents downloaded in **MEGA**, they cannot be used outside **MEGA**. These files do however remain accessible, and they can be encrypted to prevent reading of their content. Encryption method depends on the document model used. See "Creating a Business Document Pattern", page 180.

The **Storage Directory** is the directory in which business documents of the application are stored.

To define the default storage directory:

1. Start the **MEGA** "Administration.exe" application and connect with a user that has data administration authorization rights:
2. Right-click **MEGA** and select **Options > Modify**.
   The options window appears.
3. In the tree on the left, expand the **Installation** folder and select **Advanced**.

4. In the right pane, the **External files storage path** field specifies the address of the business document storage directory.

   ☛ *The address must respect UNC convention.*

5. Click **OK**.

   ☛ *User access rights to the directory are managed by the operating system (and not by the application).*

CLASSIFYING BUSINESS DOCUMENTS

Business documents can be classified by category.

Certain documents may be confidential and their access, even in read-only, is restricted to certain profiles. The notion of document models allows the end user to access only those business documents authorized for his/her profile..

Creating a Business Document Category

A category enables classification of documents according to standard subjects such as Audit, Diagram, Control Documentation. Categories are organized hierarchically from these main types.

Business document categories are hierarchical.

To create a category:

1. Open the **Utilities** navigation window.
2. Select the **Document Categories** folder and click **New > Document Category**,
3. Specify the name of the new **document category**,
4. Click **OK**.
   A new folder with the category name is created.

Creating a Business Document Pattern

The business document pattern predefines the category, confidentiality area and encryption associated with a new document.

> 🕮 *The **confidentiality area** assembles a group of **MEGA** objects. The confidentiality area can be associated with **MEGA** profiles. This means that users can see not only **MEGA** objects defined at confidentiality area level, but also **MEGA** objects associated with child confidentiality areas. The user cannot access objects defined in a parent confidentiality area.*
>
> ☛ *Use of confidentiality areas is only proposed with a **MEGA Administration - Supervisor** license. For more information, see the **MEGA Administration** guide.*

To create business document patterns, you must access the **MEGA** repository in Advanced mode.

To access the repository in advanced mode:

1. Select **Tools** > **Options**.
   The options window appears.
2. In the tree on the left, select **Repository**.
3. In the right pane in **Metamodel Access**, select "Advanced".
4. Click **OK**.

To create a business document template pattern:

1. Open the **Utilities** navigation window.

2. Right-click the **Business Document Patterns** folder and select **New > Business Document Pattern**.
3. Specify the name of the **Business Document Pattern**,
4. Select the **Encrypted** check box so that documents will be encrypted by **MEGA** at storage.
5. Select the **confidentiality area of the system object**.
6. Click **OK**.
   A new folder with the name of the business document pattern is created.

Defining Business Document Objects

By default, a certain number of MetaClasses can own or be connected to a business document. They inherit the "Element with business document" abstract MetaClass

> ☛ *Access to the **MEGA** metamodel is only possible with a **MEGA Studio** license.*
>
> ☛ *The list of MetaClasses inherited by a MetaClass is accessible from the properties dialog box of the MetaClass, in the **Characteristics** tab, **Standard** subtab. The **SuperMetaClass** field allows you to view and update this list.*

You can extend business document ownership or referencing possibilities to other MetaClasses.

To add the possibility of connecting business documents to Applications, for example:

1. Right-click "Element with business document" abstract MetaClass and select **Explore**.
   An explorer window opens.
2. Expand the "SubMetaClass" folder.
3. Connect this folder to the **Application** MetaClass.

MODIFYING DOCUMENT BEHAVIOR

You can customize opening and closing behavior of files containing business document data.

Business document customization is carried out by attaching a **Regeneration Macro**.

☛ *Use of **MEGA** macros is proposed with a license **HOPEX Studio**.*

To use the VB macro creation wizard on an object of **Business Document** type:

1. Select the **Business Document** from the explorer.

   ☛ *For more information on using the explorer, see "Exploring the Repository" in the **MEGA Common Features** guide.*
2. Display "Empty Collections".
3. Right-click the "Regeneration Macros" folder and select **New**.
   The macro creation wizard opens.
4. Select the check box corresponding to the type of macro you want to use, for example:
   - "(VB) Script macro"
   - "Existing macro": to use an existing macro, of which field **Reusable** is selected.
5. Click **Next**.
6. Enter the **Name** of the macro.
7. If the macro you have created can be reused, for another **Business Document**, select the **Reusable** box.
8. Click **Finish**.

   ☛ *You can create a macro from the Script editor. It is created in folder "Macro VBS > Unclassified Macros" with its VB Script code empty.*

When you define behavior in the macro code, note that:

- The input document should be stored in a location different from the output document.
- If several macros are created, they are sequenced.

GLOSSARY

absolute identifier

An absolute identifier is the string of characters associated with each repository object. This string is computed using the date and time the session was opened, the number of objects created since the session started, and the object creation date (in milliseconds). An absolute identifier provides a unique way to identify an object in the repository, so that even if the object is renamed, all the links to it are retained.

administrator

A user with full access rights. He/she can administer repositories using the administration application, and can create and modify elements such as descriptors and report templates (MS Word).

attribute

See *characteristic*.

characteristic

A characteristic is an attribute that describes an object type or a link. Examples: The Flow-Type characteristic of a message allows you to specify whether this message is information, or a material or financial flow. The Predicate characteristic of the Send link between message and org-unit allows you to specify the condition for sending the message. A characteristic can also be called an Attribute.

descriptor

Descriptors allow you to create reports (MS Word) containing part of the contents of the repository. A descriptor for an object includes the characteristics of the object to which can be added the characteristics of objects directly or indirectly linked to it. The readable format for each of the objects encountered is entered as text in Word for Windows. You can insert descriptors into reports (MS Word) or report templates (MS Word) or use them to produce reports. Descriptors can be created or modified using the **HOPEX Studio** technical module.

external reference

An external reference allows association of an object with a document from a source outside **MEGA**. This can relate to regulations concerning safety or the environment, legal text, etc. Location of this document can be indicated as a file path or Web page address via its URL (Universal Resource Locator).

function

A function is a feature supplied by the software to carry out certain actions. Among functions available as standard are the shapes editor and the descriptor editor.

group

Descriptors, which are available with the **HOPEX Studio** technical module, comprise a tree of several successive groups. Each group concerns one object, and defines the query or link used to access this object from the preceding object. You can connect texts and other groups to a group. Users can define the order in which groups and texts are processed.

link

A link is an association between two object types. Several types of link can exist between two types of object. Example: Source and Target between Org-Unit and Message.

matrix

A matrix is a table consisting of rows and columns containing objects from the repository. Matrices show the links between two sets of objects, and can be used to create or delete links without it being necessary for you to open the diagrams containing these objects. For example, you can build a matrix showing the messages sent by the different org-units in a project.

MetaClass

see object type.

metamodel

The metamodel defines the structure used to store the data managed in a repository. The metamodel is saved in the system repository of the environment. The metamodel contains all the object types used to model a system, as well as their attributes and the links possible between these object types. The metamodel therefore allows you to build diagrams that describe the organization or the information system of an enterprise. If desired, you can extend the metamodel to manage new types of object. Repositories that exchange data via exports, imports or extraction must have the same metamodel, otherwise certain data will be rejected or inaccessible. In metamodel definition language, object types are also called MetaClasses. Their characteristics are also called attributes.

model

A model is a formal structure which represents the organization of a company, or its information system. In another sense, a model can be a template for reproducing objects with similar characteristics. This is the case for report templates (MS Word).

object type

An object type (or MetaClass) is that part of the database containing objects of a given type. The objects you create are stored in the repository according to their type. This notion is used when searching for objects in the repository and when the metamodel is extended to include a new type of object. Example: message, org-unit, etc.

query

A query allows you to select a set of objects of the same type, using one or several selection criteria. Most of the MEGA software functions can handle these sets. For example, you can use a query to find the set of all enterprise org-units involved in a project.

reflexive link

A reflexive link is a link between two objects of the same type, for example: the link between projects that allows you to define sub-projects.

report (MS Word)

Reports (MS Word) managed by **MEGA** are objects allowing you to transfer written knowledge extracted from the data managed by the software.

report (MS Word) element

A report (MS Word) element is the instancing of a report template (MS Word) element. It is the result, formatted in the word processing software, of execution of the query defined in the report template (MS Word) element.

report template (MS Word)

A report template (MS Word) is a structure with characteristics that may be reproduced on production of reports (MS Word). A report (MS Word) can be created and modified as many times as necessary; however to produce several reports (MS Word) of the same type, use of a report template (MS Word) is recommended.

A report template (MS Word) provides the faramework for the report (MS Word), which is completed with data from the repository when the report (MS Word) is created. A template contains the formatting, footers, headers and text entered in MS Word. It also contains report template (MS Word) elements that allow you to format data from the repository. It allows you to produce reports (MS Word) associated with main objects of the repository.

report template (MS Word) element

A report template (MS Word) element is the basic element of a report template (MS Word). It comprises a query which enables specification of objects to be described and a descriptor for their formatting. When creating a report (MS Word) from a report template (MS Word), each report template (MS Word) element is instanced by a report (MS Word) element.

set

A set is a collection of elements with common characteristics. For example, you can build a set of messages sent or received by a certain org-unit in the enterprise. These sets are usually built using queries, and can be handled by most functions of the software.

setting

A setting is a parameter whose value is determined when the function with which it is associated is executed. You can use settings to condition a query (**HOPEX Studio** technical module). When you execute this query, a dialog box asks you to enter values for settings defined in the query.

style

A style is a combination of formats that you can apply to a paragraph of text in a word processor. Styles allow you to systematically apply formats such as fonts, margins, indents, etc. Several styles are available for report (MS Word) configuration. These styles have the M- prefix and are based on the M-Normal style that is similar to the Word Normal style. Note that the M-Normal style text is in blue. All these styles are contained in the Megastyl.dot style sheet.

style sheet

A style sheet groups the rules of presentation that can be applied to elements in a Web page. These rules are expressed in Cascading Style Sheet (CSS) language. For further information, consult the W3C (World Wide Web Consortium) recommendations (http://www.w3.org).

text

You can associate text with each object found when browsing object descriptors (**HOPEX Studio** technical module). This text is formatted for MS Word. It presents what will be displayed for each of the objects in the generated report (MS Word). In the text you can insert the object name, its characteristics, and its comment. You can also insert the characteristics of other objects linked to it.

ABOUT REPORT DATASET DEFINITION

Report DataSet enables to extract **MEGA** raw data and show it in tabular form. The **Report DataSet Definition** describes how to build the **Report DataSet**.

> ☛ ***Report DataSet Definition*** *is only available with* ***HOPEX Studio*** *technical module.*
>
> ☛ *To build a* ***Report DataSet Definition*** *you must have* ***Expert*** *MetaModel access.*

Although **Report DataSet Definition** creation is performed in **MEGA** (Windows Front-End) only, creation and use of **Report DataSet** is available in both **MEGA** Windows Front-End and Web Front-End for all users.

Once a **Report DataSet Definition** is created, any user can use it to query **MEGA** repository and create a **Report DataSet**. Data first shown in tabular form can then be handled and shown in graphical format using **Instant Report** feature.

The following points are covered here:

REPORT DATASET DEFINITION: THE BIG PICTURE

To create and define a **Report DataSet Definition**:

1. Define the **Report DataSet** source data: a MetaClass.
   This MetaClass is the root of the tree that represents the set of data collected from the repository.
   ☛ *See "Creating a Report DataSet Definition", page 7.*
2. (if needed) Define input parameters:
   Property parameters or **Collection parameters** are used to define both kind of filters on data extraction.
   The user is asked to enter these parameters at **Report DataSet** creation.
   - **Property parameters**
     ☛ *See "Adding Property parameters", page 9.*
   - **Collection parameters**
     ☛ *See "Adding Collection parameters", page 10.*
3. Define how to populate the **Report DataSet**:
   Define how the input parameters that feed the **Report DataSet** are retrieved.
   ☛ *See "Defining how to populate the Report DataSet", page 11.*
4. Define which data type feeds the data collection:
   Define the data you want to be displayed in the **Report DataSet**.
   ☛ *See "Defining the data that feeds the Report DataSet", page 13.*
5. Customize the **Report DataSet**.
   ☛ *See "Customizing the Report DataSet", page 17.*
6. Preview the **Report DataSet**.
   ☛ *See "Previewing the Report DataSet", page 15.*

REPORT DATASET DEFINITION CREATION

The **Report DataSet Definition** creation includes :

> ☛ *To customize the **Report DataSet Definition**, see "Previewing the Report DataSet", page 15.*

Creating a Report DataSet Definition

A **Report DataSet Definition** is MetaClass specific. The **Report DataSet Definition** creation consists in connecting a source MetaClass (concrete or abstract) to the **Report DataSet Definition**.

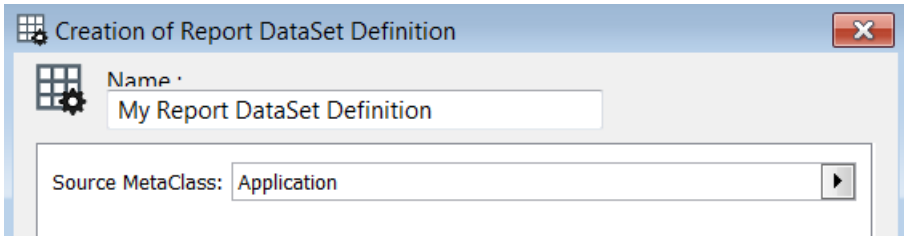
To create a Report DataSet Definition :

1. Connect to **MEGA** with MEGA Customizer business role.

   > ☛ *To build a* **Report DataSet Definition** *you must have **Expert** MetaModel access.*
2. In **MEGA** menu bar, select **View > Navigation Windows > Utilities**.
3. In the repository tree, right-click **Report DataSet Definition** folder and select **New > Report DataSet Definition**.
   The **Creation of Report DataSet Definition** windows appears
4. In the **Name** field, enter your **Report DataSet Definition** name.

   > ☛ *By default the Report DataSet Definition name is : Report DataSet Definition-x (x is a number).*
5. In the **Source MetaClass** field, click the arrow and select **Connect MetaClass**.
6. (optional) In the search field enter characters to filter the search.
7. Click **Find**  .
   The list of MetaClasses is displayed.
8. Select the MetaClass you want to connect to the **Report DataSet Definition**.

9. Click **Connect**.
   The selected MetaClass is defined as your **Report DataSet Definition** source MetaClass.



10. Click **OK**.
    You report **Report DataSet Definition** is created and added in the **Report DataSet Definition** folder.

☛ *See "Adding parameters to filter data extraction", page 8.*

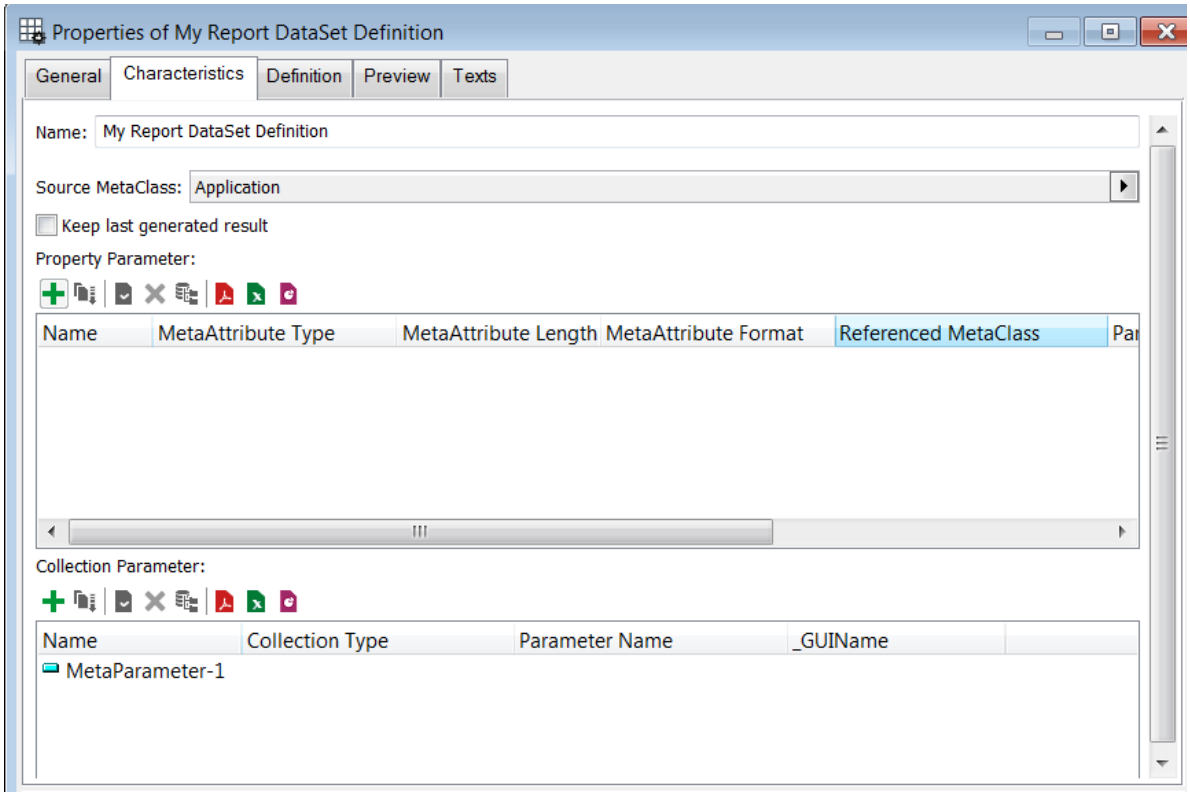
Adding parameters to filter data extraction

In the **Report DataSet Definition** you can define the parameters that are used to filter data at **Report DataSet** data collection.

If needed, you can add either **Property parameters** or **Collection parameters** or both, see:

- "Adding Property parameters", page 9
- "Adding Collection parameters", page 10

Adding parameters in the **Report DataSet Definition** **Characteristics** tab is useful for **Report DataSet Definition** re-usability.



Adding Property parameters

You can limit the collection of data set rows extracted from the source MetaClass collection. For this purpose, you can use **Property Parameters** in your **Report DataSet Definition**.

At **Report DataSet** creation the user is asked to enter the Property parameter values, which are taken into account (Property Parameters can be used in queries) to build the **Report DataSet**.

If no property parameters is specified the **Report DataSet** operates on the entire repository occurrences.

☛ *See also "Defining the data that feeds the Report DataSet: computed Property", page 15.*

To add Property parameter in the **Report DataSet Definition**:

1. From the **Utilities** Navigation Window, in the **Report DataSet Definition** folder, right-click the **Report DataSet Definition** and select properties.
2. Click the **Characteristics** tab.
3. In the **Property Parameter** pane, click **New**  .

☛ *You can add as many parameters as needed.*

4. In the **Property Parameter** pane:
   - enter your Property Parameter **Name** (value shown in the Report DataSet)

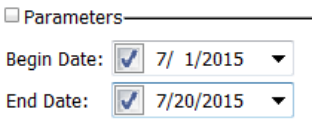
     `Example: Begin date`
   - (if needed) modify the default values for **MetaAttribute type**, **MetaAttribute Length** and **MetaAttribute Format**

     `Example: MetaAttribute Type: DateTime`
   - (if needed) in the **Referenced MetaClass**, connect a MetaClass
   - enter the Property Parameter **Parameter Name** (value used in the query you add in the **Definition** tab, see "Defining how to populate the Report DataSet", page 11)

     `Example: BeginDate`

Example:

`When you add "Begin Date" and "End Date" Property Parameters, at Report DataSet creation the user is asked to enter both dates. that filter the Report DataSet results.`



Adding Collection parameters

You can limit the collection of data set rows extracted from the source MetaClass collection. For this purpose, you can use Collection parameters in your **Report DataSet Definition**.

At **Report DataSet** creation the user is asked to enter the Collection parameter values, which are taken into account to build the **Report DataSet**.

By default once you selected the source MetaClass, a Collection parameter is added

in the **Collection Parameter** list. When defined, this Collection parameter is used to fill the root collection for data extraction.

☛ *See "Common use case of Report DataSet Definition", page 21.*

To feed the **Report DataSet Definition** input collection according to a collection:

1. From the **Utilities** Navigation Window, in the **Report DataSet Definition** folder, right-click the **Report DataSet Definition** and select properties.
2. Click the **Characteristics** tab.
3. In the **Collection Parameter** pane, click **New**  .

   ☛ *You can add as many parameters as needed.*
4. From the **Collection Parameter** pane, in the **Name** field, enter the name of the collection parameter.
5. In the **Collection Type** field, click the arrow and select **Connect MetaClass**.

6. Click **Find**  .
   The list of MetaClasses is displayed.
7. Select the MetaClass and click **Connect**.
8. In the **Parameter Name** field, enter a name for your Collection parameter.
   The Collection parameters are defined as your **Report DataSet Definition** input collections.

Example:

```
You can add the "Applications to extract" Collection
Parameter with the "Application" Collection Type, so that
at Report DataSet creation the user is asked to select the
applications to feed the Report DataSet with this
collection.
```

Defining how to populate the Report DataSet

In the **Report DataSet Definition** you have to define how to populate the **Report DataSet**.

Input data can be:

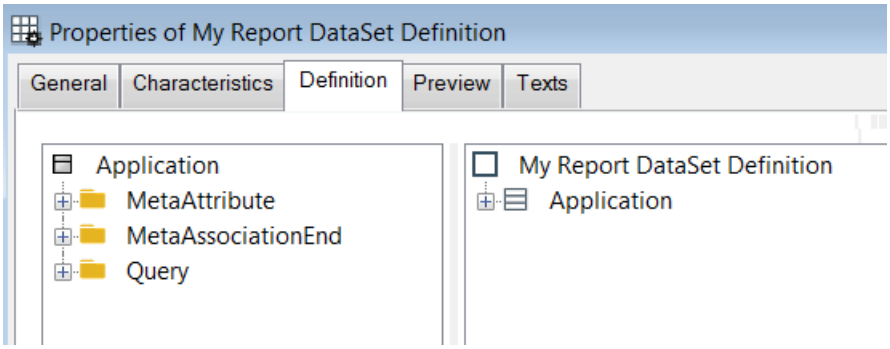
- attributes collected from the source MetaClass
- retrieved through queries related to the source MetaClass
- computed:
  - Report DataSet Property

☛ *See "Defining the data that feeds the Report DataSet: computed Property", page 15.*

To define how to populate the **Report DataSet**:

1. From the **Utilities** Navigation window, in the **Report DataSet Definition** folder, right-click the **Report DataSet Definition** and select properties.

2. Click the **Definition** tab.
   The right pane displays the root of the **Report DataSet Definition** tree structure: the Report DataSet Collection linked to the Report DataSet Structure .



3. In the right pane, right click the Report DataSet Collection and select **Properties**.

   `Example: Application`

4. To define how to populate the Report DataSet Collection:
   - in the **Populating Collection Parameter** field, select the Collection parameter.

     ☛ *A Collection parameter is available if in the **Report DataSet Definition Characteristics** tab, you added a **Collection Parameter**,* see *"Adding Collection parameters", page 10*.
   - else, in the **Populating Query** drop down list, select or create a query.

     ☛ *If you created a Property parameter in the **Report DataSet Definition Characteristics** tab (*see *"Adding Property parameters", page 9*)*, you can use it in the query.*

5. Click **OK**.

Defining the data that feeds the Report DataSet

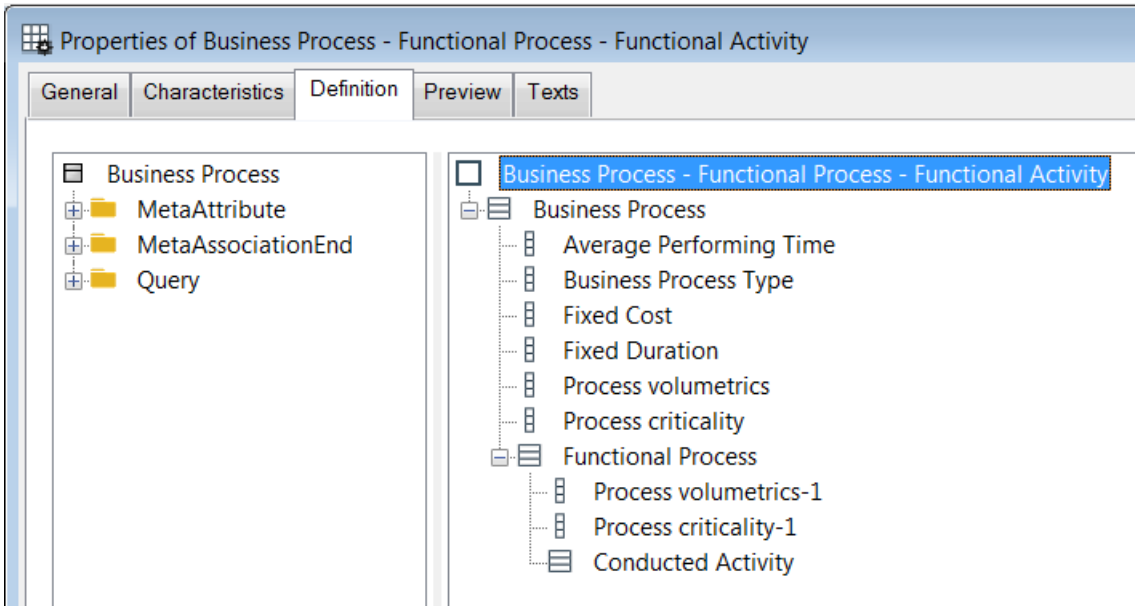
To define the data that feeds the **Report DataSet** you can:

- drag and drop MetaAttributes, MetaAssociationEnds, or queries directly belonging to the source MetaClass

  ☛ *See "Defining the data that feeds the Report DataSet: drag and drop", page 14.*

- create computed Properties

  ☛ *See "Defining the data that feeds the Report DataSet: computed Property", page 15.*



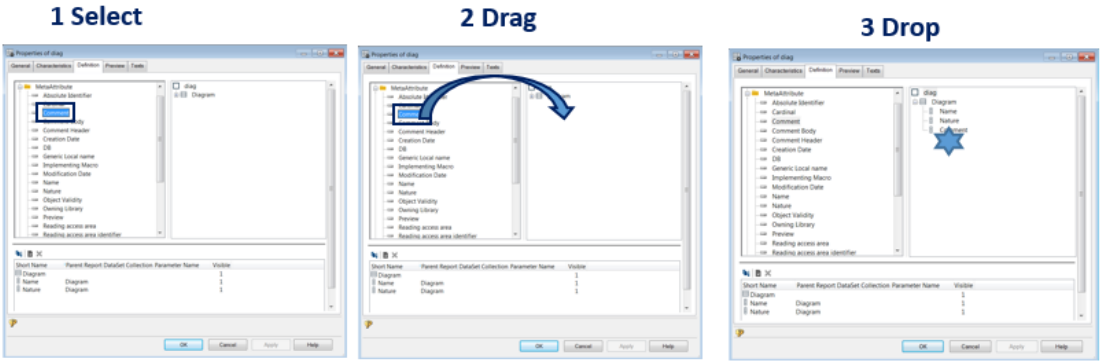
The Report DataSet Structure ☐ includes at least a root Report DataSet Collection ☰ with as many items (from drag and drop or computed) as needed:

- Report dataSet properties ☰
- Report dataSet Collections ☰ with:
  - Report dataSet properties ☰
  - Report dataSet Collections ☰

Defining the data that feeds the Report DataSet: drag and drop

You can feed the **Report DataSet** with:

- MetaAttributes
- MetaAssociationEnds
- queries



To define the data that feeds the **Report DataSet**:

1. From the **Utilities** Navigation Window, in the **Report DataSet Definition** folder, right-click the **Report DataSet Definition** and select properties.
2. Click the **Definition** tab.
   The left pane displays the source MetaClass with its associated MetaAttributes, MetaAssociationEnds, and queries.
3. In the left pane, expand the **MetaAttribute**/**MetaAssociationEnd**/**Query** folder.
4. Select :
   - a MetaAttribute of the MetaClass
   - a MetaAssociationEnd of the MetaClass
   - a query that retrieves data associated with the MetaClass
   - or any item under the MetaAttribute/MetaAssociationEnd of the MetaClass
5. Drag & drop the MetaAttribute/MetaAssociationEnd/MetaClass/query in the right pane.

   ☛ *You can drag and drop as many MetaAttributes/ MetaAssociationEnds/MetaClasses/queries as needed.*

   The MetaAttribute/MetaAssociationEnd/MetaClass or query is added:
   - in the right pane, in the **Report DataSet Definition** tree structure.
   - in the bottom pane: in the list of items. Each item represents a column header of the **Report DataSet**.

     ☛ *To sort the columns in the **Report DataSet** display, see "Modifying the display order of the Report DataSet columns", page 17.*

     ☛ *To hide a column in the **Report DataSet** display, see "Hiding a column of the Report DataSet", page 17.*

Defining the data that feeds the Report DataSet: computed Property

You can feed the **Report DataSet** with a computed property. The property is computed locally for the dataset. As computed, the property cannot be drag and drop. For this purpose you create a computed **Report DataSet** Property from the **Report DataSet Collection** using a macro.

To create a computed **Report DataSet** Property to define the data that feeds the **Report DataSet**:

1. From the **Utilities** Navigation Window, in the **Report DataSet Definition** folder, right-click the **Report DataSet Definition** and select properties.
2. Click the **Definition** tab.
3. In the right pane, right-click the **Report DataSet Collection** and select **New> Report DataSet Property**:
   - Select **Computed** property.
   - Define the MetaAttribute characteristics (**Type**, **Length**, **Format**)
   - In the **Implementation** field, click the arrow and select **Create a macro**.

☛ *For an example, see "Advanced use case of Report DataSet Definition: Applications of a Portfolio", page 21.*

Defining the data that feeds the Report DataSet: collection count Property

You can feed the **Report DataSet** with a collection count property. The property is collected locally for the dataset from a query or a MetaAssosicationEnd. As collected, the property cannot be drag and drop. For this purpose you create a collection count **Report DataSet** Property from the **Report DataSet Collection** using a query or a MetaAssosicationEnd.

To create a collection count **Report DataSet** Property to define the data that feeds the **Report DataSet**:

1. From the **Utilities** Navigation Window, in the **Report DataSet Definition** folder, right-click the **Report DataSet Definition** and select properties.
2. Click the **Definition** tab.
3. In the right pane, right-click the **Report DataSet Collection** and select **New> Report DataSet Property**:
   - Select **Collection count** property.
   - Define the MetaAttribute characteristics (**Type**, **Length**, **Format**)
   - In the **Implementation** field, click the arrow and select **Create a macro**.

Previewing the Report DataSet

From the **Report DataSet Definition** properties you can test the **Report DataSet**.

To preview the **Report DataSet**:

1. Access the **Report DataSet Definition** properties.
2. Click the **Preview** tab.

3. Click **Create Preview**.
4. In the **Parameters** pane, define the source data:
   - if needed enter the parameter value

     ☛ *See "Adding parameters to filter data extraction", page 8.*

   - if needed, in the source MetaClass, click Connect and connect all the MetaClass items you want.
5. In the **Report DataSet** pane, click **Refresh** .
   The **Report DataSet** displays:
   - date and time of computation
   - collected data in tabular form

     ☛ *Click a column header to sort the result according to this data.*

     ☛ *Objects from a Data Reading access not accessible to the user are not displayed.*

CUSTOMIZING THE REPORT DATASET

You can:

- modify the display order of the **Report DataSet** columns
- hide a column of the **Report DataSet**
- modify the name of a **Report DataSet** column header
- speed up the **Report DataSet** display

Modifying the display order of the Report DataSet columns

Once the **Report DataSet Definition** tree is defined you can modify the **Report DataSet** column display order.

To modify the display order of the **Report DataSet**:

1. From the **Utilities** Navigation Window, in the **Report DataSet Definition** folder, right-click the **Report DataSet Definition** and select properties.
2. Click the **Definition** tab.
3. In the bottom pane, click **Reorganize** .
4. Select the item you want to move and drag and drop where you want.

Hiding a column of the Report DataSet

To ease the **Report DataSet** report readability you can hide columns.

To hide a column:

1. From the **Utilities** Navigation Window, in the **Report DataSet Definition** folder, right-click the **Report DataSet Definition** and select properties.
2. Click the **Definition** tab.
3. In the bottom pane, clear the **Visible** field of the column you want to hide (0: the column is hidden).

Modifying the name of a Report DataSet column header

When performing a drag and drop (see "Defining the data that feeds the Report DataSet: drag and drop", page 14) **Report DataSet** item names are automatically defined according to what they stand for. When a name is already used in the same **Report DataSet Definition**, a suffix is added to the name (-n: -1, -2 etc.).

To modify the name of a **Report DataSet** column header you have to modify the **Report DataSet** item name.

To modify a **Report DataSet** item name:

1. From the **Utilities** Navigation Window, in the **Report DataSet Definition** folder, right-click the **Report DataSet Definition** and select properties.
2. Click the **Definition** tab.
3. In the bottom pane, click the **Short Name** field concerned and modify its name.

Speeding up the Report DataSet display

To speed up the **Report DataSet** result display:

1. From the **Utilities** Navigation Window, in the **Report DataSet Definition** folder, right-click the **Report DataSet Definition** and select properties.
2. Click the **Characteristics** tab, select **Keep last generated result**.
   The last generated **Report DataSet** is stored, so that at next session, the first **Report DataSet** display time is shorten.

How To

See:

How to add a column in a Report DataSet?

Each **Report DataSet** column corresponds to each data you defined as feeding the **Report DataSet**.

To add a column in a **Report DataSet** you can either:

- In the **Report DataSet Definition** definition tab, drag and drop object from the left pane to the **Report DataSet** tree structure.

  ☛ *See "Defining the data that feeds the Report DataSet", page 13.*

- In the **Report DataSet Definition** definition tab, from the **Report DataSet Collection** create a **Report DataSet Property**.

  ☛ *See "Defining the data that feeds the Report DataSet: computed Property", page 15.*

How to hide a column in a Report DataSet?

You can hide a column from:

- the **Report DataSet Definition**
  Data is not extracted from the repository.

  ☛ *See "Hiding a column of the Report DataSet", page 17.*

- the **Report DataSet**
  Data is extracted from the repository but not displayed in the **Report DataSet**.

  ☛ *See "Handling the Report DataSet", page 28.*

How to duplicate a Report DataSet

You might need to duplicate a **Report DataSet Definition** to modify it according to your needs.

To duplicate a **Report DataSet Definition:**

1. From the **Utilities** Navigation Window, in the **Report DataSet Definition** folder, right-click the **Report DataSet Definition** and select **Duplicate**.
2. Define the name of the duplicated **Report DataSet Definition**.

3. Click **OK**.

USE CASES

See:

Common use case of Report DataSet Definition

A common use case of **Report DataSet Definition** consists in defining the source collection as input parameter for the **Report DataSet**.

To create a common **Report DataSet Definition**:

1. Define the source MetaClass.

   ☛ *See "Creating a Report DataSet Definition", page 7.*

2. Define the filtering parameter: collection parameter.
   The collection parameter is pre-set or already set.

   ```
   E.g.: if the source MetaClass is Diagram, the "Diagram List"
   Collection Parameter is automatically added with the
   "Diagram" Collection Type value. This Collection parameter
   is used to list all the diagrams of the repository.
   ```

   ☛ *See "Adding Collection parameters", page 10.*

3. Define how to retrieve the repository data.
   In the **Populating Collection Parameter** field, select the collection parameter defined step *2*.

   ☛ *See "Defining how to populate the Report DataSet", page 11.*

4. Define the data you want to display in the **Report DataSet**.

   ☛ *See "Defining the data that feeds the Report DataSet", page 13.*

Advanced use case of Report DataSet Definition: Applications of a Portfolio

You can create a **Report DataSet Definition** that collects applications of a specific portfolio of a given vendor.

For this purpose you have to create:

- two **Property parameters**:
  - "Vendor Name" to define from who the end user wants to collect the applications
  - "Portfolio" to define from which portfolio the end user wants to collect the applications
- a **Query** ("APM - Get Applications of a Portfolio using Technologies of a given Vendor") that computes the input data:

```
Select [Org-Unit] Into @OrgUnits Where [Name] = &"Vendor"
Select [Software Technology] Into @Technologies Where
[Vendor] in @OrgUnits
Select [Application] Where ([Used Technology] in
@Technologies)
          And [Initiative].[Owner Portfolio] = &"Portfolio"
Or  [Deployed Object]:[Software
Installation].[Initiative].[Owner Portfolio] = &"Portfolio"
```

- a **Report DataSet Property** ("Number of processes") computed with a **Macro** ("Application Process.Implementation"):

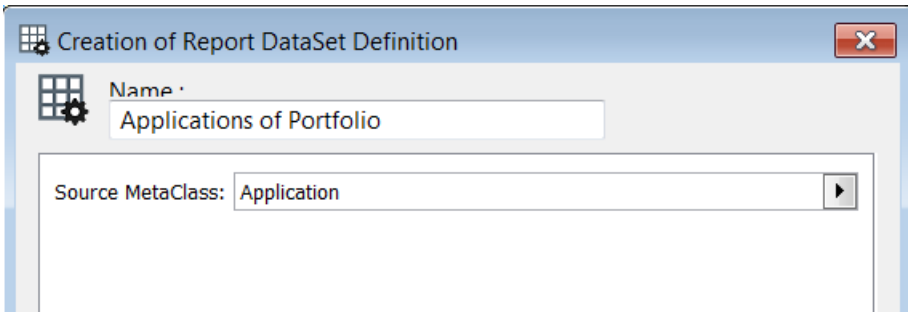
```
Sub GetAttributeValue(Object as MegaObject,AttributeID as
Variant,Value as String)
  Dim mgApp
  Set mgApp =
Object.GetRoot.GetCollection("Application").Item("~" &
Object.GetProp("~QPo(Ab(VL1Z0[Application]"))
  If mgApp.Exists Then
    Value = mgApp.GetCollection("~h4n)MzlZpK00[Business
Process]").count
  Else
    Value = 0
  End If
  ' Value = 4
End Sub
```

To create the Report DataSet Definition:

1. Create a Report DataSet Definition based on Application source MetaClass.

   e.g.: Report DataSet Definition **Name**: `Applications of Portfolio`.

   ☛ *See "Creating a Report DataSet Definition", page 7.*

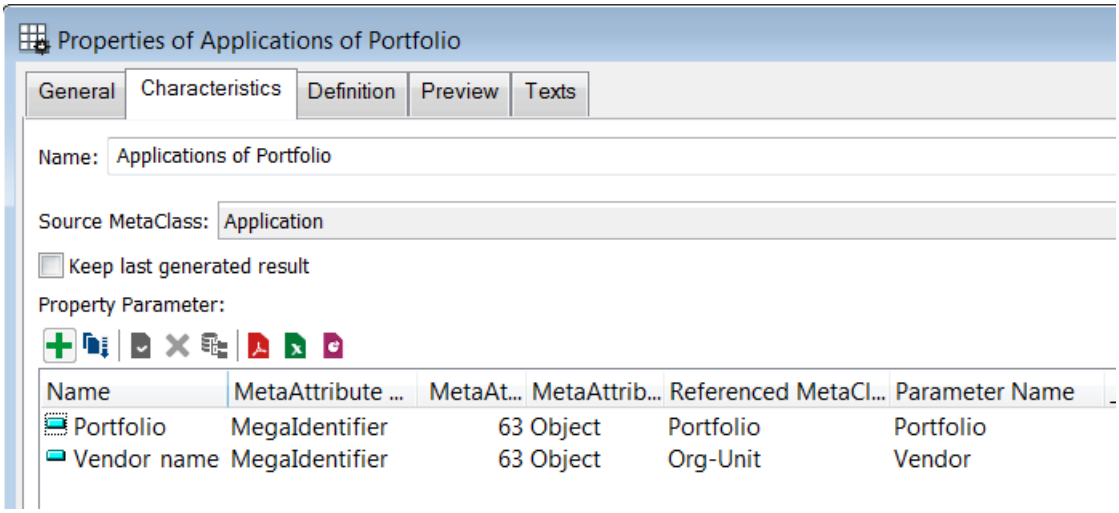


2. Create the "Portfolio" **Property parameter**:
   - In **MetaAttribute Type** field: select MegaIdentifier
   - In **MetaAttribute Format** field: select Object
   - In **Referenced MetaClass** field: connect Portfolio MetaClass
   - In **Parameter Name** field: enter "Portfolio"

   ☛ *See "Adding Property parameters", page 9.*

3. Create the "Vendor name" **Property parameter**:
   - In **MetaAttribute Type** field: select MegaIdentifier
   - In **MetaAttribute Format** field: select Object
   - In **Referenced MetaClass** field: connect Org-Unit MetaClass
   - In **Parameter Name** field: enter "Vendor"

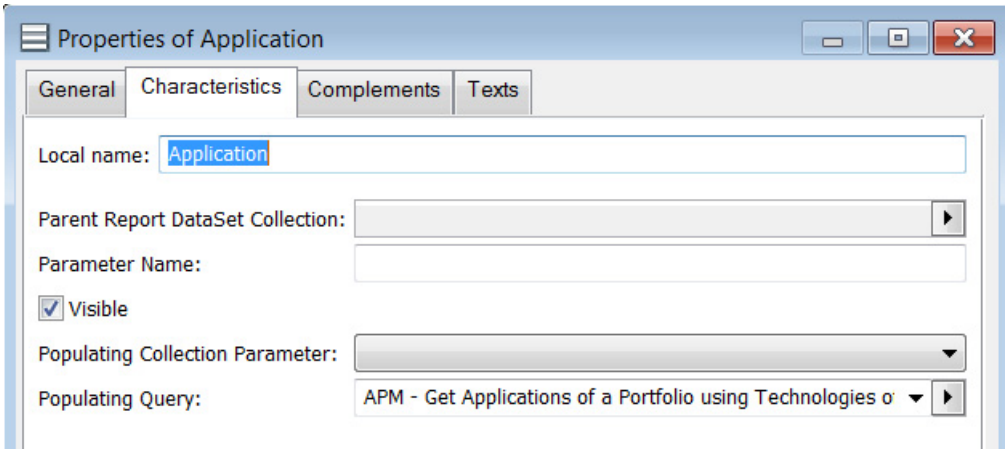
   ☛ *See "Adding Property parameters", page 9.*



4. Define how to collect the input data, from the **Report DataSet Collection** properties.
   In the **Populating Query** field, create the query "APM - Get Applications of a Portfolio using Technologies of a given Vendor".
   From the Query **Characteristics** tab:
   - in **Stereotype** field select "Internal Query".
   - in **Query Implementation** field select "Select"
   - in the **Collection Type** field connect "Application" MetaClass

   In the **Query Code** tab, enter the query code

   ☛ *See "Defining how to populate the Report DataSet", page 11.*

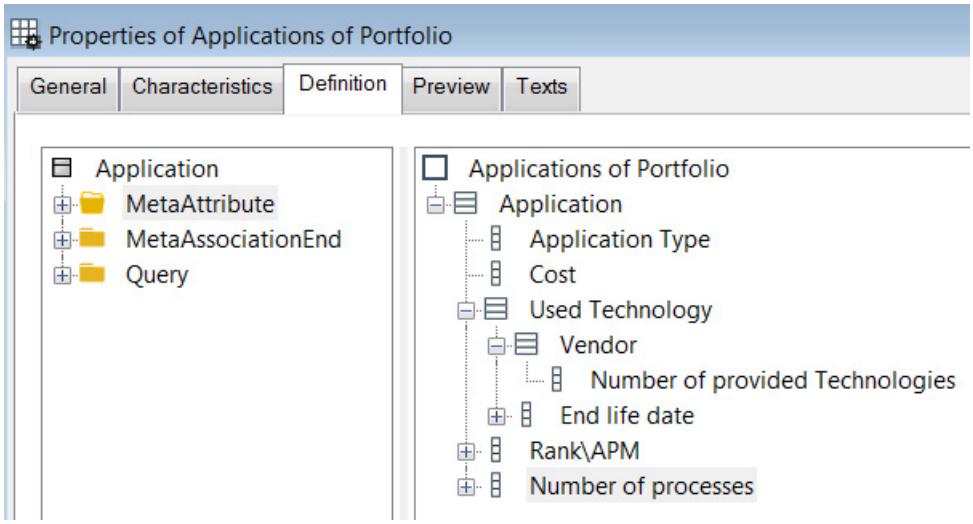


5. Define the data you want to display in the **Report DataSet**, from the **Definition** tab:
   In the left pane, select the object and drag and drop it in the right pane:
   - **Application Type** MetaAttribute
   - **Cost** MetaAttribute
   - **Software Technology (used technology)** MetaAssociationEnd
   - **Org-Unit (Vendor)** MetaAssociationEnd (belonging to **Software Technology (used technology)** MetaAssociationEnd)
   - **Number of provided Technologies** MetaAttribute (belonging to **Org-Unit (Vendor)** MetaAssociationEnd)
   - **End life date** MetaAttribute (belonging to **Software Technology (used technology)** MetaAssociationEnd)
   - **Rank/APM** MetaAttribute

   ☛ *See "Defining the data that feeds the Report DataSet: drag and drop", page 14.*

6. Define the data you want to display in the **Report DataSet**, from the **Definition** tab:
   Create a computed parameter "Number of processes":
   Select **Computed Property**.
   - in the **MetaAttribute Type** field, select "Long"
   - in the **MetaAttribute Length** field, enter 63
   - in the **Implementation** field, select **Create macro** and create the "Application Process.Implementation" macro.

   ☛ *See "Defining the data that feeds the Report DataSet: computed Property", page 15.*



7. Reorder the **Report DataSet** columns and hide the Application Type column.

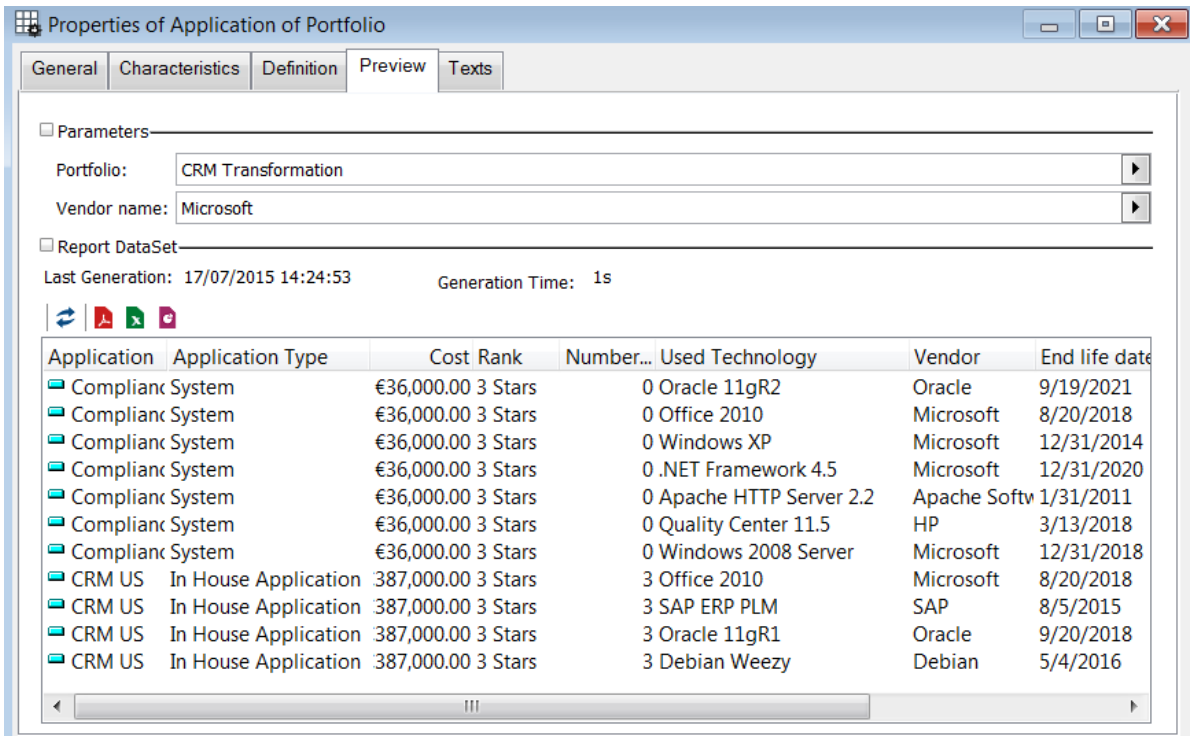
   ☛ *See "Modifying the display order of the Report DataSet columns", page 17.*

   ☛ *See "Hiding a column of the Report DataSet", page 17.*

Short Name	Parent Report DataSet Collection	Parameter Name	Visible
Application			1
Application Type	Application		0
Cost	Application		1
Rank\APM	Application		1
Number of Processes	Application		1
Used Technology	Application		1
Vendor	Used Technology		1
End life date	Used Technology		1
Number of provided Technologies	Vendor		1

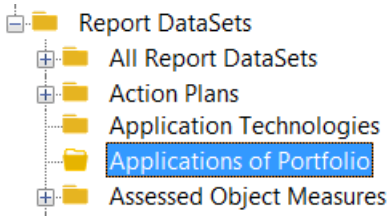
8. In the **Preview** tab test your **Report DataSet Definition**.

☛ *See "Previewing the Report DataSet", page 15.*



REPORT DATASET CREATION

Once the **Report DataSet Definition** is created a new **Report DataSet Definition** folder is added in the **Documentation** Navigation window > **Report DataSets** folder.



The **Report DataSet Definition** is available for any user to create a **Report DataSet**.

See :

- "Creating a Report DataSet", page 27
- "Handling the Report DataSet", page 28

Creating a Report DataSet

Creating a Report DataSet (from Enterprise Architecture Desktops)

To create your **Report DataSet**:

1. Access your Enterprise Architecture Desktop (Windows Front-End or Web Front-End).
2. From the **Documentation** Navigation Window > **Report DataSets** folder, right-click the **Report DataSet Definition** and select **New > Report DataSet**.
3. Enter the **Name** of your Report DataSet**.**
4. Click **OK**.
   You **Report DataSet** is created.
5. From the **Report DataSet** Properties, click the **Data** tab.
6. In the **Parameters** pane, enter the requested parameters.
7. In the **Report DataSet** pane, click **Refresh** .
   The **Report DataSet** is fed with data.

Creating a Report DataSet (from MEGA Solutions)

To create your **Report DataSet** (Web Front-End):

1. From your **MEGA Solution** Desktop, select **Reports > Other reports > My Report DataSets**.
2. From the Edit area, in **My Report Datasets** tab click **New**.

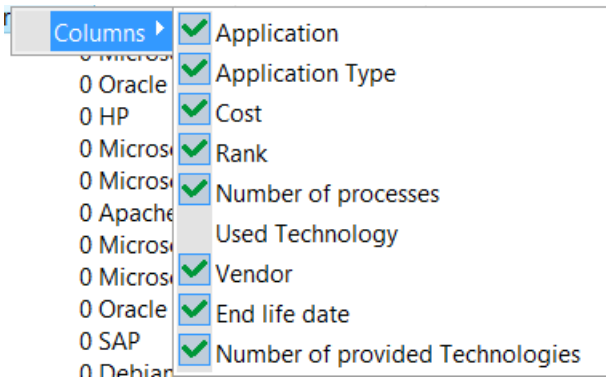
3. Enter the **Name** of your Report DataSet.
4. In the **Report DataSet Definition** drop down list, select a **Report DataSet Definition**.
5. In the **Parameters** pane, enter the requested parameters.
6. Click **OK**.
   You **Report DataSet** is created.
7. In the **Report DataSet** pane, click **Refresh**.
   The **Report DataSet** is fed with data.

Handling the Report DataSet

Once the **Report DataSet** is created, you can:

- modify the **Report DataSet** status.
  By default the **Report DataSet** is public. It is stored in **MEGA** and shared with other users.

  ☛ *To keep the **Report DataSet** private, from the **Report DataSet** properties > **Characteristics** tab, in the **Report DataSet Sharing** drop-down list, select "Private".*
- create an Instant Report from your **Report DataSet**: click **Instant Report**.
- refresh the **Report DataSet** with updated data: click **Refresh**.
- export the **Report DataSet**: click **PDF** or **Excel**.
- hide **Report DataSet** columns: right-click the table header, select **Columns** and clear the items you want to hide.



☛ *For detailed information on **Report DataSet** handling see Common Features User Guide.*

HOPEX Studio - Report Studio

Report Template Creation and Customization

1 Introduction

To use **Report Studio** features, you need **HOPEX Studio** license.

Dynamic reports enable to analyze repository data through different focus.

Each report is built from a report template, which defines parameters on which is based the report.

> ☞ For more details on the use of reports, see *MEGA Common Features* user guide, “Documentation” part, "Generating Documentation with MEGA" chapter.

MEGA supplies predefined report templates. Each report template is specific to a MEGA Suite product (e.g.: Risk analysis).

Report Studio tool enables you to create your own report templates.

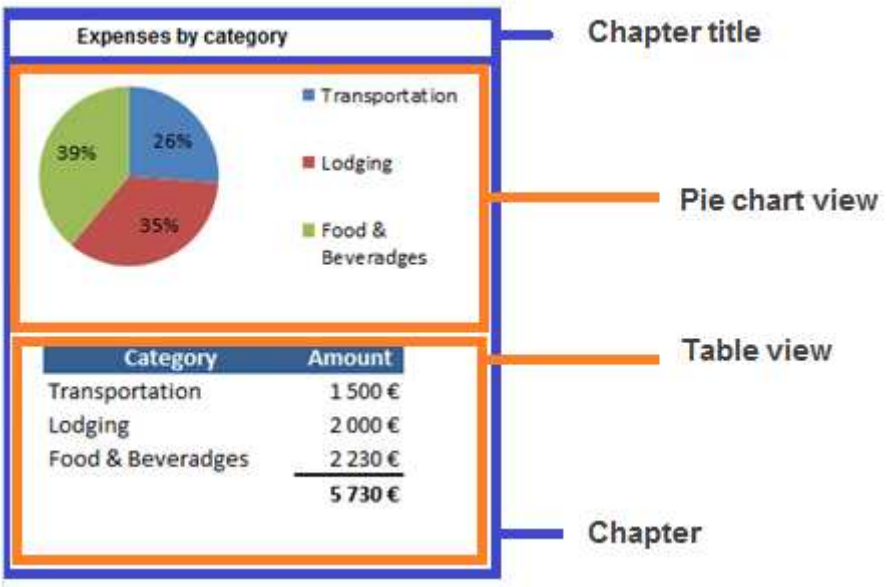
This guide details how to create and customize a report template.

Note: before starting with **Report Studio**, check that you are in “Expert” Metamodel Access (**Repository** Options).

2 Report Structure

A report can include one or several chapters.

Each chapter includes a set of views (or a macro), each of them showing values (data structure) displayed graphically or not according to a renderer type.



To create a report template, you must define:

- (Optional) the parameters, which define the objects on which the report rely on.
- the report chapter(s).

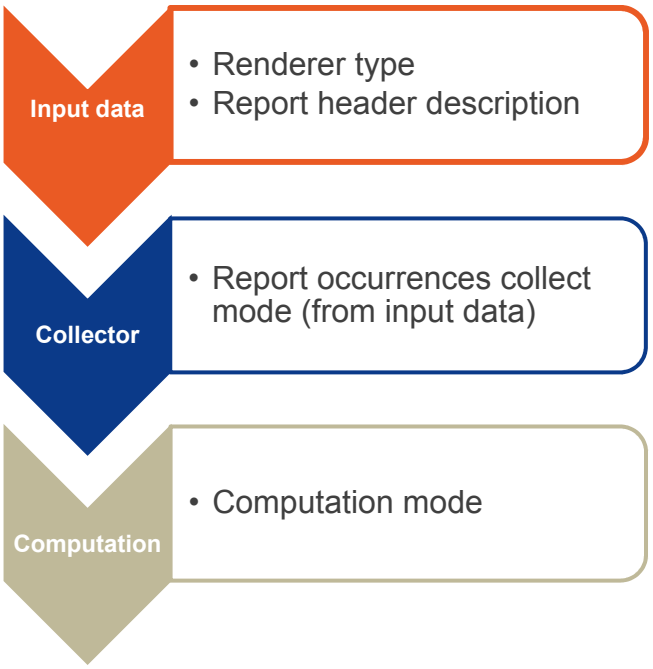
  A chapter processes report objects (in the above example the chapter processes expenses).

- either the macro or the view(s) that make up each chapter.

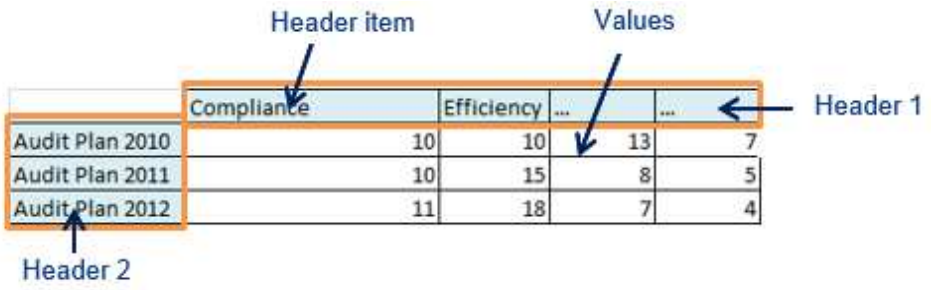
If the chapter is based on:

- a macro, this one must refer to java code that implement dedicated interfaces.
- a set of views, for each view you must define:
  - the render type (graphical)

    The render type selected defines the data structure with the header number (e.g.: two headers for a matrix, three headers for a bubble-diagram).
  - input data, which makes up data structure headers and from which the report values/occurrences are collected
  - value collect mode
  - value computation mode



Data structure example:



3 Report Studio vs Java Report

3.1 Report Studio

Report Studio enables to quickly generate simple reports but does not provide as many possibilities as the reports written in Java do. You first need to identify which of the two techniques best suits your needs. This section details renderer scope and limitations.

3.1.1 Renderers

Within the same chapter you can display different renderers on several lines, and on each of these lines in several columns.

`Example: “Execution and performance Heatmaps” sample delivered in MEGA.`

With the same input data you can display several renderer types.

Parameters

Each of the renderers provides parameters that have an effect on the display (e.g.: legend location, 3D, color).

When designing the report template, default values are set for these parameters, but these parameters can be modified by the report designer and also by the report user.

Limitations

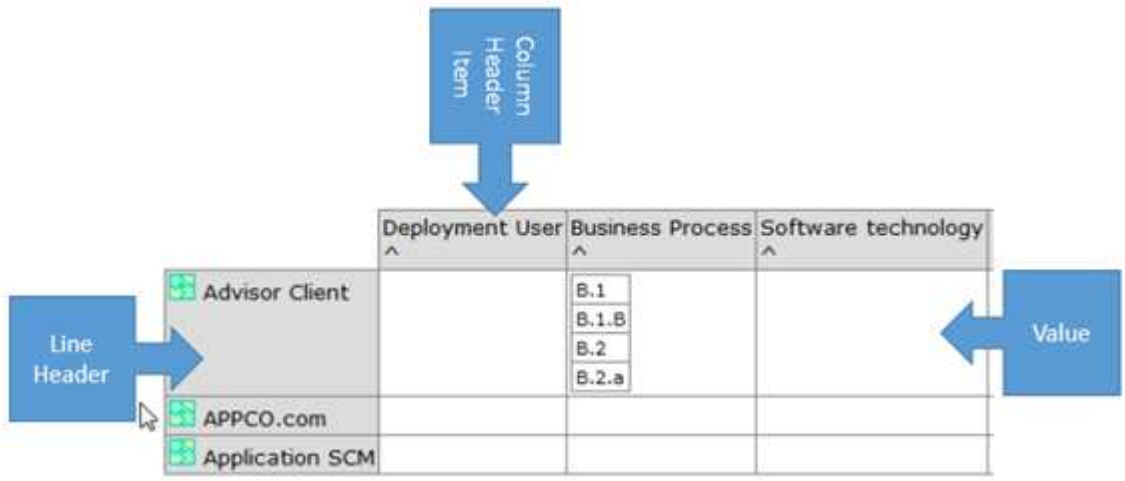
The renderer size is automatically calculated. You cannot change it, or define its margins.

The set of delivered colors are limited but you can define new ones.

The query used as provider cannot use two parameters of the same type.

3.1.2 Table / Matrix / Heatmap

Table/matrix/heatmap type reports are built using “Table” or “ VerticalTextTable” renderer.



Content

Header items consist of a list of occurrences, which are retrieved through a query, or defined one by one by a specific occurrence (MetaAttribute, CodeTemplate).

Displayed values can be:

- an occurrence retrieved by browsing a link or a query from a report input parameter or from an occurrence in another table column

  The displayed attribute can be customized.
- text
- numerical values (e.g.: calculation result from the column occurrences)

  Common calculation functions are available on this collection in a column processing script.

Formatting

Each cell background color can be based on a column processing script. If Boolean is returned it is shown as a green checkmark or a red cross.

With the View parameter you can define text alignment, background color.

Cell size can be set for the whole table (Width of column) and table size can be set (Width of table)

Customization

See Customizing a report template p. 63.

Limitation

You cannot modify:

- the font
- the header formatting
- the sorting capability
- the editing capability

Editing

You can sort the table content according to any columns. When a header corresponds to an occurrence, its associated image is displayed and enables to access the object menu.

When the cell shows an occurrence attribute, this attribute can be edited in-place.

When a column shows an occurrence list, a drill-down is available.

Exports

When a cell value includes a table, this table is not exported and the cell is empty in Excel.

3.1.3 LineCharts, BarCharts, AreaCharts, RadarCharts

Content

Header items consist of a list of occurrences, which are retrieved through a query, or defined one by one by a specific occurrence (MetaAttribute, CodeTemplate).

One header defines the data series shown also in the legend. The other header defines the items of the x-axis of the chart.

The displayed values can be:

- numerical values (e.g.: calculation result from the column occurrences)

  Common calculation functions are available on this collection in a column processing script.

Formatting

Two set of colors are provided. It is possible for Advanced Report Template creator to define other color sets.

Limitation

This is not the built-in way to define a line chart that represents an evolution in a defined period (year, month…).

The scale is automatic.

Editing

On each part of the chart a drill-down is available showing the collection input of the computation method associated with this part. This collection can be overloaded if needed.

3.1.4 Pie Charts

Content

This renderer needs a single header that is used to fill the legend of the pie chart.

Header items consist of a list of occurrences, which are retrieved through a query, or defined one by one by a specific occurrence (MetaAttribute, CodeTemplate).

The values used by the renderer are:

- numerical values (e.g.: calculation result from the column occurrences)

  Common calculation functions are available on this collection in a column processing script.

Each part of the pie can be computed separately if needed by defining a different computation for each header item.

Formatting

Two set of colors are provided. It is possible for Advanced Report Template creator to define other color sets.

Editing

On each part of the pie a drill-down is available showing the collection input of the computation method associated with this part. This collection can be overloaded if needed.

3.1.5 List

Content

This renderer needs a single header.

Header items consist of a list of occurrences, which are retrieved through a query, or defined one by one by a specific occurrence (MetaAttribute, CodeTemplate).

The values used by the renderer can be:

- an occurrence retrieved by browsing a link or a query from a report input parameter or from an occurrence in another table column

  The displayed attribute can be customized.
- text
- numerical values (e.g.: calculation result from the column occurrences)

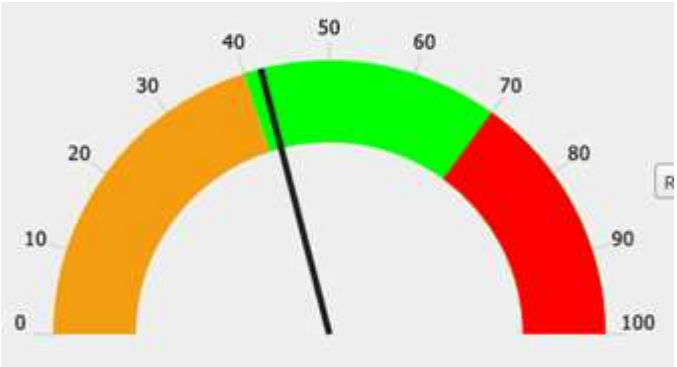
  Common calculation functions are available on this collection in a column processing script.

Limitation

You cannot modify:

- the text alignment
- the font

3.1.6 Gauge



Limitation

Gauge parameter is defined statistically (not dynamically).

3.2 Java Reports

Java Reports provide more extensive possibilities but require development skills, as it is required to implement a macro in Java.

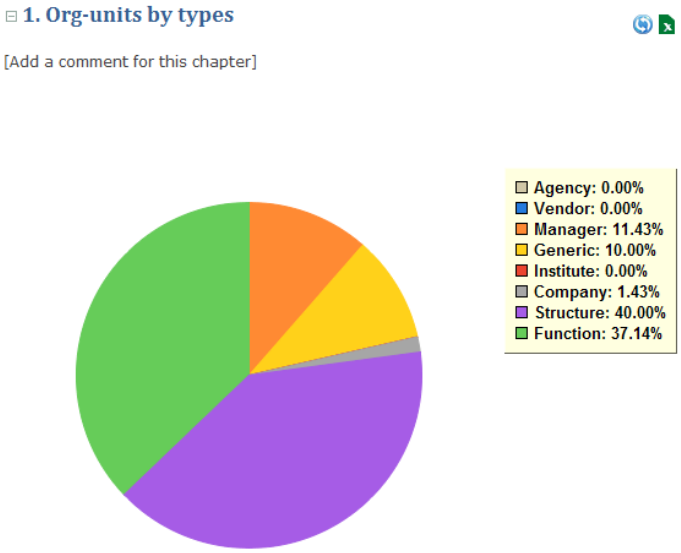
☞ See *Writing Java Report Chapters* Technical Article.

4 Report Templates creation examples

4.1 “Sorting org-units by type” Report Template

Reports generated from the “Sorting org-units by type” Report Template show in a pie chart the percentage of org-units sorted according to their type.

To do so, you have to define a report input parameter “Org-Unit” of type Org-Unit, from which the header item values will be collected. The report results will be computed from this header using a query.



Input data:

- renderer type: Pie Chart
- a single header: Org-unit type

Collector:

- input parameter: Org-Unit type
- query: **Select** [Org-Unit] **Where** [Name] **= &"Name" And** [Org-Unit Type] **= &"Type"**

Computation:

- method:

```
Function Compute(input,config)
  Compute = input.count
End Function
```

4.1.1 Creating a Report Template: “Sorting org-units by type”

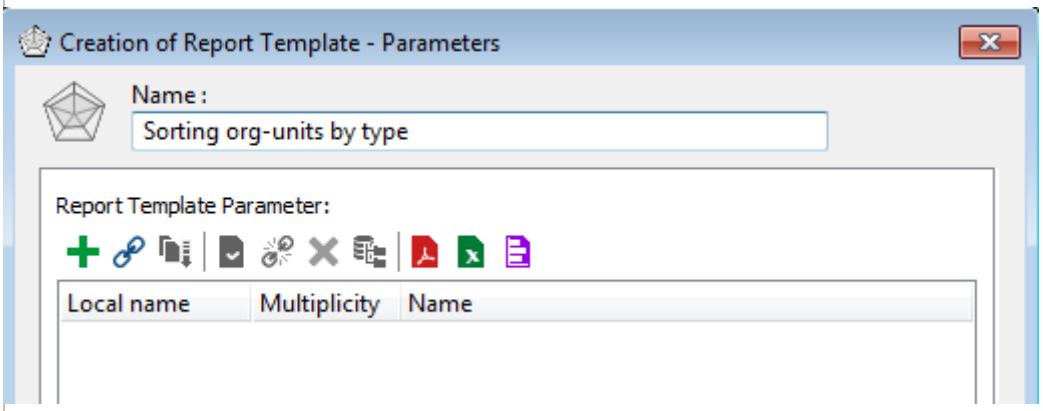
To create the “Sorting org-units by type” report template:

Report Template creation

1. From the **Utilities** Navigation window, right-click the **Report Templates** folder and select **New** > **Report Template**.

   The **Creation of Report Template** wizard appears.
2. Enter the Report Template **Name** and eventually a comment.
3. Click **Next**.

   The **Parameters** definition wizard page appears.



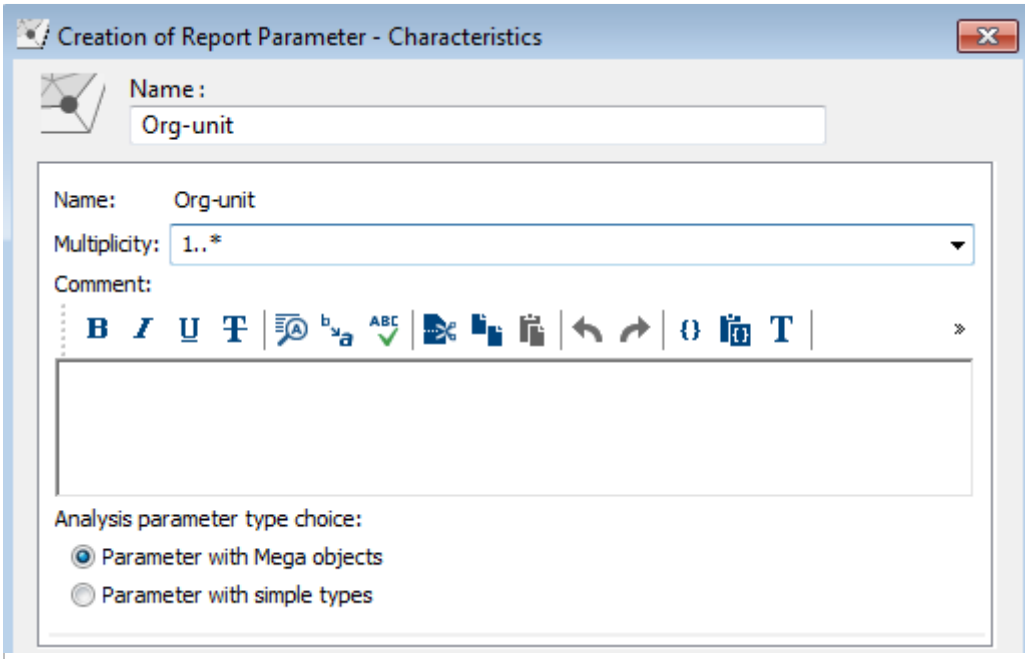
Parameter creation and definition

To define the parameter name, multiplicity (the user can select 1 to n occurrences or org-units as report input), type, and to which object type the parameter refers (Org-Unit).

4. In the **Report Template Parameter** pane, click **New**  .

   ☞ For more details on parameters, see Defining new parameters in a report template p. 67.

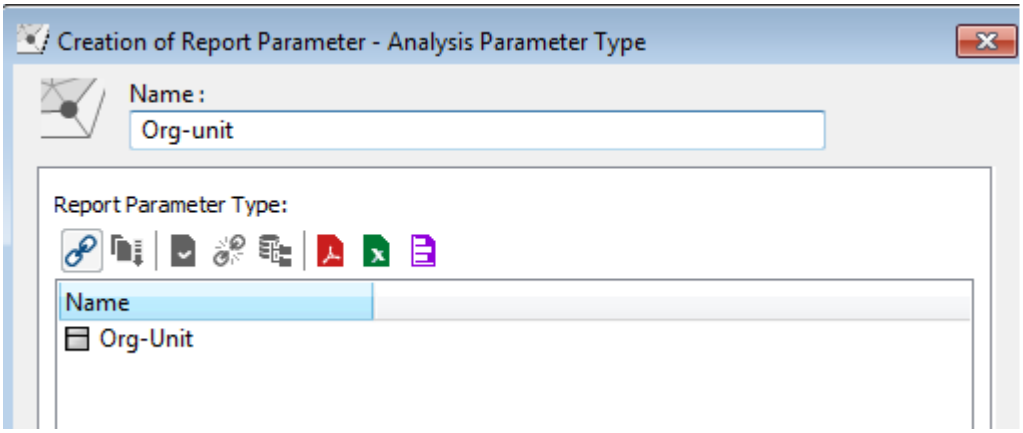
   The **Creation of Report Parameter** wizard appears.
5. Enter the parameter **Name**: “Org-unit”.
6. Click **Next**.
7. In the **Multiplicity** field, select “1..*”.
8. In the **Analysis parameter type choice**, keep the “Parameter with MEGA objects” option selected.



9. Click **Next**.

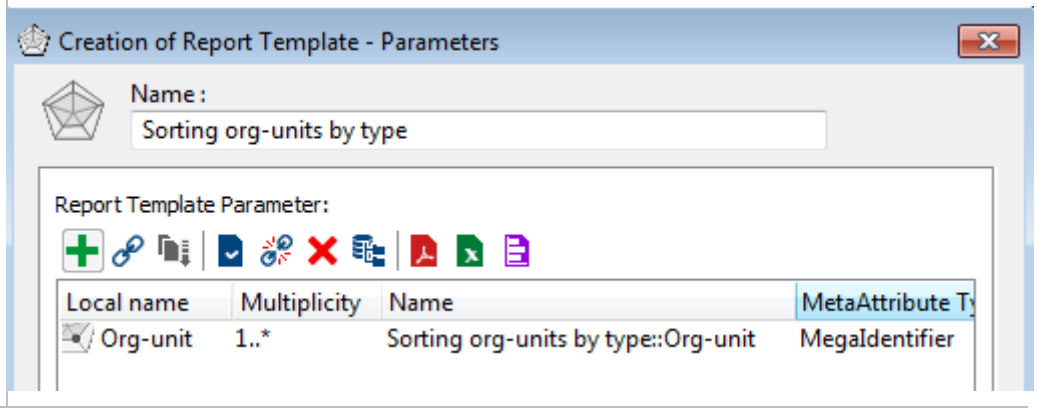
   The **Analysis Parameter Type** wizard page appears.

10. In the **Report Parameter Type** pane, click **Connect**  .
11. In the **Connecting** window, select the **Org-Unit** object type and click **Connect**.



12. Click **Finish**.

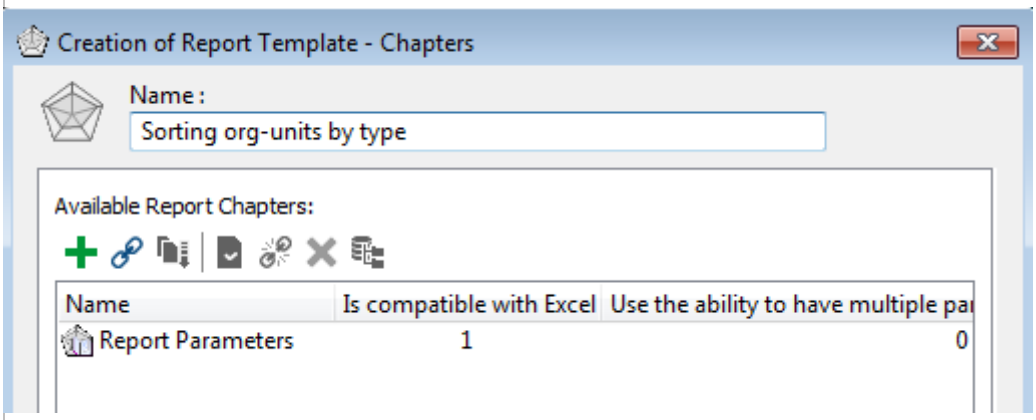
    The “Org-unit” parameter is created and defined.



13. Click **Next**.

    The **Chapters** wizard page appears.

    A default chapter is already created, it displays the **Report Parameters**.



Chapter creation

To create the report chapter that will display the org-units by type results.

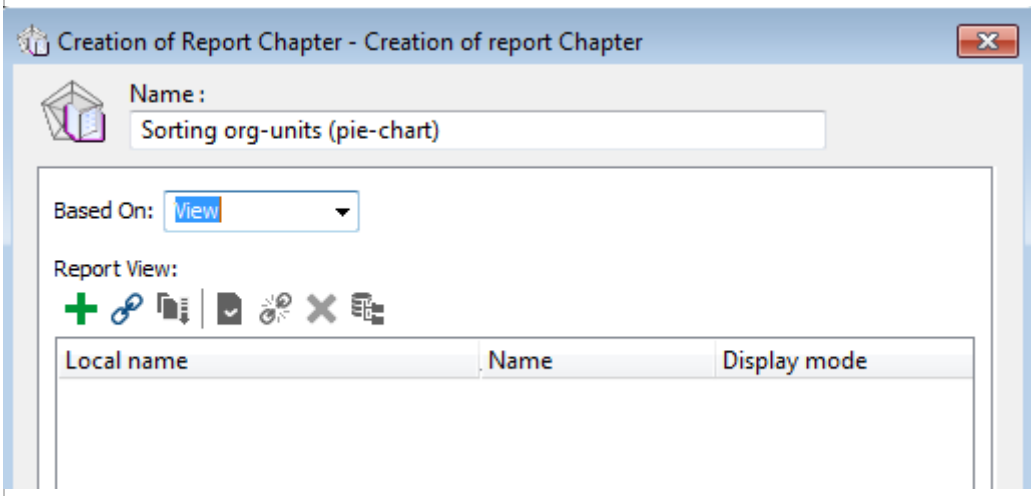
The report is based on a view.

14. In the **Available Report Chapters** pane, click **New** .

    The **Creation of report Chapter** wizard appears.

15. Enter the chapter **Name**.
16. Indicates that it is **Based On** a **View**.

    The **Report View** pane appears.



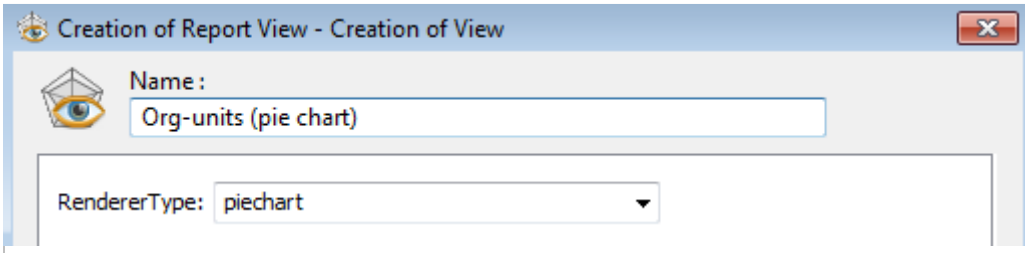
17. In the **Report View** pane, click **New** .

    The **Creation of View** wizard appears.

View creation

To select the renderer type used for data display (Pie chart).

18. (optional) Enter the view **Name**.
19. In the **RendererType** field, select “piechart”.



20. Click **Next**.

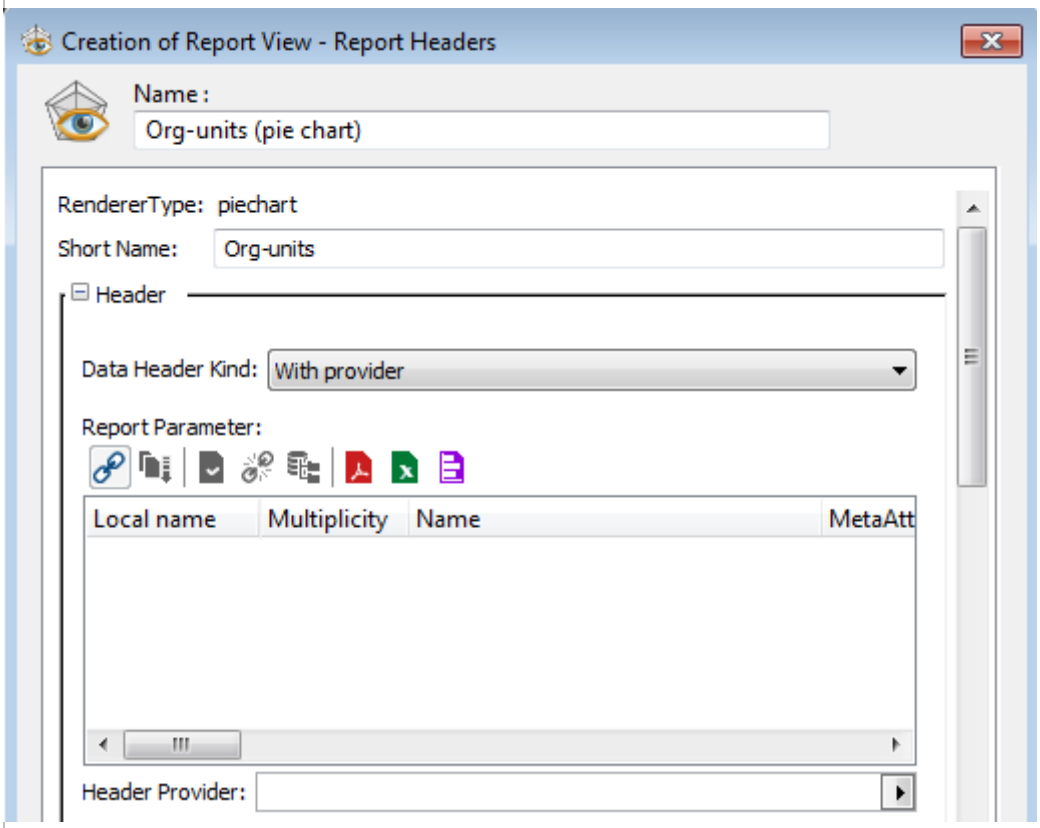
    The **Report Headers** wizard page appears.

Report Headers definition

The data structure is created with the header number intended by the renderer type display.

Here (Pie chart) a single header: “Org-units by type”.

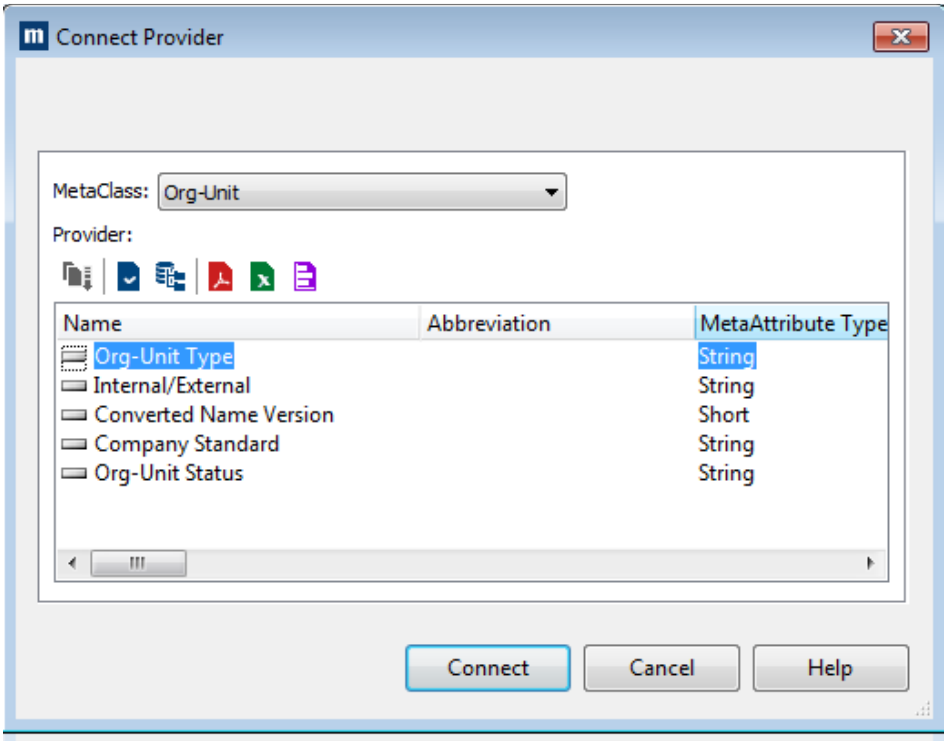
21. In the **Short Name** field, enter the Header name.



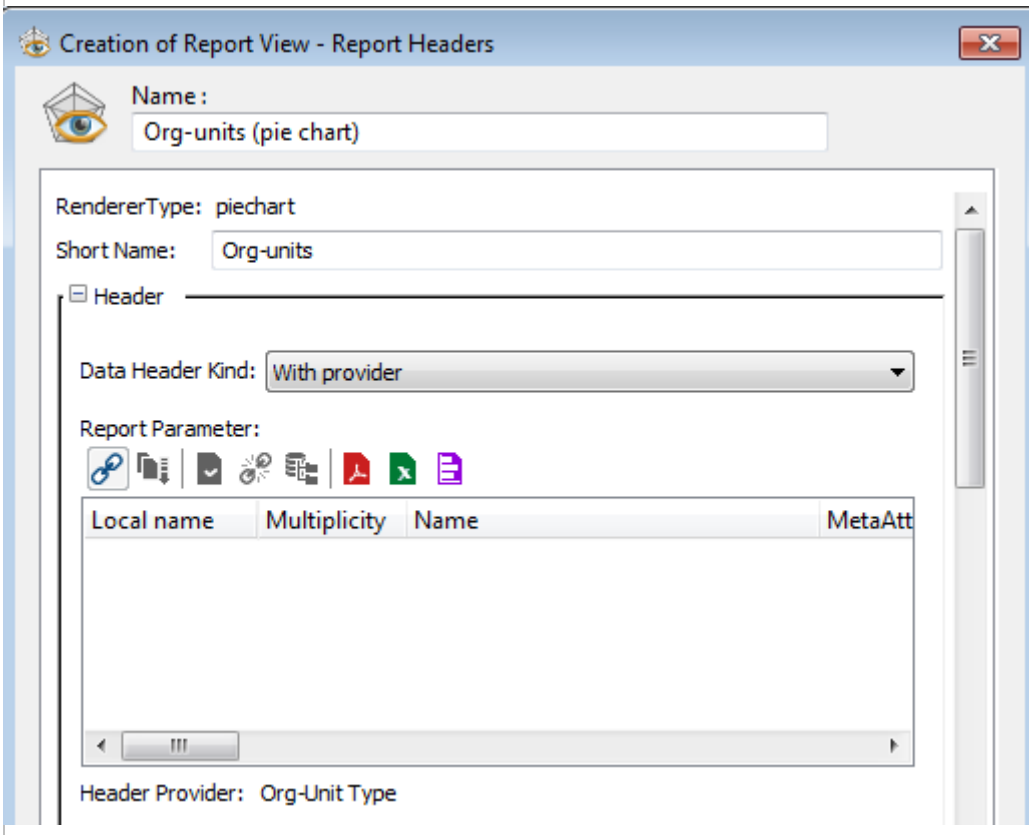
22. Define how the header items are retrieved:

    In this example, a query enables you to retrieve the header items from the parameter defined as report template input.

    - in the **Data Header Kind** field, select “with provider”.
    - leave the **Report Parameter** pane empty.
    - In the **Header Provider** field, select “Connect MetaAttribute” (Select MetaClass: Org-Unit, Provider: Org-Unit Type).



23. Click **Connect**.

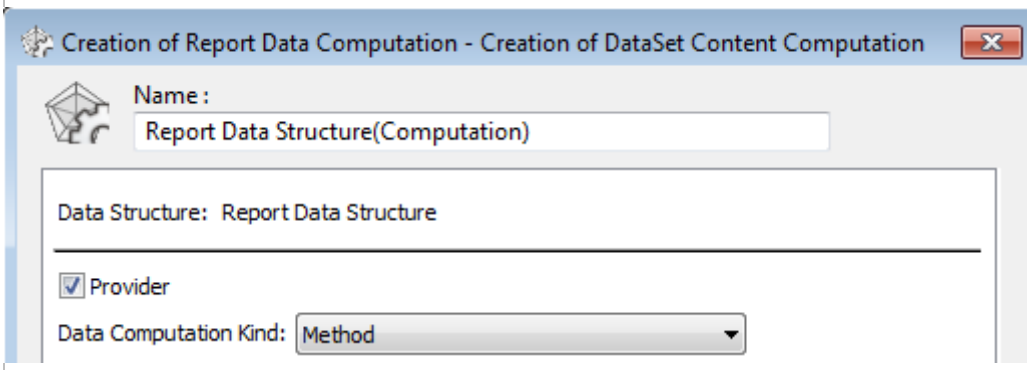


24. Click **Finish**.

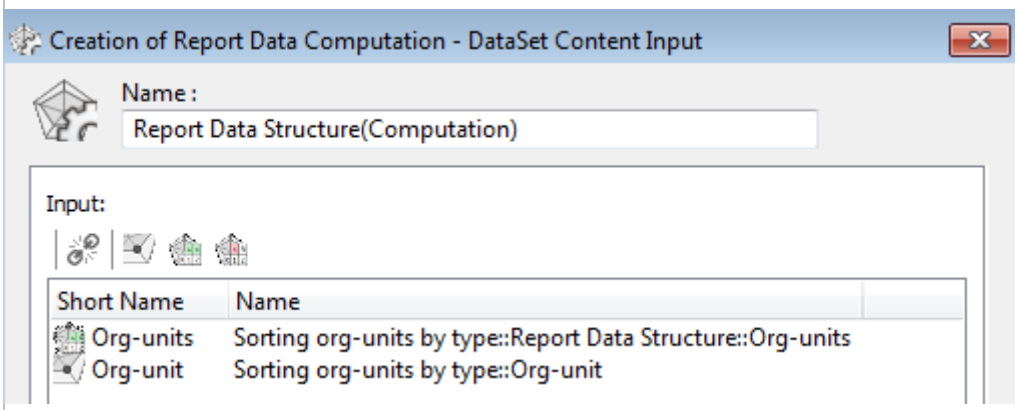
    The **Creation of Report Data Computation** wizard appears.

Report data computation creation

25. From the **Creation of DataSet Content Computation** wizard page, in the **Report Data Structure** pane, select **Provider**.
26. In the **Data Computation Kind**, select **Method** computation mode.



27. Click **Next**.
28. In the **Input** pane:
    - click **Connect Report Parameter** and select the corresponding objects (Org-unit).
    - click **Connect Header** and select the corresponding objects (Org-units).



29. Click **Next**.

    The **Dataset Content Computation Provider** wizard page appears.

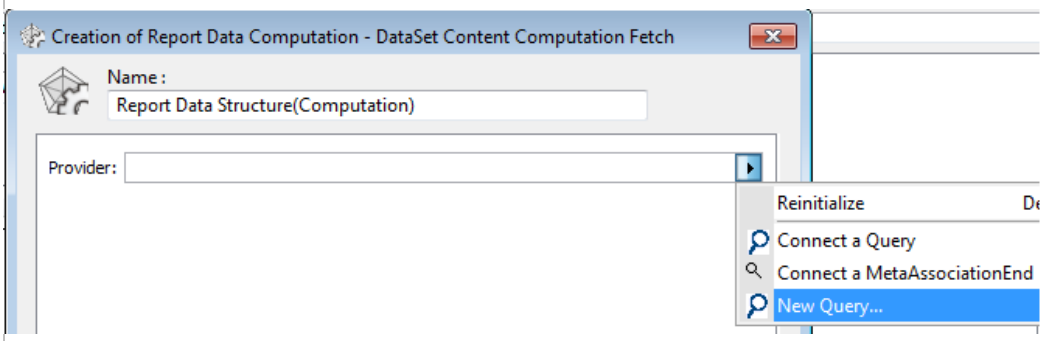
DataSet content computation creation

This wizard defines the input data processing and/or computation, which enable to retrieve the values stored in the data structure and provide them to the renderer.

When a parameter is associated with the provider, the wizard enables to select a query.

DataSet content input

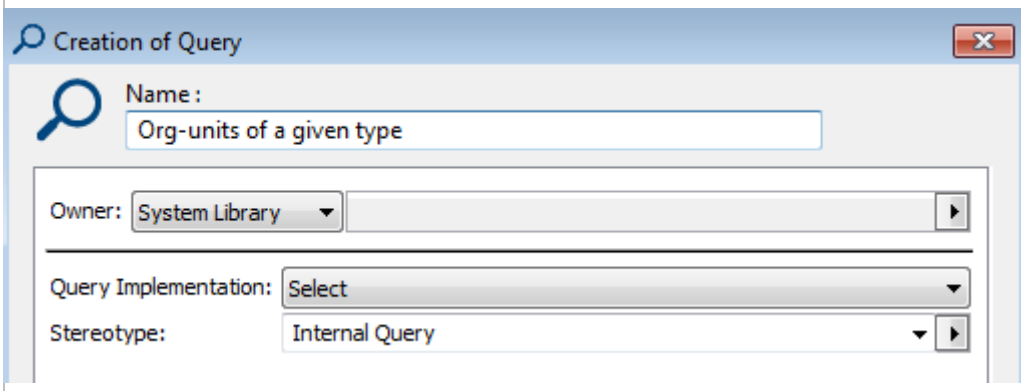
30. In the **Provider** field, select **New Query**.



    The **Creation of Query** window appears.

31. In the **Name** field, enter the query name.

32. In the **Query Implementation** field, keep **Select**.
33. In the **Stereotype** field, select **Internal Query**.

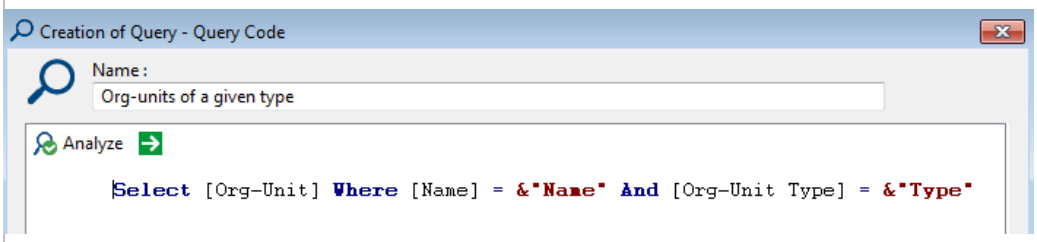


34. Click **Next**.

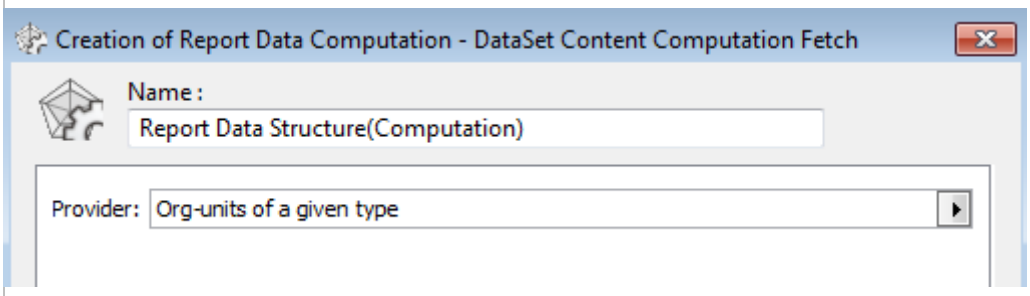
    The **Query Code** wizard page appears.

35. In the **Query Code**, enter:

```
Select [Org-Unit] Where [Name] = &"Name" And [Org-Unit Type] = &"Type"
```



36. Click **Finish**.



37. Click **Next**.

    The **Dataset Content Method** wizard page appears.

DataSet content method

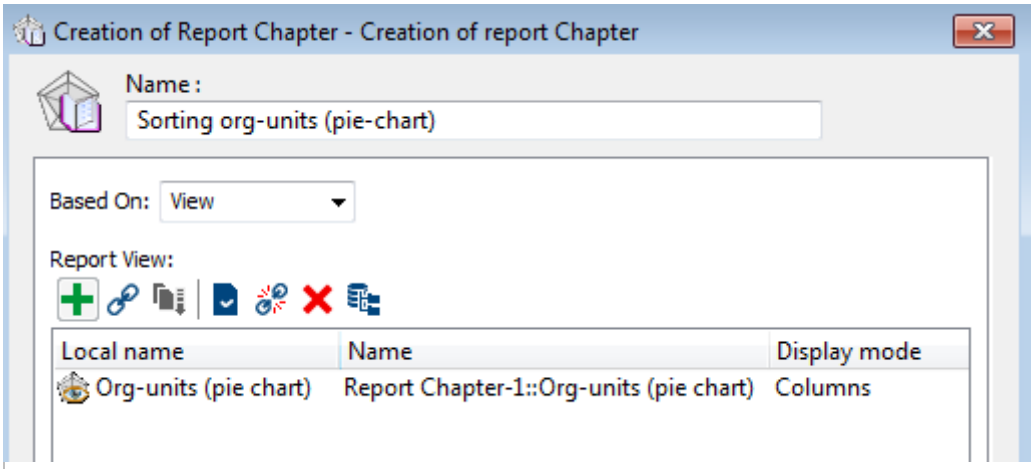
To count the input data.

38. Enter the function code:

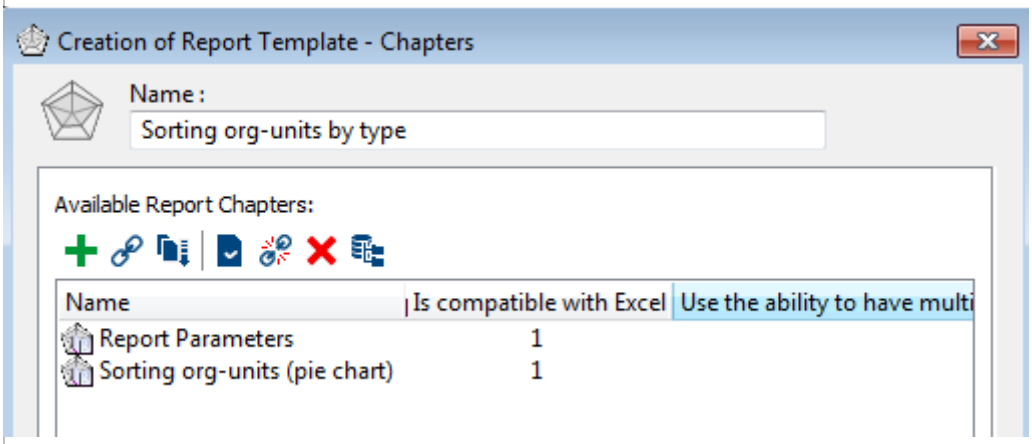
```
Function Compute(input,config)
  Compute = input.count
End Function
```

39. Click **Finish**.

    The Report Chapter is created.



40. Click **Ok**.



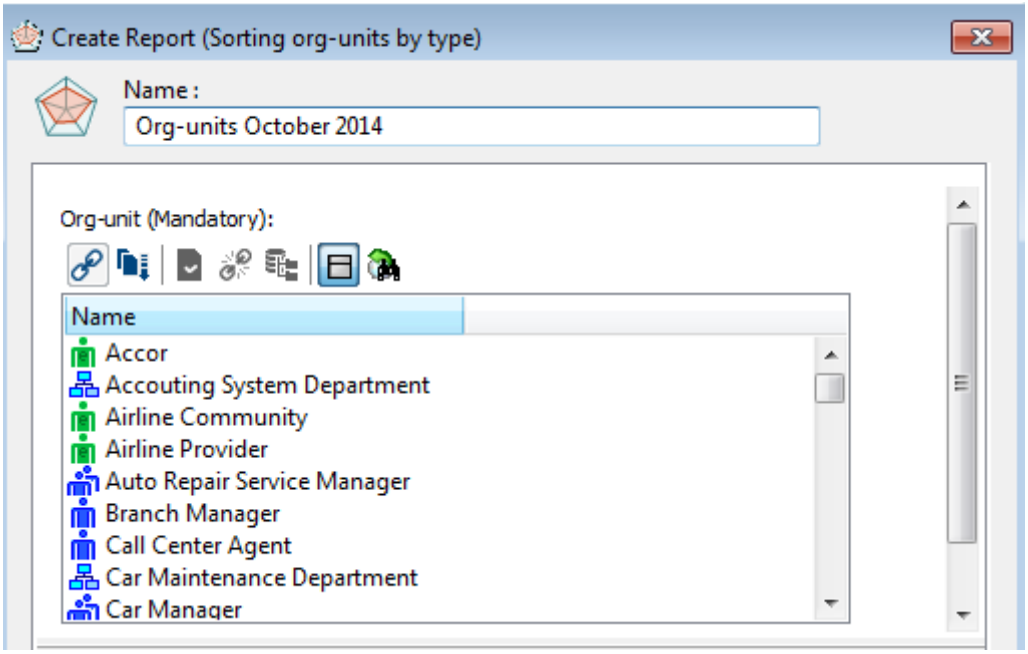
41. Click **Finish**.

4.1.2 Testing your report template

To test your report template:

1. From the **Utilities** navigation window, right your Report Template and select **Test**.
2. In the **Name** field, enter a name for the report.
3. Click **Next**.

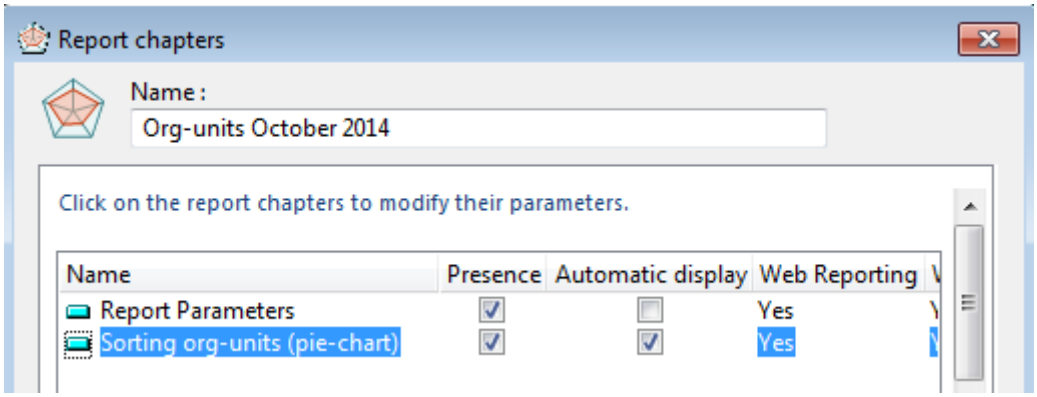
   The parameter selection wizard appears.
4. Select for example all the available org-units.



5. Click **Next**.

   The Report chapters wizard appears.

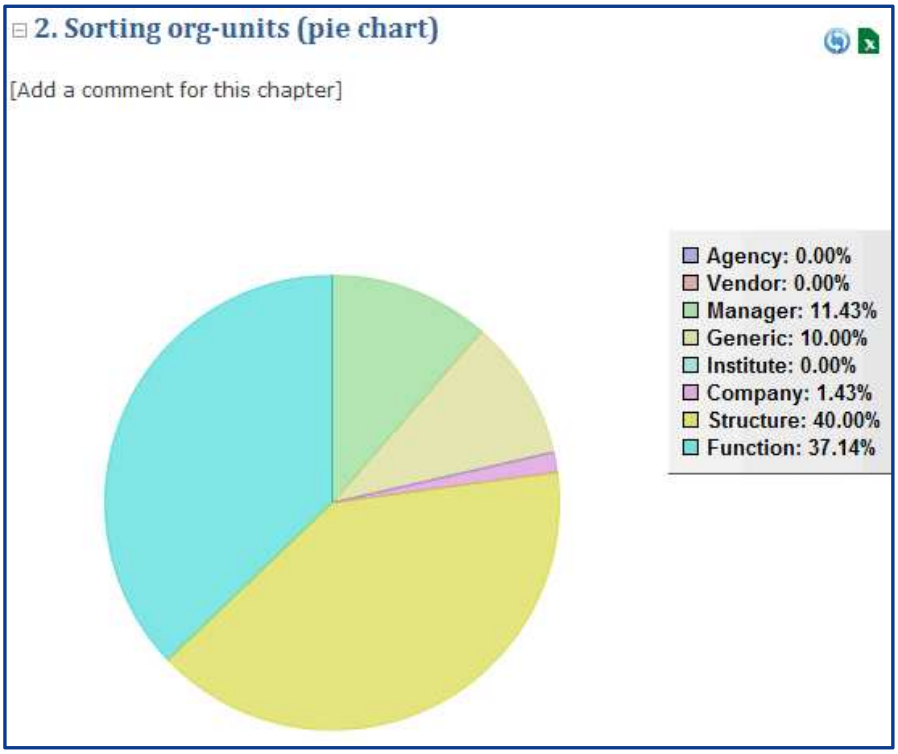
6. In your Chapter line, select **Automatic display**.



7. Click **Finish**.

   The report is displayed.

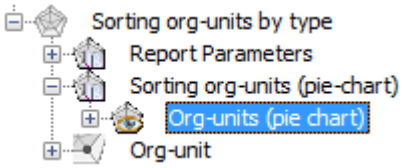
   ☞ To customize the report display, see Customizing your report template p. 20.



4.1.3 Customizing your report template

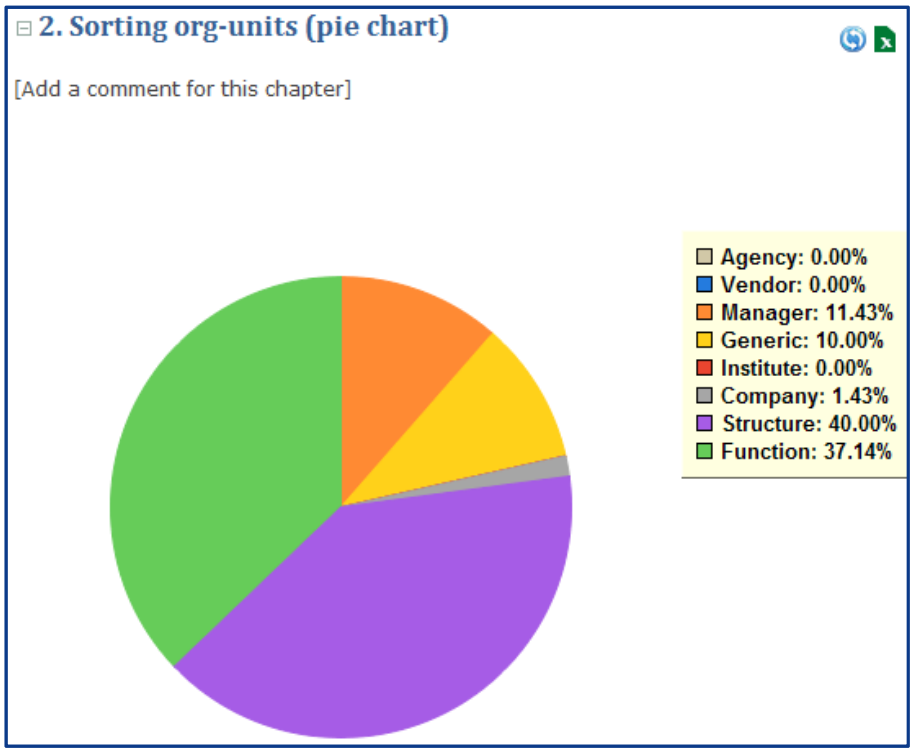
To customize your report display:

1. From the **Utilities** navigation window, access your Report template View properties.



2. In the **Characteristics** tab, in the **RendererType** pane, select **piechart**.
3. The Renderer Parameters are displayed.
4. For example:
   - in the **Colors** field, select “Hopex Pie Colors”.
   - in the **Legend background colors**, select a color.
5. Click **OK**.
6. In the report you already created click **Refresh** .

   Your report display is updated according to your modifications.


2. Sorting org-units (pie chart)
[Add a comment for this chapter]
Agency: 0.00%
Vendor: 0.00%
Manager: 11.43%
Generic: 10.00%
Institute: 0.00%
Company: 1.43%
Structure: 40.00%
Function: 37.14%

4.2 “Sorting org-units by type with different displays” Report Template

Reports generated from the “Sorting org-units by type with different displays” Report Template show in a table and in a bar chart the percentage of org-units sorted according to their type and according to their characteristics (internal/external).

To do so, you have to define a report input parameter “Org-Units” of type Org-unit, from which header item values will be collected (org-unit type and internal/external). The report results (cell values and bar values) will be computed from these header items using a query.

Input data:

- renderer types: Table and bar Chart
- two headers: Org-unit type and org-unit characteristics (internal/external)

Collector:

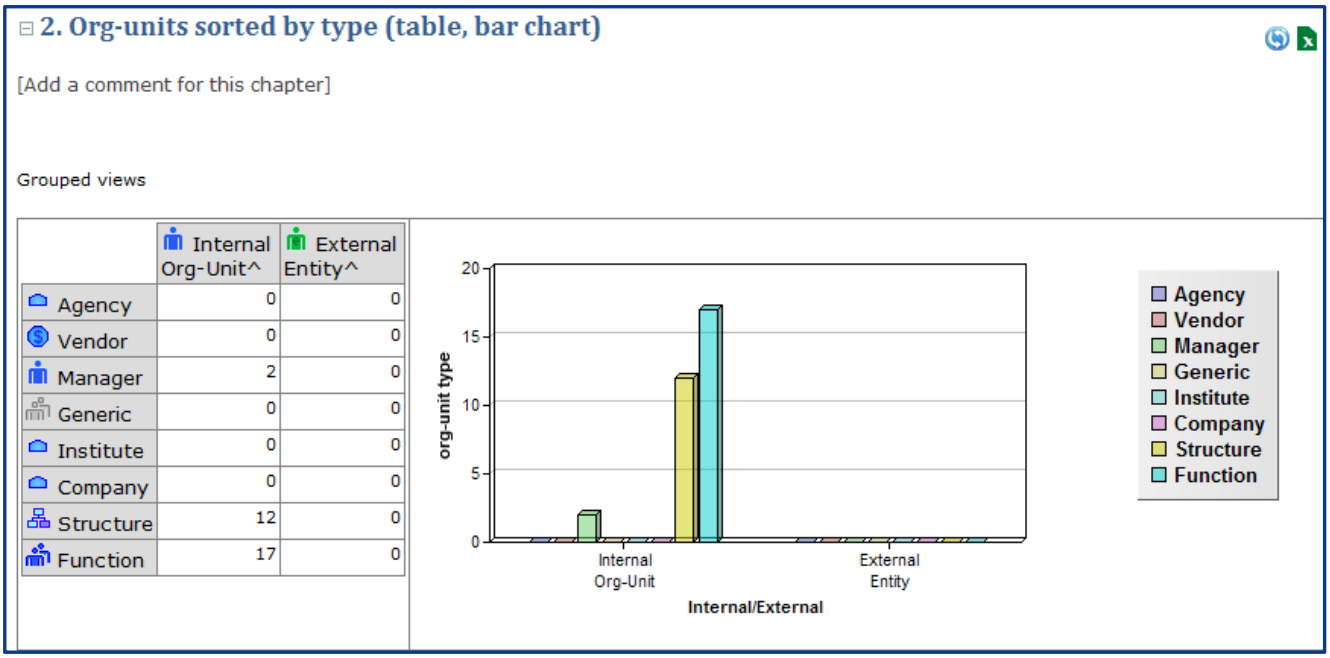
- input parameter: Org-Unit type
- query: **Select** [Org-Unit] **Where** [Name] **= &"Name" And** [Org-Unit Type] **= &"Type" And** [Internal/External] **= &"Internal/External"**

Computation:

- method:

```
Function Compute(input,config)
  Compute = input.count
End Function
```

Generated report example:



4.2.1 Creating a Report Template: “Sorting org-units by type with different displays”

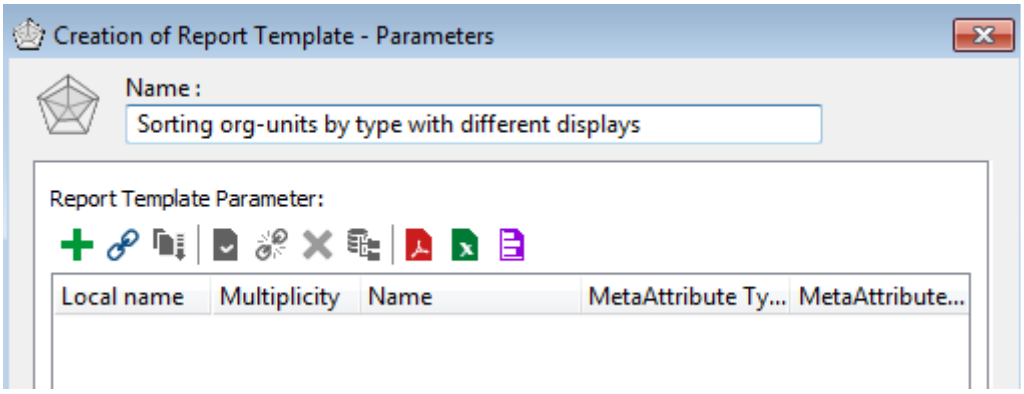
To create the “Sorting org-units by type with different displays” report template:

Report Template creation

1. From the **Utilities** Navigation window, right-click the **Report Templates** folder and select **New** > **Report Template**.

   The **Creation of Report Template** wizard appears.
2. Enter the Report Template **Name** and eventually a comment.
3. Click **Next**.

   The **Parameters** definition wizard page appears.



Parameter creation and definition

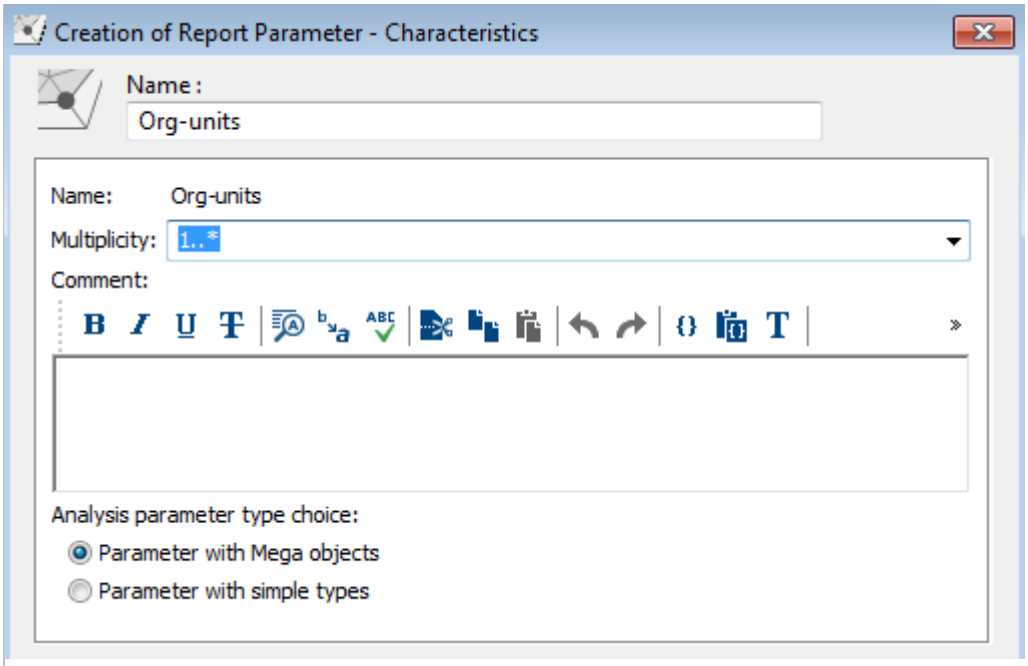
To define the parameter name, multiplicity (the user can select 1 to n occurrences or org-units as report input), and type.

To define to which object type the parameter refers (Org-Unit).

4. In the **Report Template Parameter** pane, click **New**  .

   ☞ For more details on parameters, see Defining new parameters in a report template p. 67.

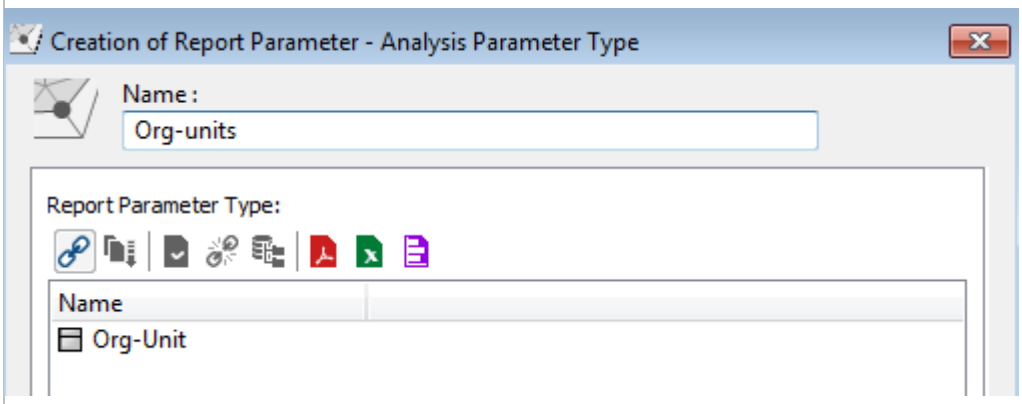
   The **Creation of Report Parameter** wizard appears.
5. Enter the parameter **Name**: “Org-units”.
6. Click **Next**.
7. In the **Multiplicity** field, select “1..*”.
8. In the **Analysis parameter type choice**, keep the “Parameter with MEGA objects” option selected.



9. Click **Next**.

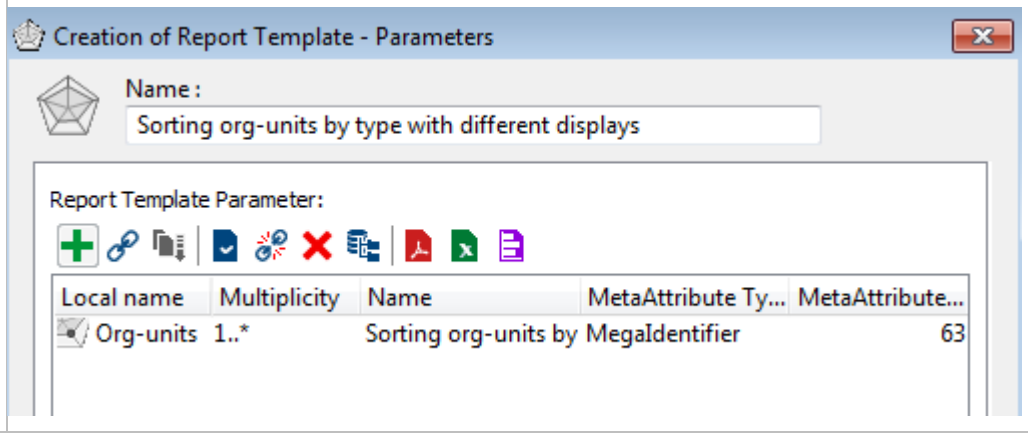
   The **Analysis Parameter Type** wizard page appears.

10. In the **Report Parameter Type** pane, click **Connect**  .
11. In the **Connecting** window, select the **Org-Unit** object type and click **Connect**.



12. Click **Finish**.

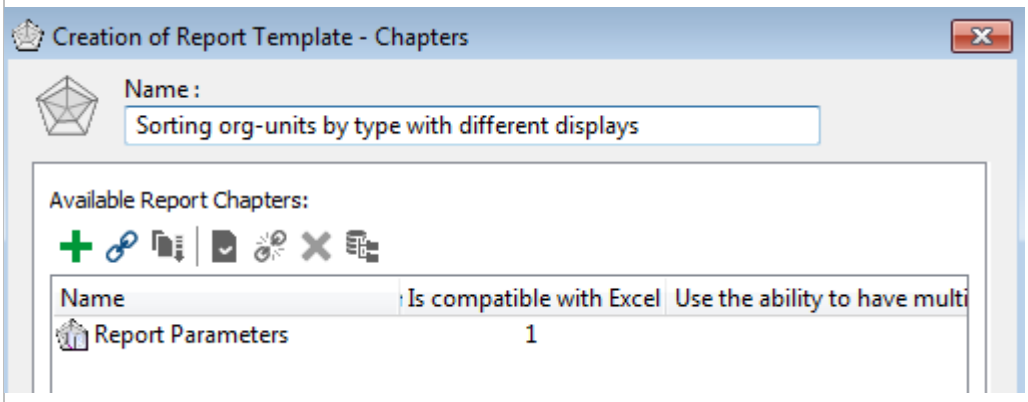
    The “Org-units” parameter is created and defined.



13. Click **Next**.

    The **Chapters** wizard page appears.

    A default chapter is created, it displays the **Report Parameters**.



Chapter creation

To create the report chapter that will display the results of the org-units sorted both by type and by internal/external characteristics.

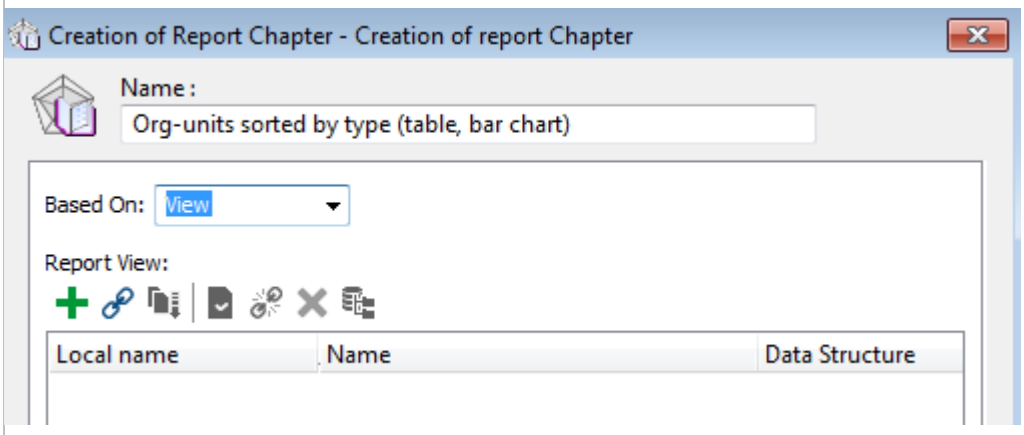
The report chapter is based on a view (grouped views).

14. In the **Available Report Chapters** pane, click **New** .

    The **Creation of report Chapter** wizard page appears.

15. Enter the chapter **Name**.
16. Indicates that it is **Based On** a **View**.

    The **Report View** pane appears.



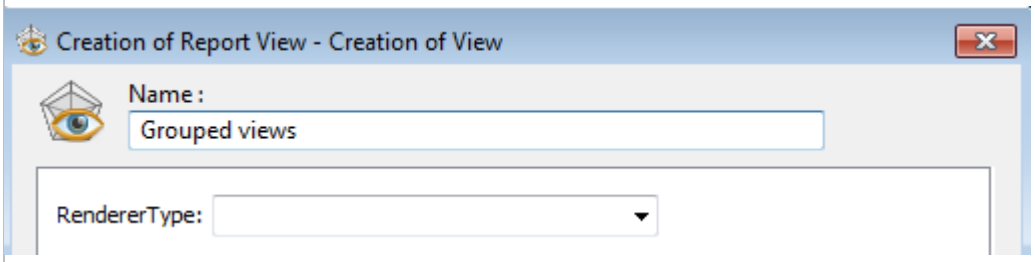
17. In the **Report View** pane, click **New** .

    The **Creation of Report View** wizard appears.

Grouped view creation

To create grouped views (**RendererType** field empty), which will include two views displayed on two columns

18. (optional) Enter the view **Name**.
19. Leave the **RendererType** field empty.



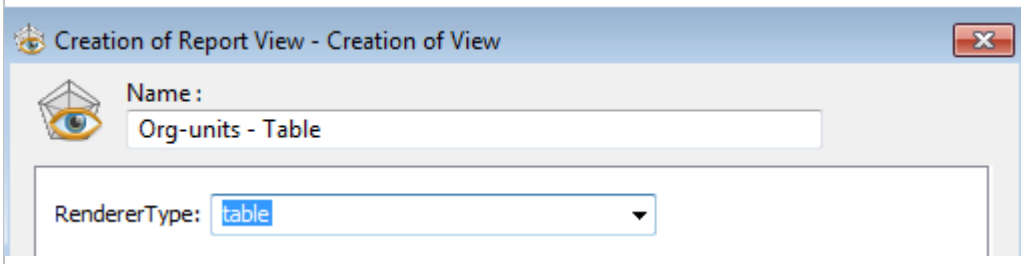
20. Click **Next**.

21. In the **Number of columns** field, enter “2”.
22. In the **Grouped Views** pane click **New**  .

View creation

To create the first view select the renderer type used for data display (table).

23. (optional) Enter the view **Name**.
24. In the **RendererType** field, select “table”.



25. Click **Next**.

    The **Report Headers** wizard page appears

Report Headers definition

The data structure is created with the header number intended by the renderer type display.

Here (table) two headers:

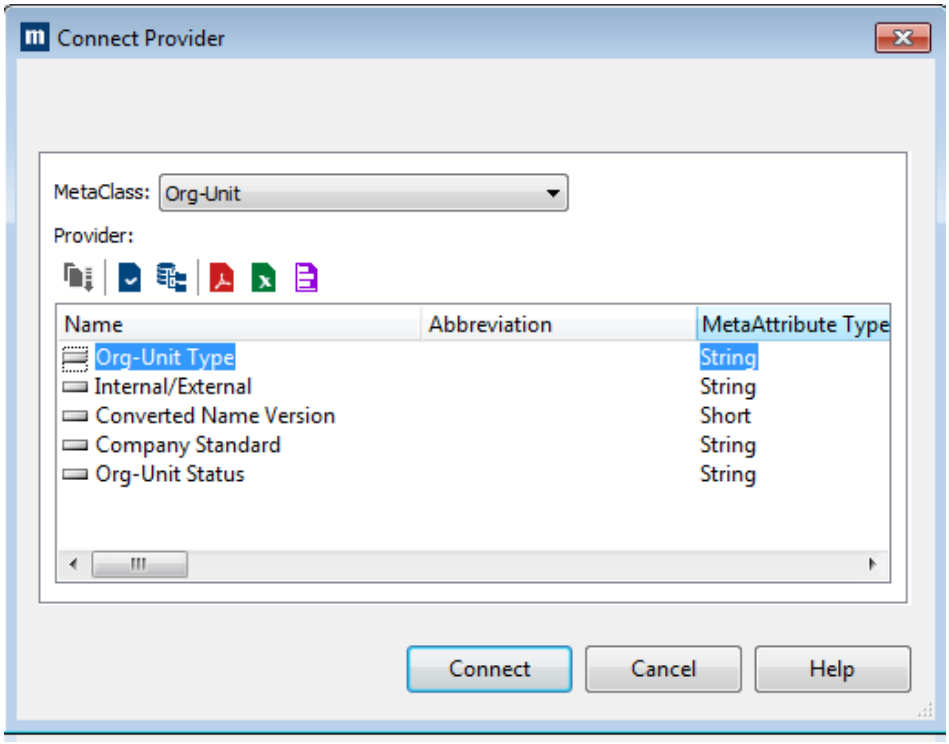
“Org-unit type” and Internal/external characteristics.

Header items are retrieved with a query.

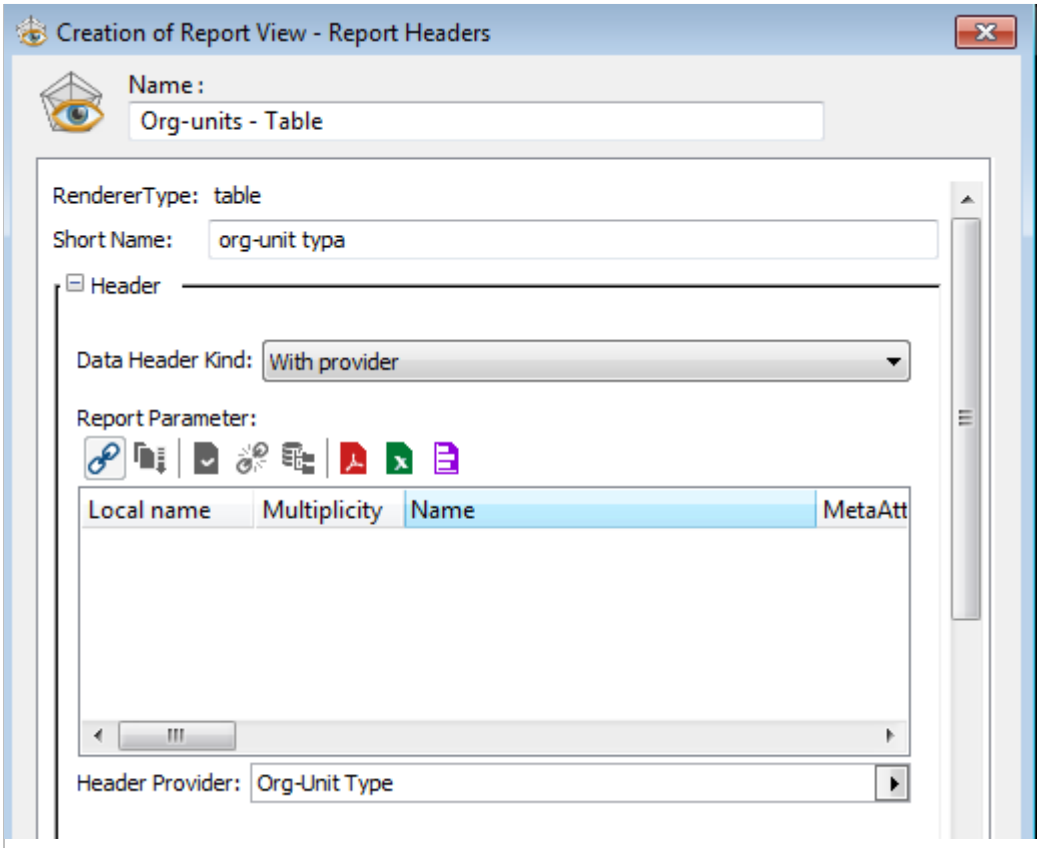
26. In the **Short Name** field, enter the Header name.
27. Define how the header items are retrieved:

    In this example, a query enables you to retrieve the header items from the parameter defined as report template input.

    - in the **Data Header Kind** field, select “with provider”.
    - leave the **Report Parameter** pane empty.
    - In the **Header Provider** field, select “Connect MetaAttribute” (Select MetaClass: Org-Unit, Provider: Org-Unit Type).



28. Click **Connect**.



Org-unit type first header is defined.

29. Click **Next** to define the second header.

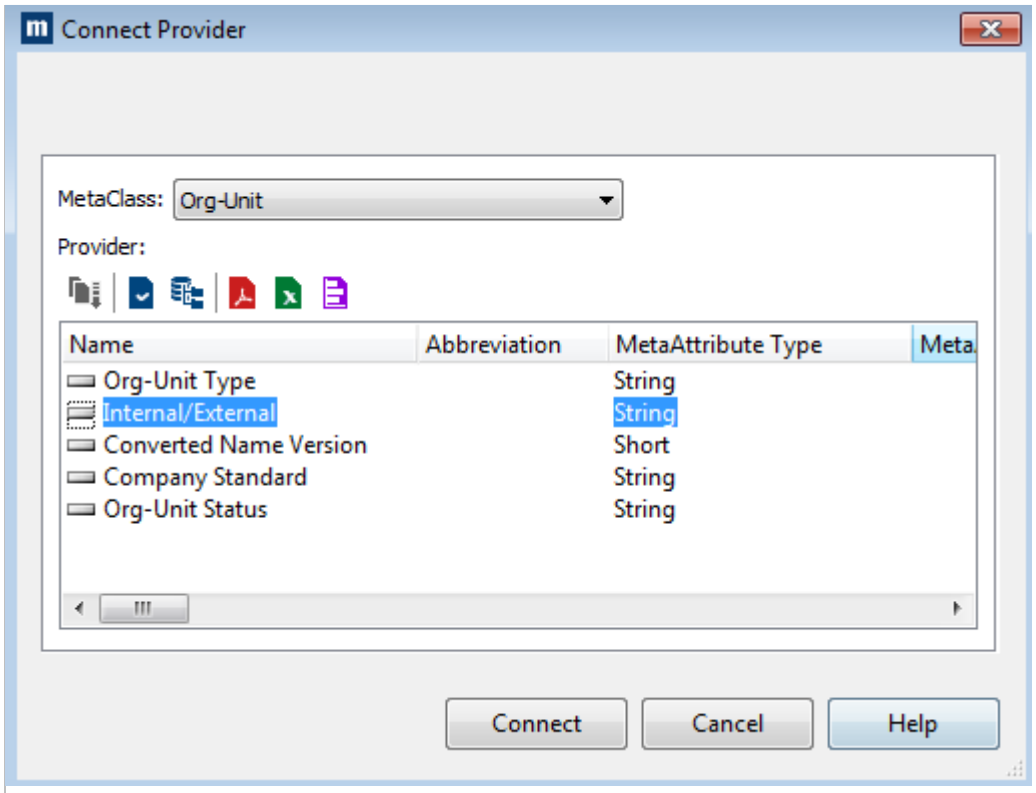
    The second **Report Headers** wizard page appears.

30. In the **Short Name** field, enter the Header name.

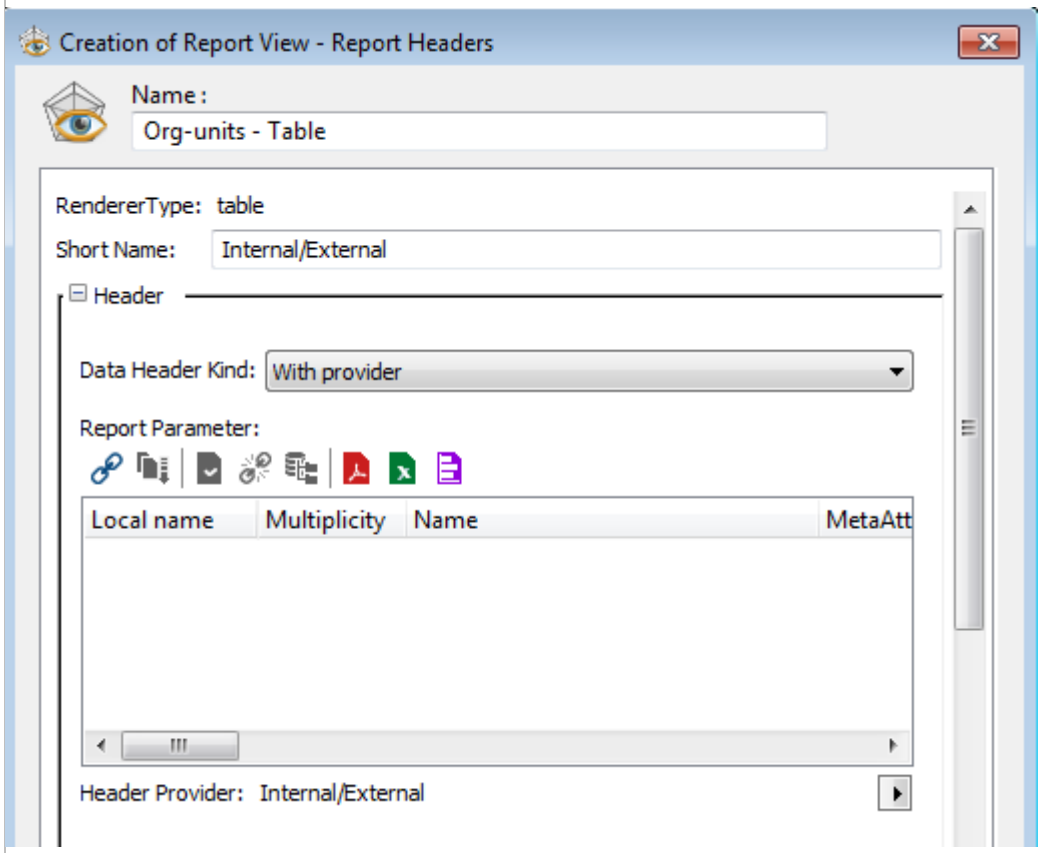
31. Define how the header items are retrieved:

    In this example, a query enables you to retrieve the header items from the parameter defined as report template input.

    - in the **Data Header Kind** field, select “with provider”.
    - leave the **Report Parameter** pane empty.
    - In the **Header Provider** field, select “Connect MetaAttribute” (Select MetaClass: Org-Unit, Provider: Internal/External).



32. Click **Connect**.



Internal/External second header is defined.

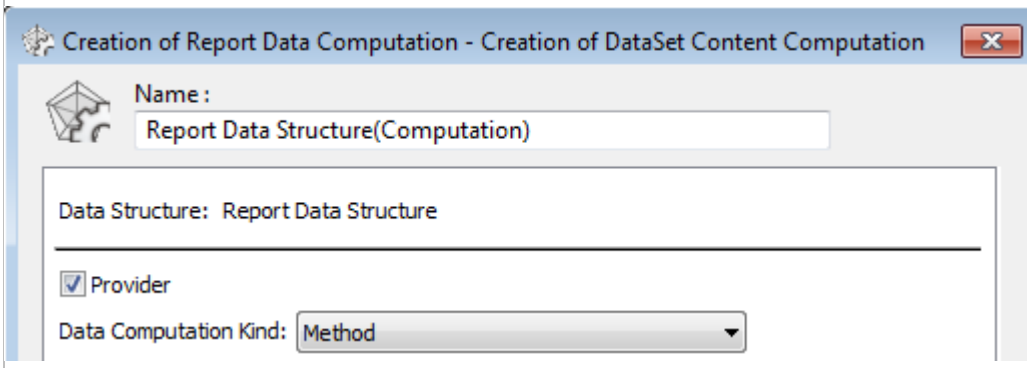
Both headers are defined.

33. Click **Finish**.

    The **Creation of Report Data Computation** wizard appears.

Report data computation creation

34. From the **Creation of DataSet Content Computation** wizard page, in the **Report Data Structure** pane, select **Provider**.
35. In the **Data Computation Kind**, select **Method** computation mode.

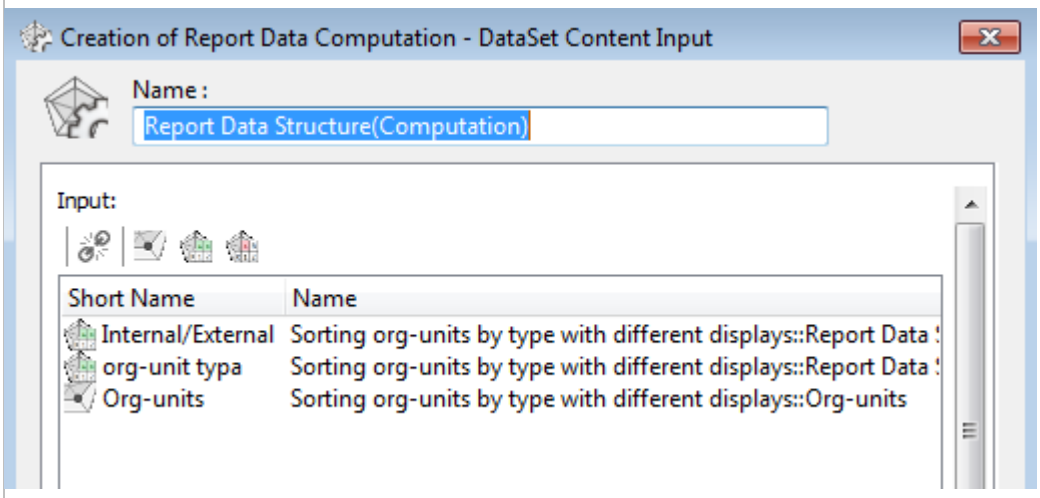


36. Click **Next**.
37. In **Input** pane:
    - click **Connect Report Parameter** and select the corresponding objects (Org-units).
    - click **Connect Header** and select the corresponding objects (Internal/External and org-unit type).

DataSet content computation creation

This wizard defines the input data processing and/or computation, which enable to retrieve the values stored in the data structure and provide them to the renderer.

When a parameter is associated with the provider, this wizard enables to select/create a query.



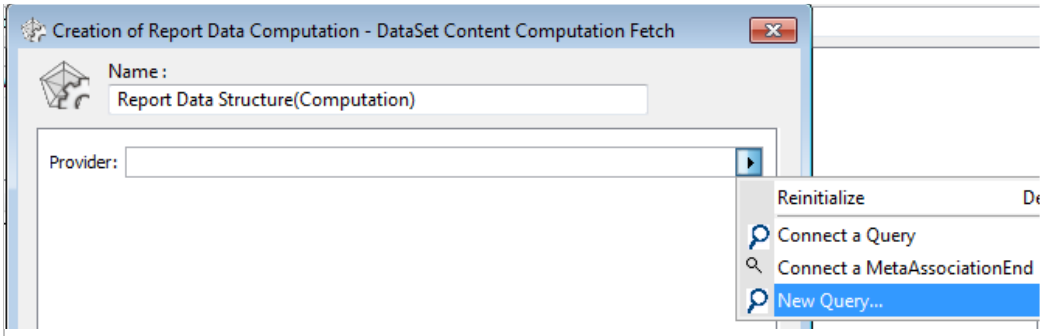
38. Click **Next**.

    The **Dataset Content Computation Provider** wizard page appears.

DataSet content input

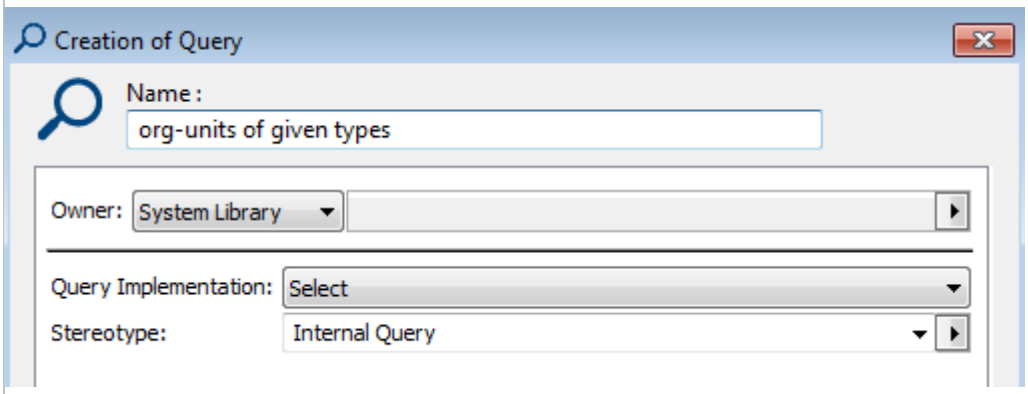
The Provider (through a query) retrieves all the occurrences (org-units) matching the query.

39. In the **Provider** field, select **New Query**.



The **Creation of Query** window appears.

40. In the **Name** field, enter the Query name.
41. In the **Query Implementation** field, select « Select ».
42. In the **Stereotype** field, select « Internal Query ».

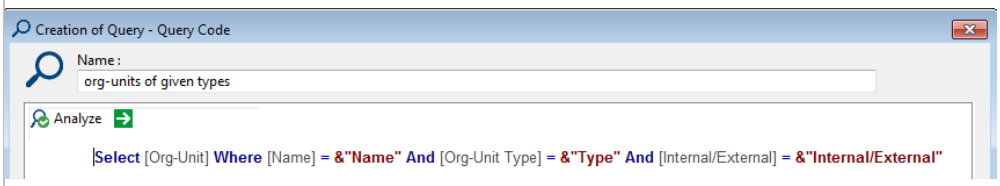


43. Click **Next**.

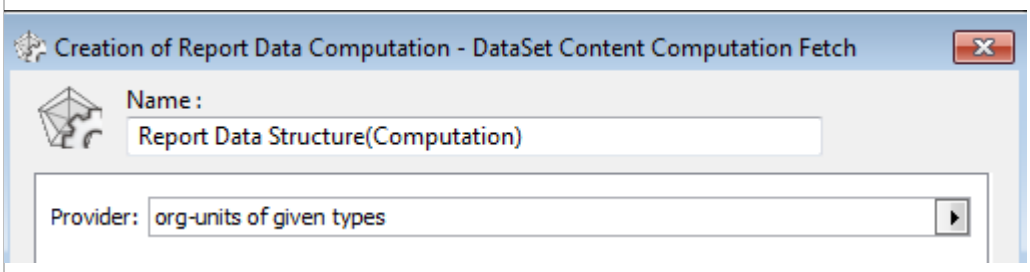
    The **Query Code** wizard page appears.

44. In the **Query Code** wizard page, enter:

    Select [Org-Unit] **Where** [Name] **= &"Name" And** [Org-Unit Type] **= &"Type" And** [Internal/External] **= &"Internal/External"**



45. Click **Finish**.



46. Click **Next**.

    The **Dataset Content Method** wizard page appears.

DataSet content method

To count the input data matching the query.

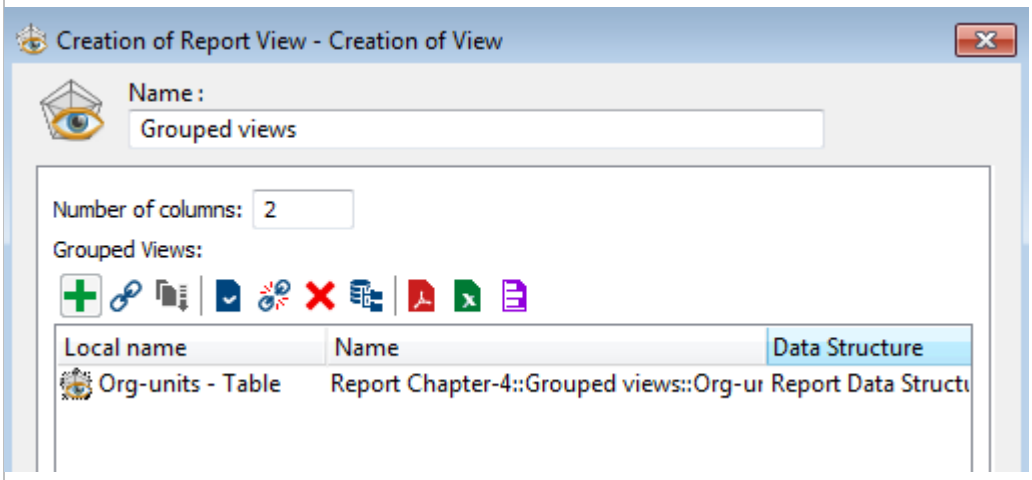
47. Enter the function code:

```
Function Compute(input,config)
  Compute = input.count
End Function
```

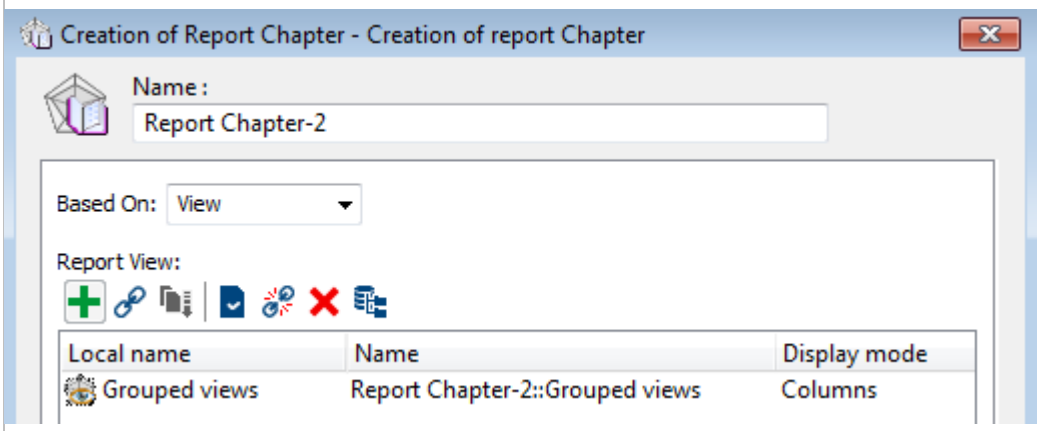
```
'Input is a collection with the following functic
' count()
' item(...)
' avg(oAtt)
' max(oAtt)
' min(oAtt)
' sum(oAtt)
'---
Function Compute(input,config)
 Compute = input.count
End Function
```

48. Click **Finish**.

    The first Report Chapter view is created.



49. Click **Finish**.

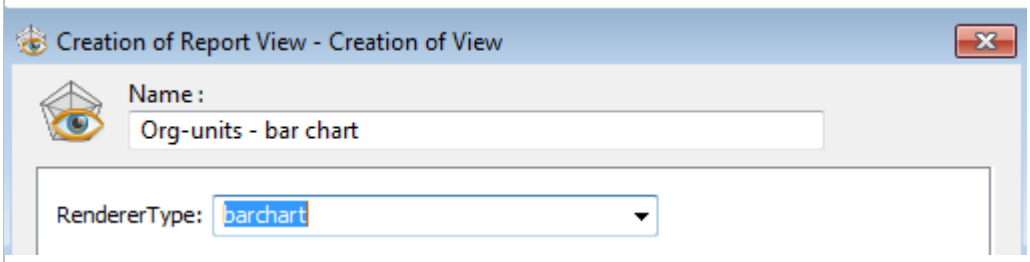


50. In the **Grouped Views** pane click **New**  .

View creation

To create the second view select the renderer type used for data display (bar chart).

51. (optional) Enter the view **Name**.
52. In the **RendererType** field, select “barchart”.



53. Click **Next**.

    The **Report Headers** wizard appears

Report Headers definition

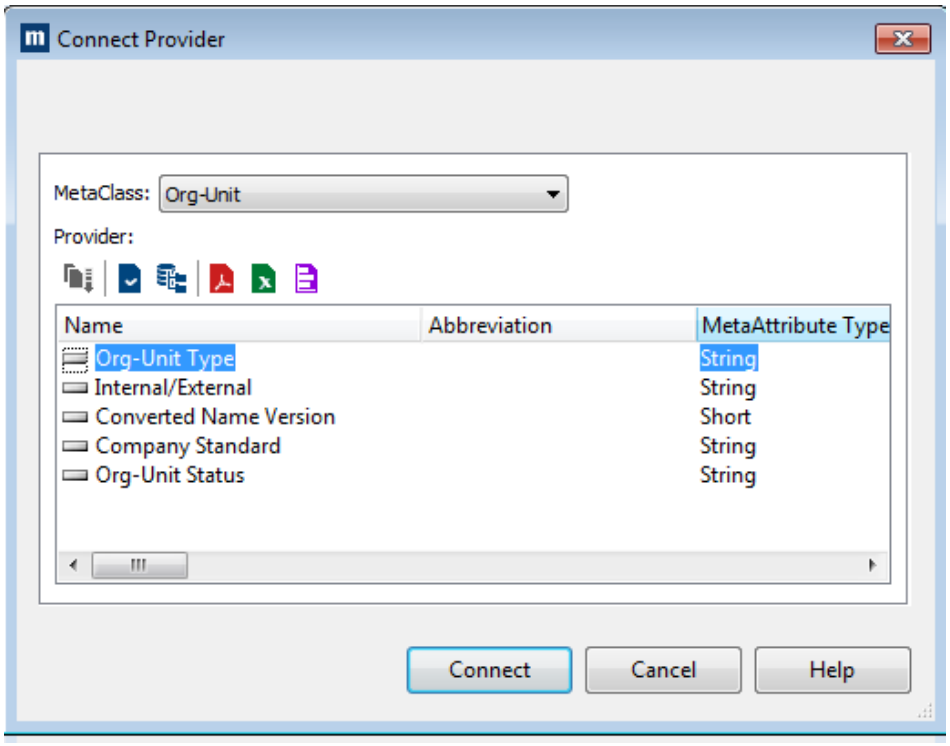
The data structure is created with the header number intended by the renderer type display.

Here (bar chart) two headers: “Org-unit type” and Internal/external characteristics.

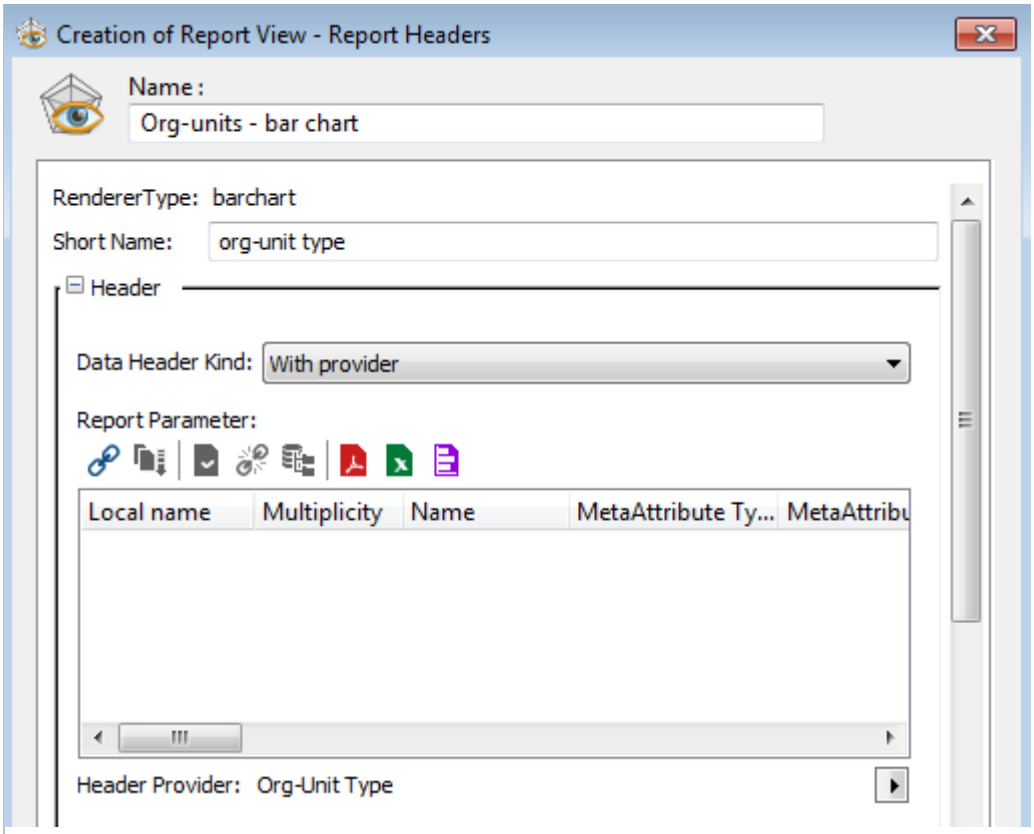
54. In the **Short Name** field, enter the Header name.
55. Define how the header items are retrieved:

    In this example, a query enables you to retrieve the header items from the parameter defined as report template input.

    - in the **Data Header Kind** field, select “with provider”.
    - leave the **Report Parameter** pane empty.
    - In the **Header Provider** field, select “Connect MetaAttribute” (Select MetaClass: Org-Unit, Provider: Org-Unit Type).



56. Click **Connect**.



Org-unit type first header is defined.

57. Click **Next** to define the second header.

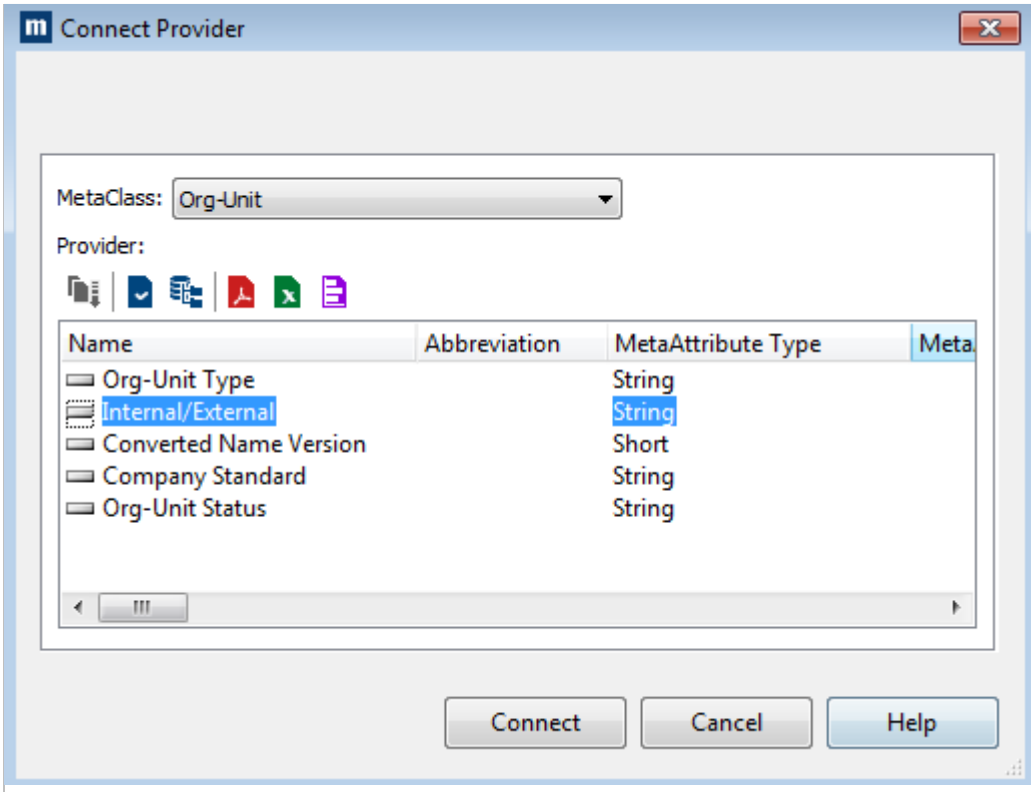
    The second **Report Headers** wizard page appears.

58. In the **Short Name** field, enter the Header name.

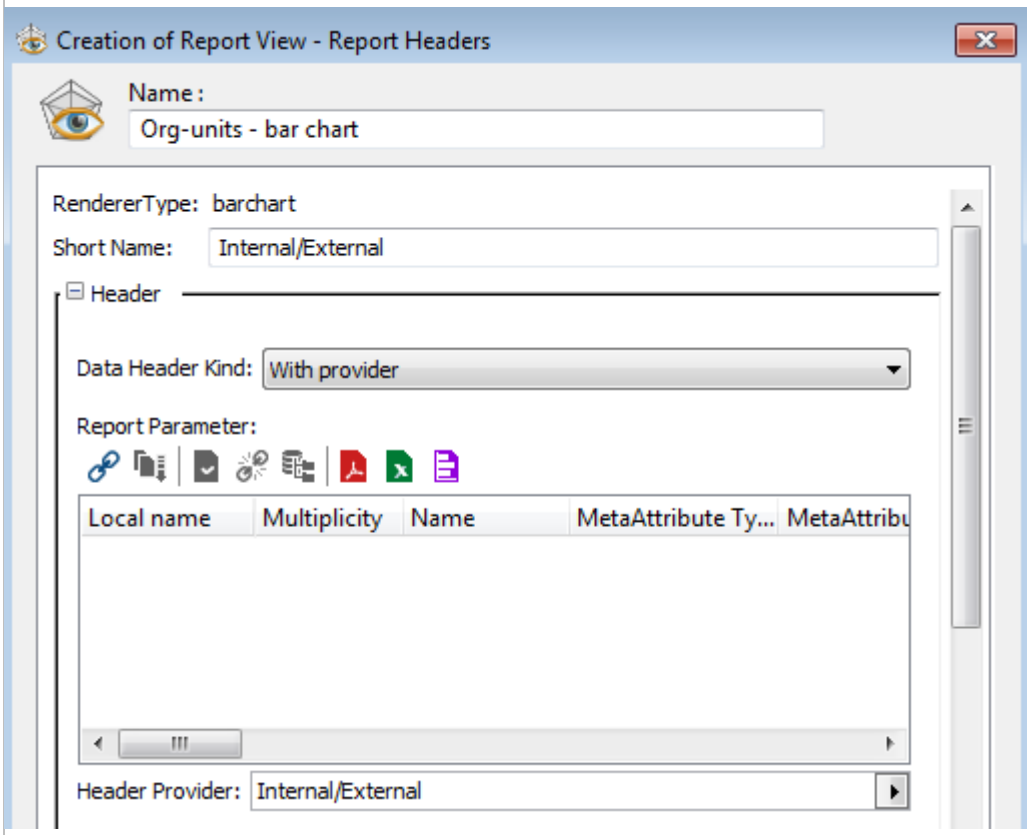
59. Define how the header items are retrieved:

    In this example, a query enables you to retrieve the header items from the parameter defined as report template input.

    - in the **Data Header Kind** field, select “with provider”.
    - leave the **Report Parameter** pane empty.
    - In the **Header Provider** field, select “Connect MetaAttribute” (Select MetaClass: Org-Unit, Provider: Internal/External).



60. Click **Connect**.



Internal/External second header is defined.

Both headers are defined.

61. Click **Finish**.
62. The **Creation of Report Data Computation** wizard appears.

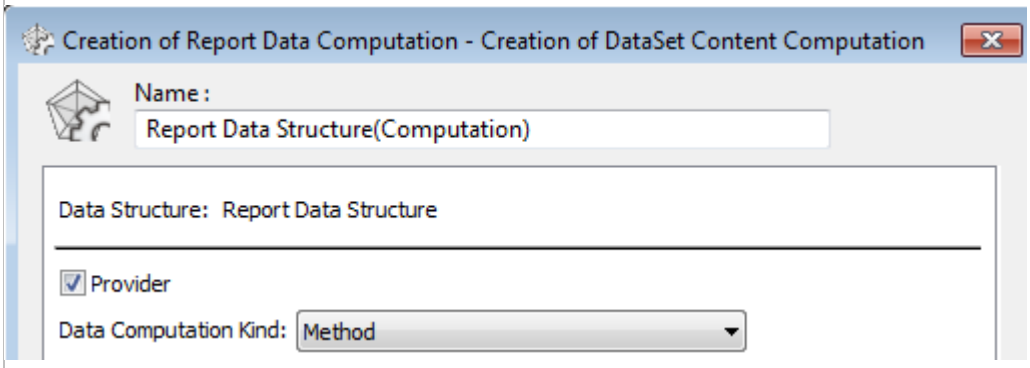
Report data computation creation

DataSet content computation creation

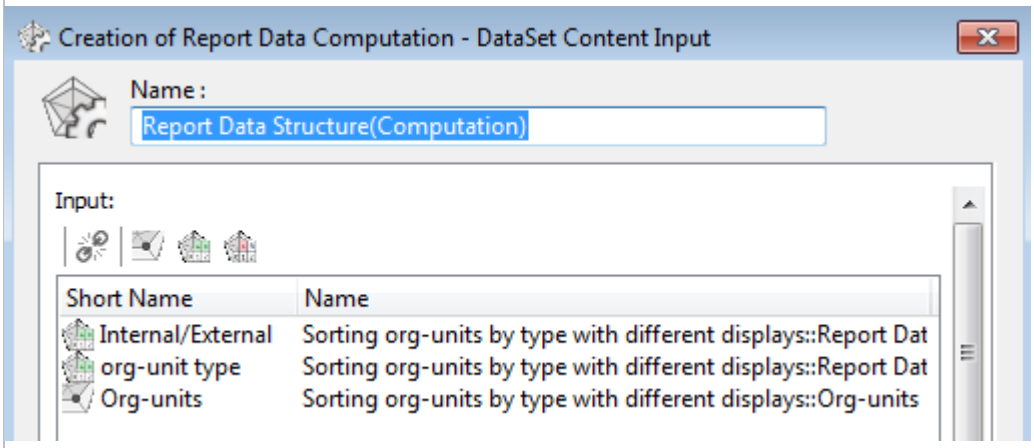
This wizard defines the input data processing and/or computation, which enable to retrieve the values stored in the data structure and provide them to the renderer.

When a parameter is associated with the provider, this wizard enables to select a query.

63. From the **Creation of DataSet Content Computation** wizard page, in the **Report Data Structure** pane, select **Provider**.
64. In the **Data Computation Kind**, select **Method** computation mode.



65. Click **Next**.
66. In the **Input** pane:
    - click **Connect Report Parameter** and select the corresponding objects (Org-units).
    - click **Connect Header** and select the corresponding objects (Internal/External and org-unit type).



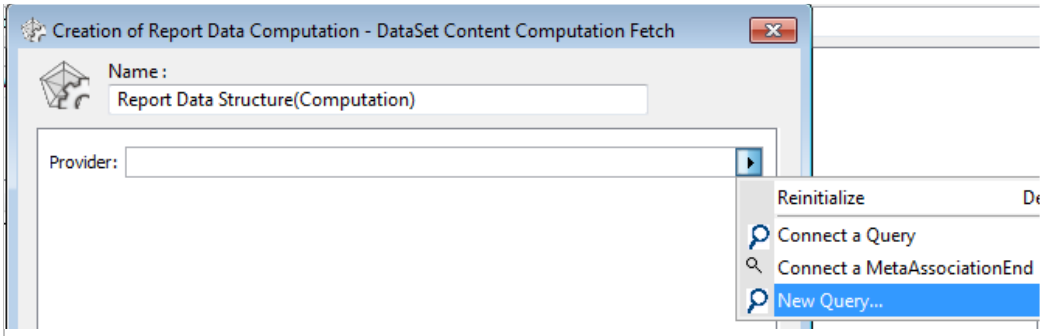
67. Click **Next**.

    The **DataSet Content Computation Provider** wizard page appears.

DataSet content input

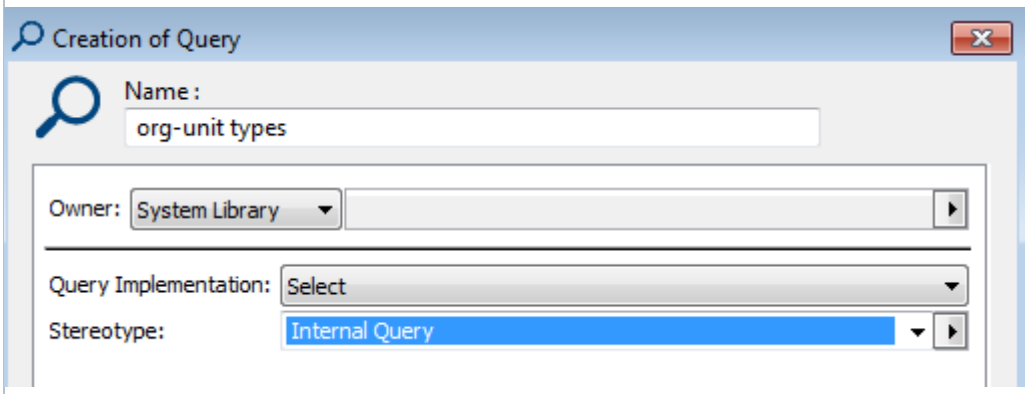
The Provider (through a query) retrieves all the occurrences (org-units) matching the query.

68. In the **Provider** field, select **New Query**.



The **Creation of Query** window appears.

69. In the **Name** field, enter the Query name.
70. In the **Query Implementation** field, select « Select ».
71. In the **Stereotype** field, select Internal « Query ».

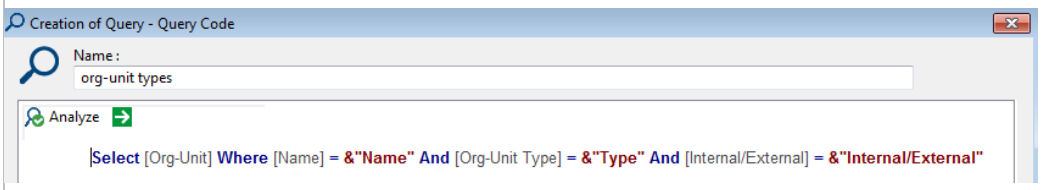


72. Click **Next**.

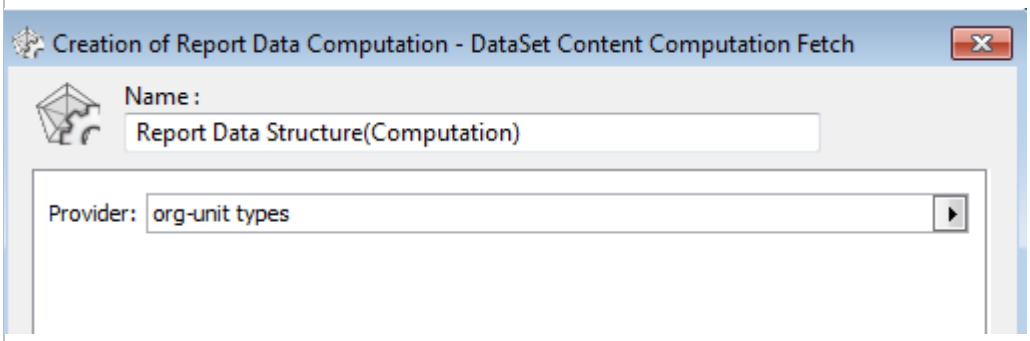
    The **Query Code** wizard page appears.

73. In the **Query Code** wizard page, enter:

    Select [Org-Unit] **Where** [Name] **= &"Name" And** [Org-Unit Type] **= &"Type" And** [Internal/External] **= &"Internal/External"**



74. Click **Finish**.



75. Click **Next**.

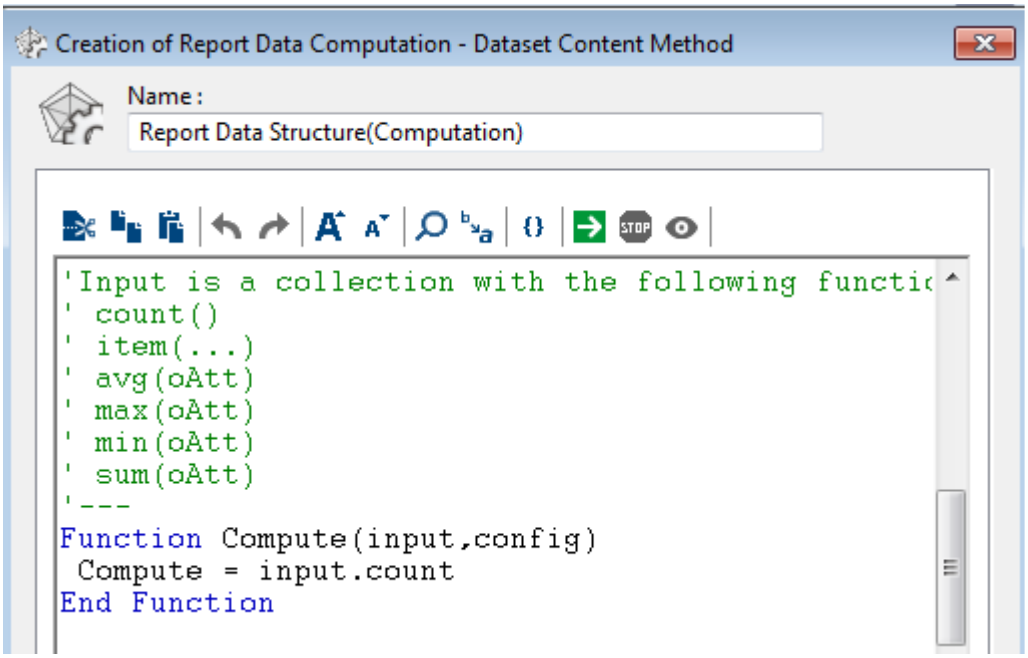
The **Dataset Content Method** wizard page appears.

DataSet content method

To count the input data matching the query.

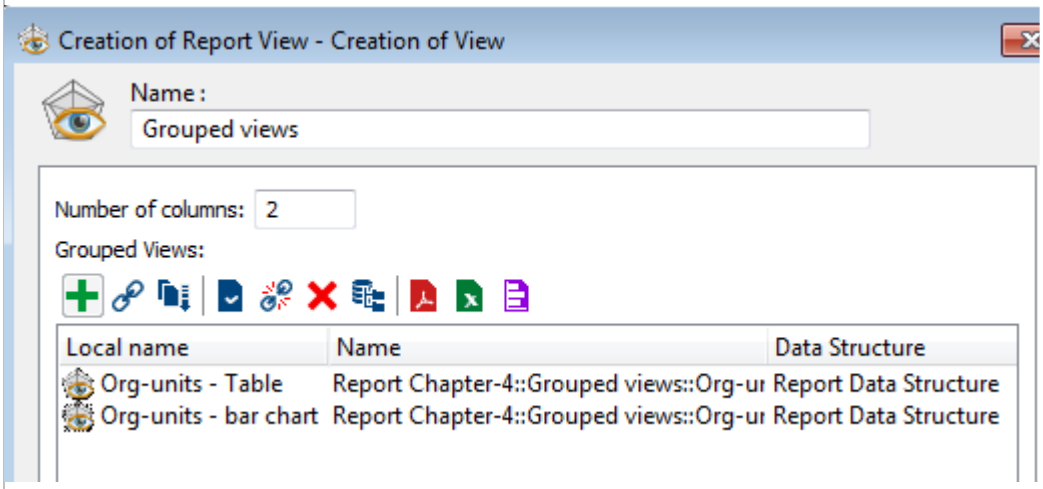
76. Enter the function code:

```
Function Compute(input,config)
  Compute = input.count
End Function
```

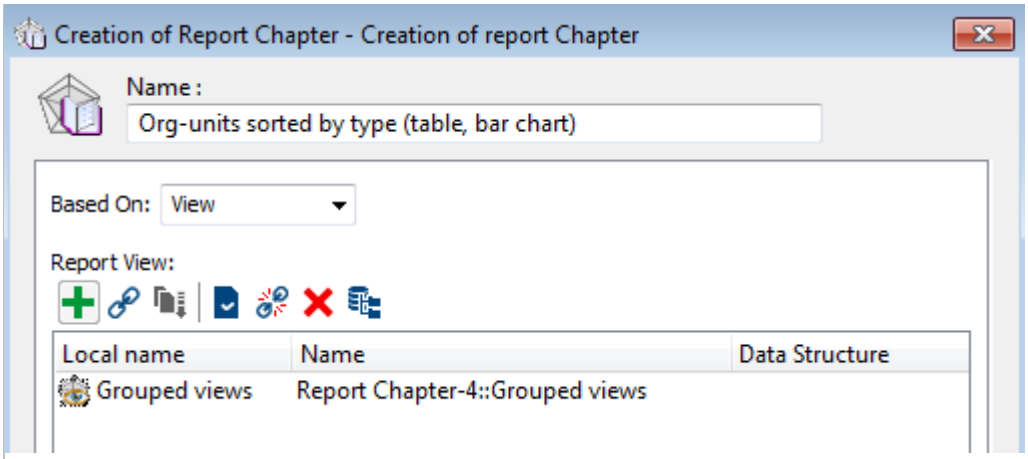


77. Click **Finish**.

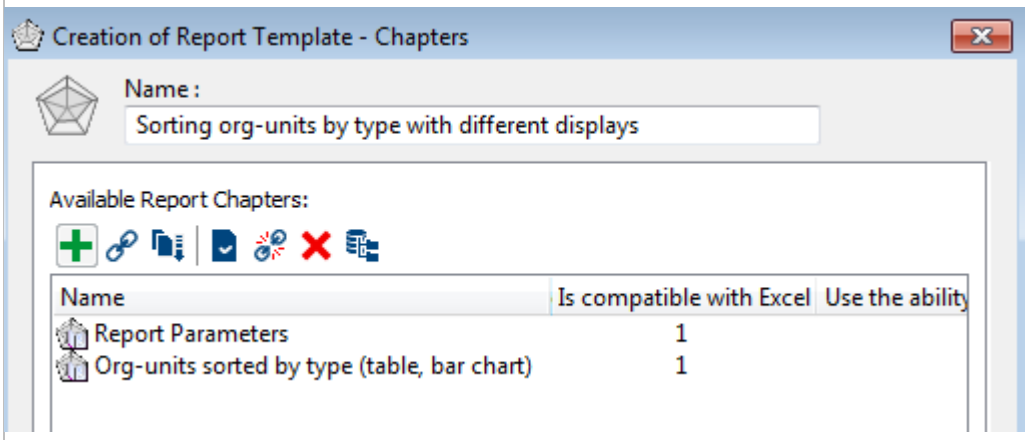
The second Report Chapter View is created.



78. Click **Finish**.



79. Click **Finish.**
80. Click **OK**.



81. Click **Finish**.

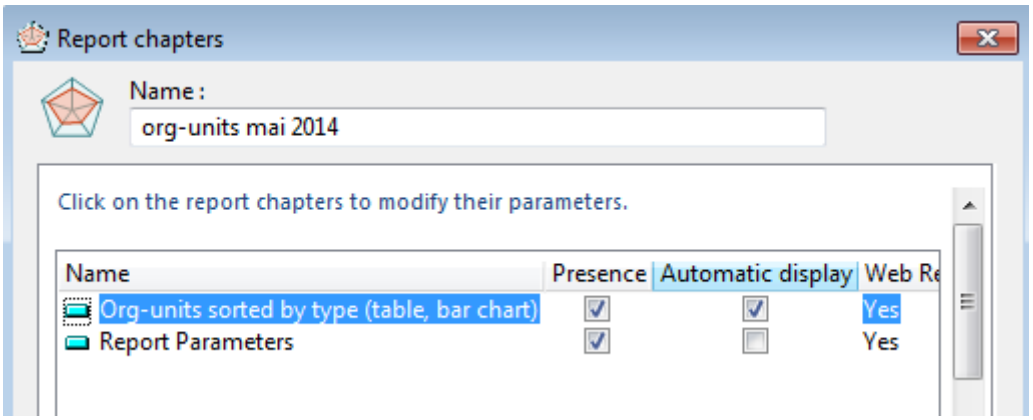
4.2.2 Testing your report template

To test your report template:

1. From the **Utilities** navigation window, right your Report Template and select **Test**.
2. In the **Name** field, enter a name for the report.
3. Click **Next**.

   The parameter selection wizard appears.
4. Select for example all the available org-units.
5. Click **Next**.

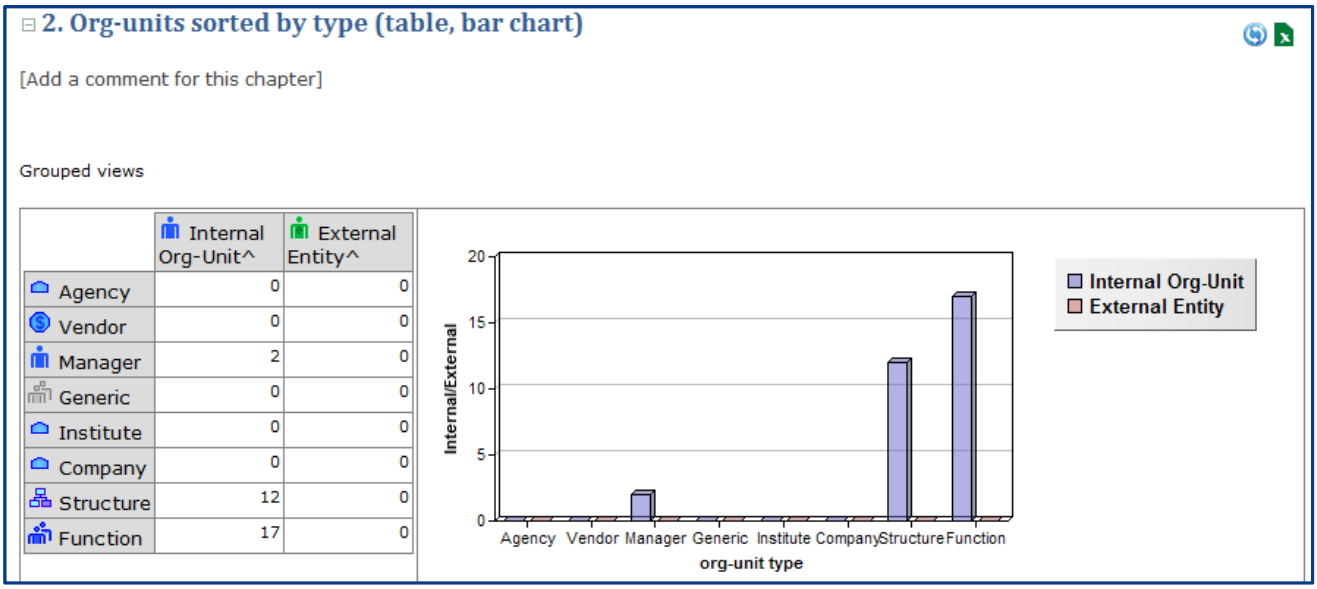
   The **Report chapters** wizard appears.
6. In your Chapter line, select **Automatic display**.



7. Click **Finish**.

   The report is displayed.

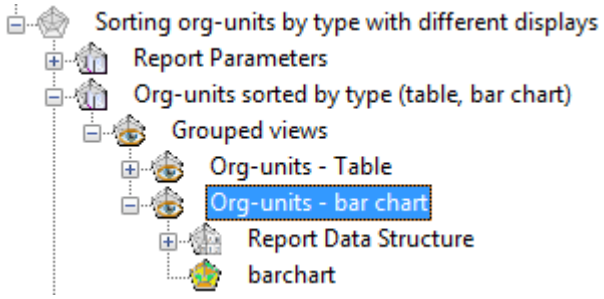
   To modify the bar-chart display see Modifying your report display p. 40.



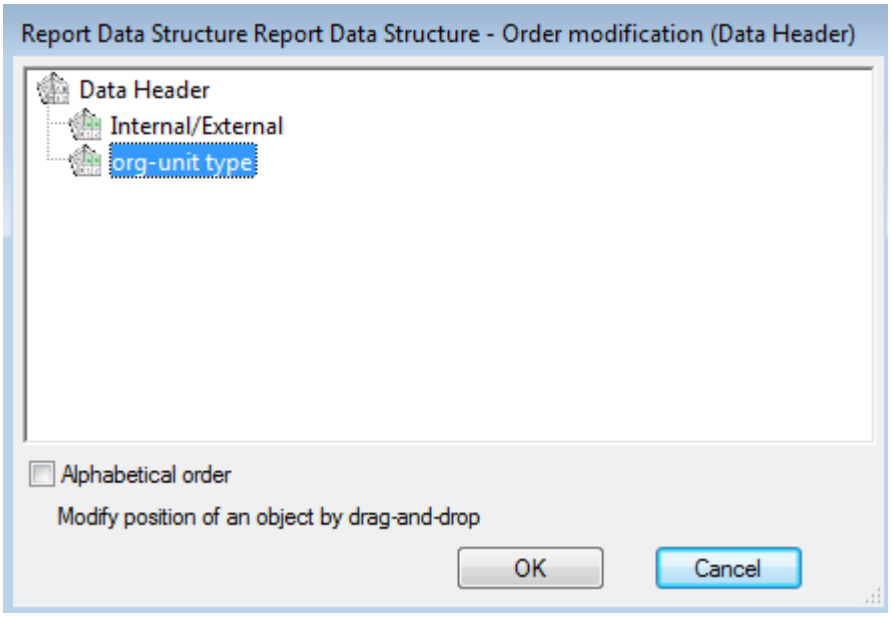
4.2.3 Modifying your report display

To switch the input data in the bar chart axes in your report display:

1. From the **Utilities** navigation window, access your bar chart view properties.



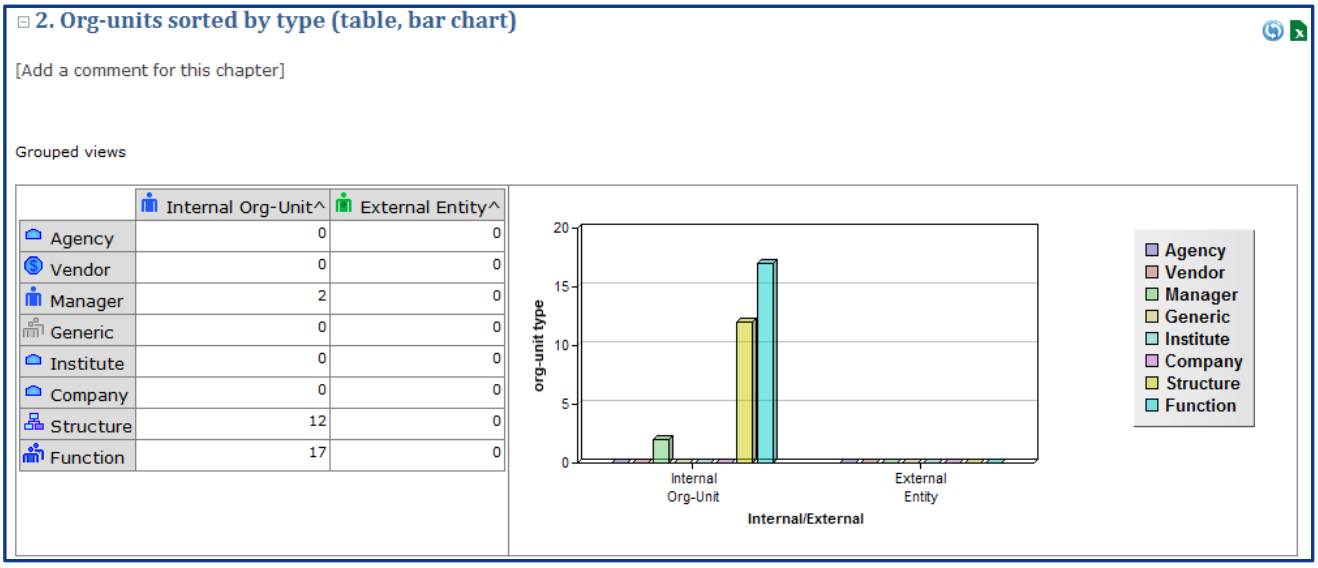
2. Click the **Headers** tab.
3. Click **Reorganize** .
4. In **Order modification (Data Header)**, modify the header order.



The first header represents X-axis and the second header represents Y-axis.

5. Click **OK**.
6. In the report you already created click **Refresh** .

Your report display is updated according to your modifications.



To change the Grouped Views display from Column toTab display:

1. From the **Utilities** navigation window, access your grouped view properties.
2. From the **Characteristics** tab, in the **Display mode** field select “Tabs”
3. Click **OK**.
4. In the report you already created click **Refresh** .

   Your report display is updated according to your modifications.

Grouped views

Org-units - Table | Org-units - bar chart

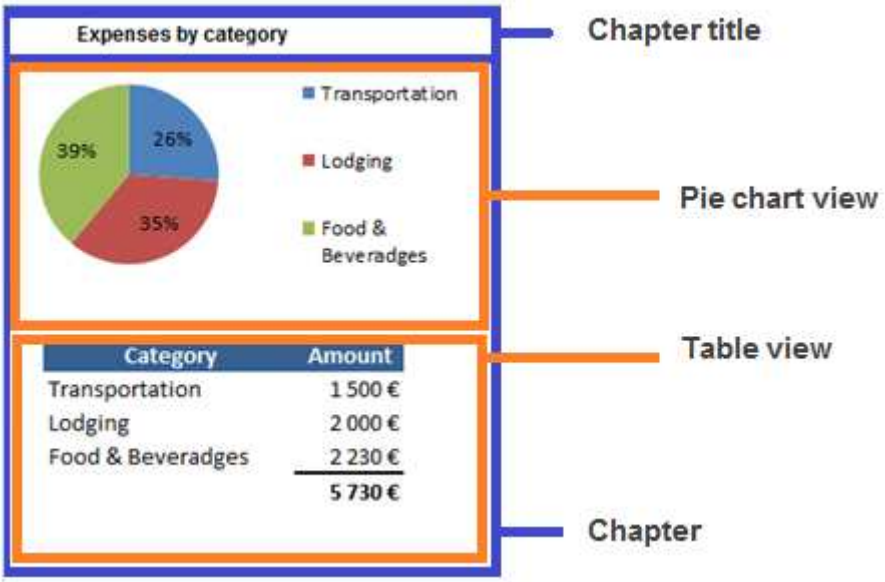
	Internal Org-Unit	External Entity
Agency	0	0
Vendor	0	0
Manager	2	0
Generic	0	0
Institute	0	0
Company	0	0
Structure	12	0
Function	17	0

5 Concepts

5.1 Chapter

The chapter creation is based on either a macro, a view, or a set of views that can be combined together (**Grouped views**).

The chapter organization and display is defined through these macro, view, or set of views.



5.2 Macro

Chapters can be implemented using a Java macro.

☞ See *Writing Java Report Chapters* Technical Article.

5.3 View

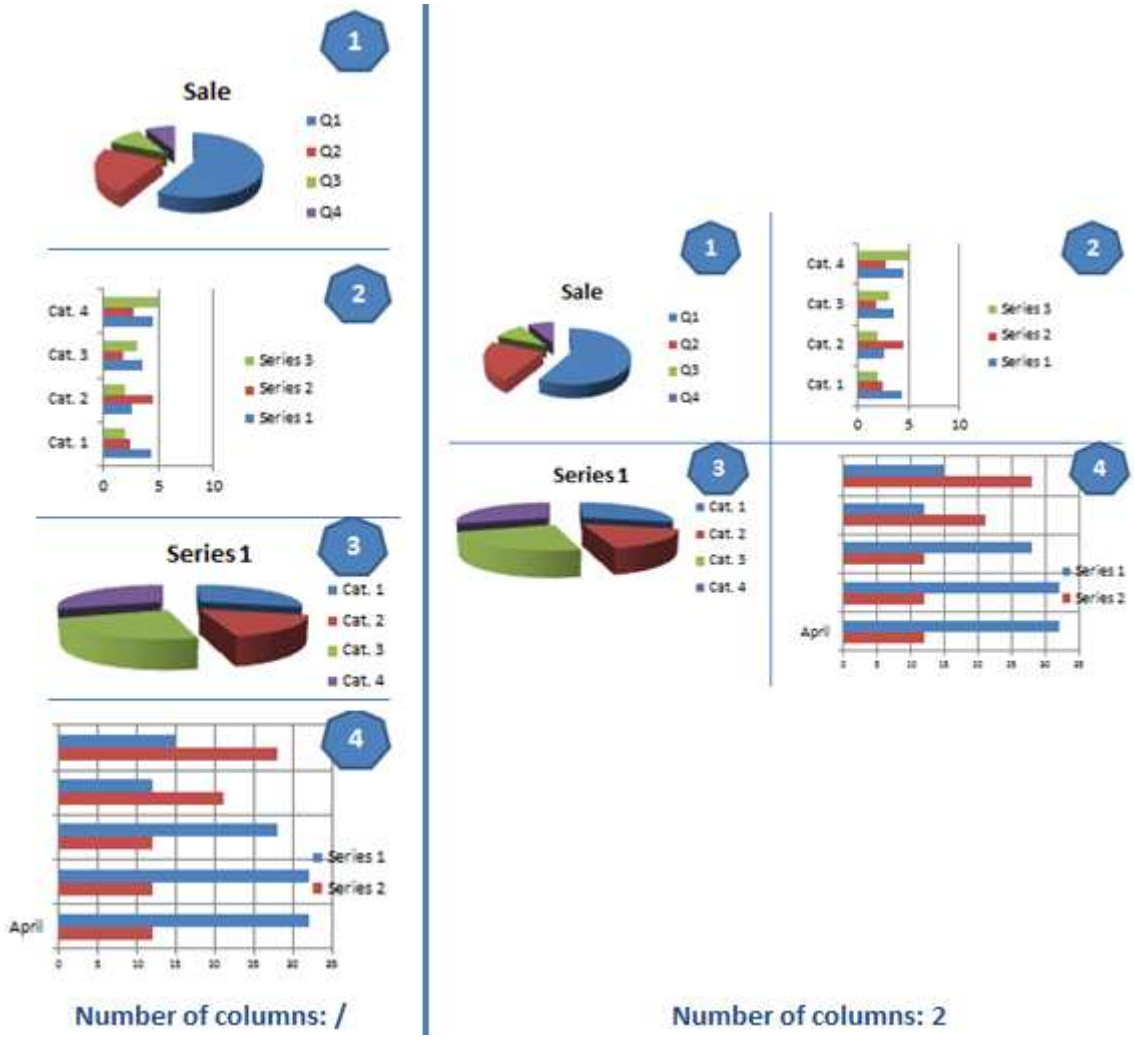
A view is a chapter area that shows a set of values (**Data Structure**) displayed graphically or not according to the **Renderer type** selected

5.3.1 Organization

The chapter views are organized in line. By default the views are displayed in a single column.

You can include several views (**Grouped views**) in a single view. You can organize the **Grouped views** in several columns in the same line (specified through the **Number of columns**).

The following displays four grouped views in a single column (**Number of columns** field empty) and the same four grouped view in two columns (**Number of column** field value = 2):



You can change the **Display mode** grouped view for “Tabs”, for example when your views are two large to be displayed in several columns.

Display mode: Tabs

5.3.2 Content

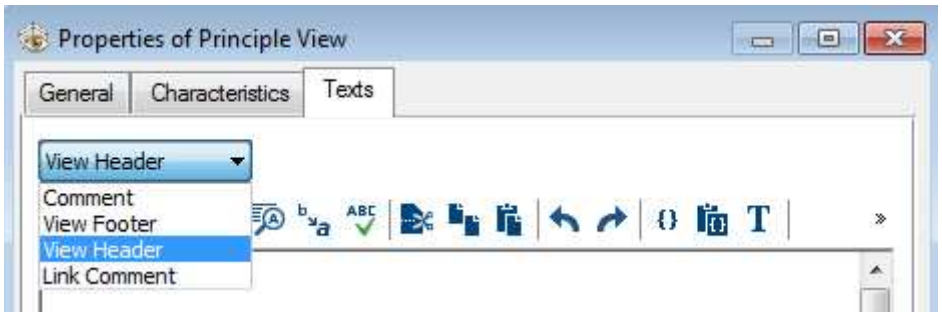
Each view includes either a set of views, or a content.

This content is described with:

- a **Data structure**, which defines the set of processed values, and
- a **Renderer type**, which defines the processed value display.

A view can be associated with several renderer types, which enable several displays (in tab display) of the same data structure.

A view can also have a Header and Footer text that wil be displayed



5.3.3 Renderer types

The available renderer types are the following:

- areachart
- barchart
- linechart
- list
- gauge
- piechart
- radarchart
- table

- verticaltexttable

You can customize each renderer display, see Customizing the renderer display p. 76.

5.4 Data structure

The data structure provides the data to the renderer type. The data structure can include as many headers as needed by the render type. The renderer type choice defines the header number.

Data structure headers are made up of either:

- a set of business occurrences (e.g.: applications of the portfolio used as a report parameter),
- a set of attribute values (e.g.: application types),
- attributes, or
- character strings (e.g.: code templates),

Header number examples:

- a single header: list, piechart
- two headers: table, stacked bar, linechart:

5.5 Data headers

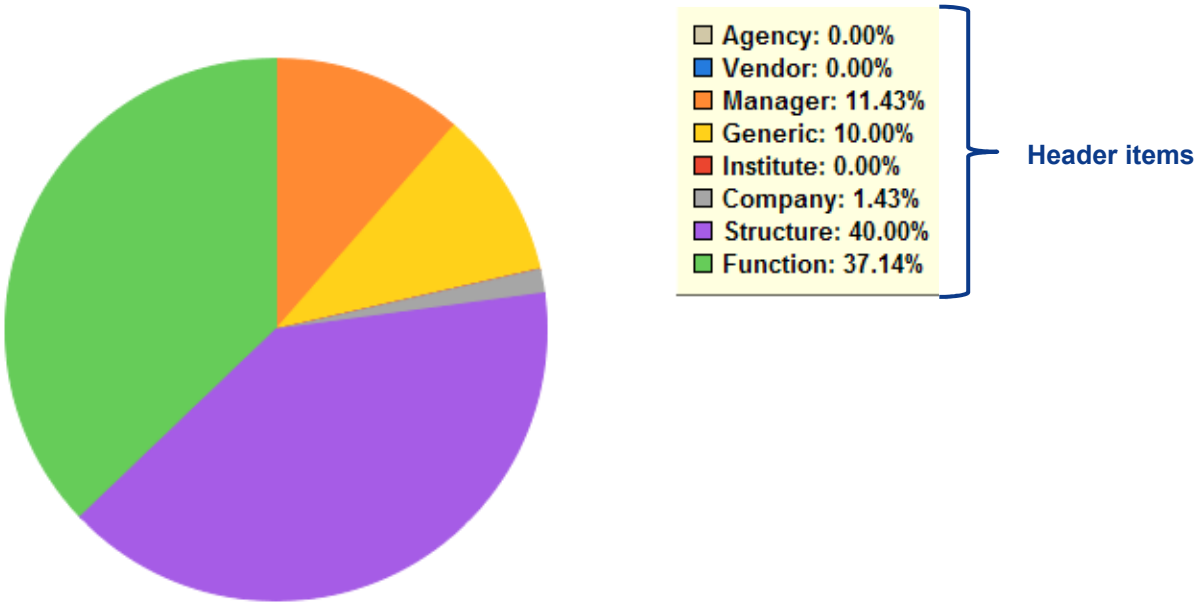
Report legend is built from headers.

The data structure header consists of a set of items.

Examples:

The « Application category » header, for which items correspond to several application categories.

Org-units sorted according to each org-unit type as shown in the following pie chart (header: Org-units, header items: org-unit types).



These items are processed to feed one of the table or graph axes.

This set of header items can be explicit, provided by a query or a MetaAssociationEnd, or retrieved from input parameter.

Data Header Kind attribute defines how the header items are retrieved:

- **Header With Provider**

  Header items are occurrences provided by a collector (Data Header Provider). The collector is a query or a MetaAssocitionEnd).

  The set of header items is computed (**Provider**). The computation is based on either:

  - a query with or without parameter.

    If the query includes a parameter, the set of elements is fed from a report type input parameter associated with the header (necessarily an occurrence).

    Example: with a parameter associated with a “Portfolio” header type, the “Application of ptf” query enables to retrieve these header items, which are the portfolio applications.

  - a MetaAssociationEnd, which is used to retrieve a set of occurrences linked to the occurrence, which is a header parameter.
  - a tabulated value attribute/abstract property: values are directly exploited as as many header items.

    Example: with the « audit plan categories » attribute, header items are made up with the different values of the audit plan categories, in the order that linked them to their attribute.

- **Explicit header Items**

  Each header item is described by an attribute with a unique computation for all of them.

The value displayed by the renderer type is the MetaAttibute name, but you can overload it by using the _guiname of the item.

Example: the following explicit header items are based on attributes.

Code	Recommandation Name	Recommandation owner	User Code	Action	Action Description	Action Type

- **Explicit header Items with computation**

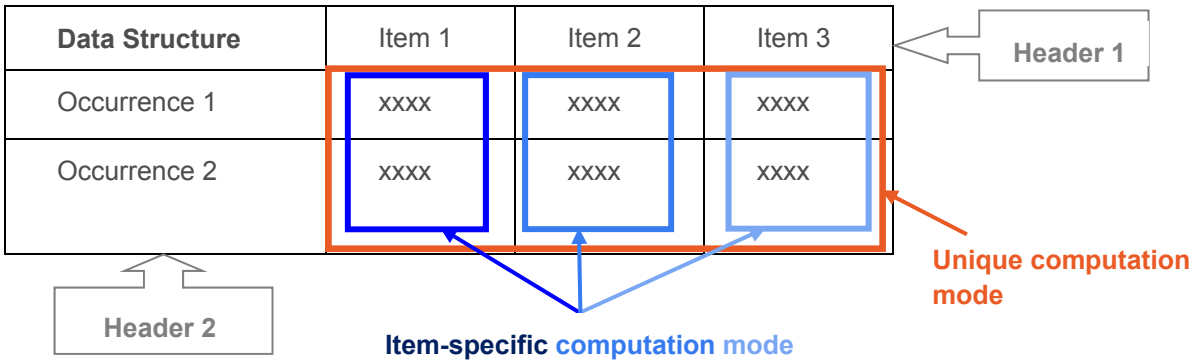
  Each item is described (name and corresponding attribute) and a specific Data Computation is applied to each of them.

5.6 Data computation

Each report value is computed from current header items (one item for a single-header structure, two items for a two-header structure). You can use a global computation to retrieve the report values or use a specific computation for each header item.

Example of a computation applied to two headers:

The computation mode of each table cell content can be either identical for all of the cells, either specific for each header item.



5.6.1 Data computation steps

The data computation is performed in three steps. The first two steps collect the data that will be the input of the method, the third one performs the computation with the given method.

These steps are activated or not according to the computation method chosen.

- Collection and global computation: data collection and computation are applied regardless of header items. When a header item is not explicitely described, only a global computation is possible.
- Collection and item-specific computation: data collection and computation are header item- specific. The item-specific computation is only possible when header items are explicitly described, in that case all of the items must have a computation, possibly the same one.

5.6.2 Data collection steps - Input and Provider

The data collection is built either directly by the specified input data, or by a provider that uses an input.

- **input data**

  Input data can be given by a report parameter or by a header (meaning the associated occurrences) or by an item.

  or

- **collected data**

  In that case the **Provider** checkbox must be selected. The **Provider** collects occurrences using:

  - a query

    The query can have 0 to n parameters. These parameters are the occurrences specified as input. Parameters refer to only one occurrence, but in case of collection, the query is executed n times and the result is aggregated.

or

- a MetaAssociationEnd (applied to occurrences specified as input).

The query includes at least one of the following parameters (or none if there is no input):

- **report parameter**: the provider input is one (or n) occurrence(s) of a type specified for the parameter.
- **header**: the provider input is the occurrence of type associated with the header. If the header is fed through a query or a MetaAssociationEnd, it is the type specified by the query target or by the MetaAssociationEnd. In other cases, it is the type of the parameter associated with the header. If there is no parameter associated with the header it is the type of the parameter associated with the dataset.
- **item**: the provider input is the occurrence associated with this item. If this item is based on a provider it is the type of occurrence in output of this provider, else the type is determined by the header.

The result of this step is a collection of data used as input by the computation step, see Computation step - methods p. 49.

5.6.3 Computation step - methods

The following computation methods are provided, they are performed on each cell and can take the following values:

- **Attribute Value**

  Gets the value of the attribute on the occurrence(s) defined as input.

- **Condition**

  Executes a MetaTest on the object instance defined as input

- **Method**

  Executes the computation code on the occurrence(s) defined as input.

  Delivers the value processed by the renderer type for the cell addressed by the n current elements (one element by header)

  The computation code includes key words and enables to filter the occurrence collection and/or to perform calculation on these occurrences.

  This code enables to:

  - use the object collection defined as input as a typical MegaCollection (browsing, count, etc.)
  - use calculation functions on an attribute of this collection: Average, Sum, Min, Max…
  - modify a cell color (when available with the renderer type).

The following output value types are available:

- object
- collection of objects
- numerical value
- boolean

- text
- date

The following display formats are available:

- stored currency (data entry)
- default currency
- specific currency
- %
- Image (of the object)

When the computation result is a numerical value, the set of occurrences that were used for computation is accessible through drill-drown. You can overload or define this collection in the computation text.

6 Report Template creation wizard

To create a report template, follow these steps:

Step	Action	See page
1.	Set up the report template (launch the Report Template creation wizard and define if needed the report template parameter types)	51
2.	Create a chapter	53
3.	Create the view	55
4.	(if needed) Create the report data computation	60

6.1 Setting up the report template

To set up the Report Template:

Report Template creation

To define the report template basic properties

1. From the **Utilities** Navigation window, right-click **Report Templates** and select **New > Report Template**.

   The **Basic Properties** wizard page appears.
2. Enter the Report Template **Name** (e.g.: Application) and eventually a comment.
3. Click **Next**.

   The **Parameters** definition wizard page appears.

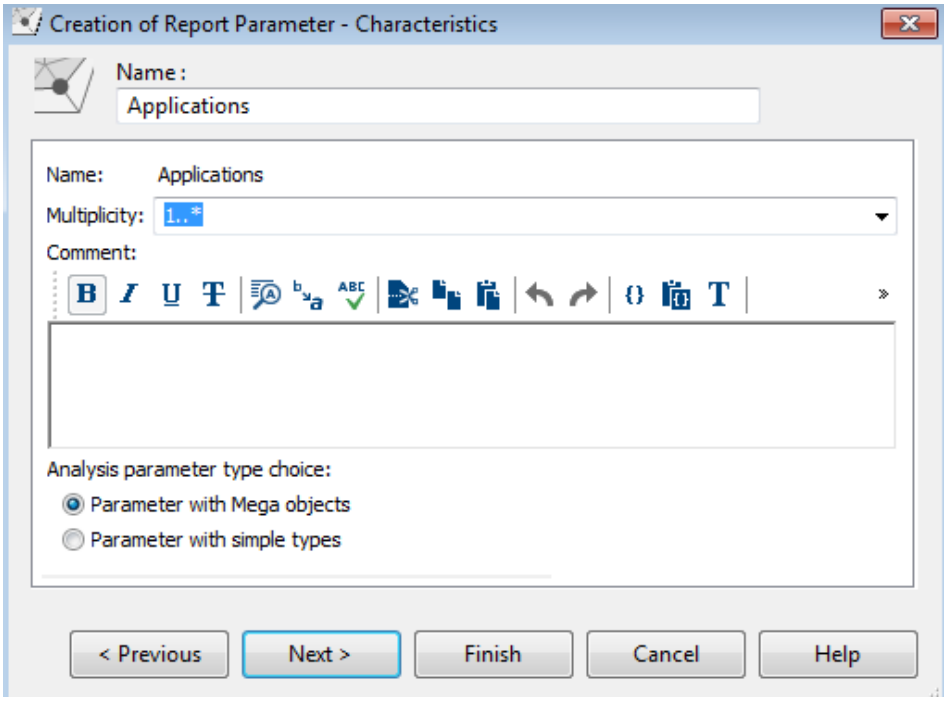
Report Template parameter creation

Parameter definition indicates the input data type the user must enter for the report calculation (e.g.: Applications).

Do not define any parameter if you want the report to be launched without requiring the user to select any input objects.

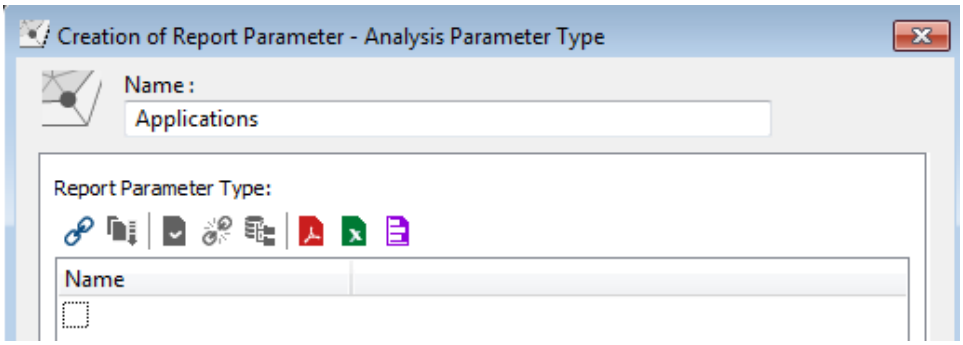
For each parameter define the available object type that can be instantiated for the report (parameter with Mega objects or with simple types)

4. (If needed) In the **Report Template Parameter** pane, click New  , and for each parameter define its required **Characteristics**:
   - the parameter **Name**.
   - its **Multiplicity**.
   - the object types that can be instantiated for the report



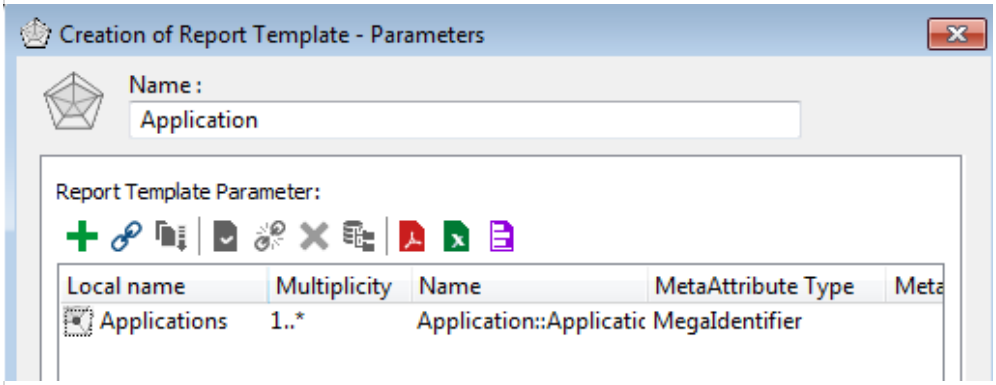
5. Click **Next**.

The **Analysis Parameter Type** wizard page appears.



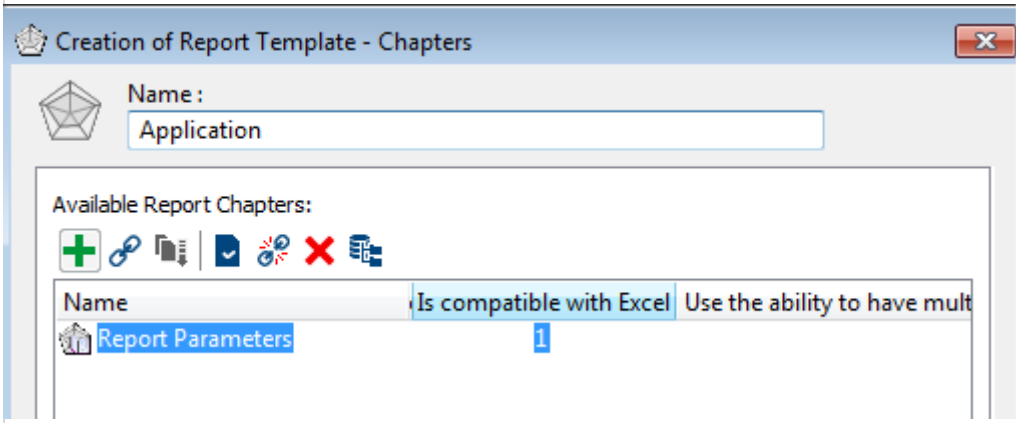
6. In the **Report Parameter Type** pane, click **Connect**  .
7. Select the report parameter type(s).
8. Click **Finish**.

The input data type the user must enter for the report calculation are defined.



9. Click **Next**.

The **Chapters** creation wizard appears, see Creating a chapter p. 53.



A default chapter (“Report Parameters”) is already created.

- ☞ To customize the report parameter display see
- ☞ Customizing Parameter Display p. 69.

10. Go to next step: Creating a chapter p. 53.

6.2 Creating a chapter

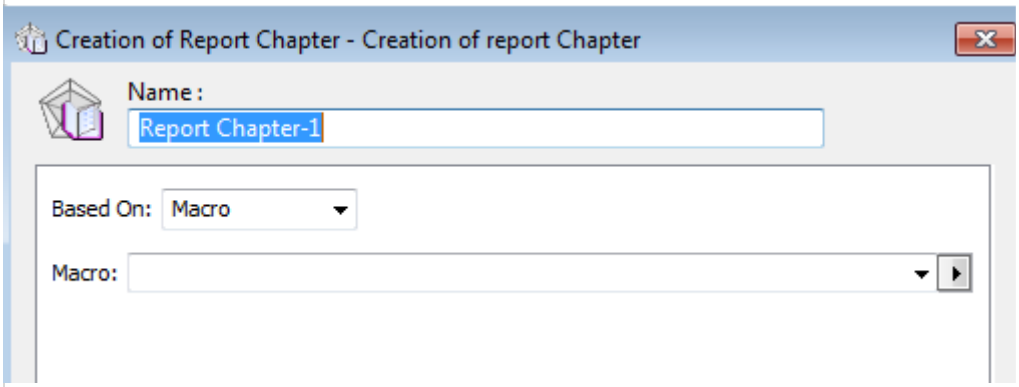
☞ You can also create a new chapter from an existing report template, from the Report Template properties (**Chapters** tab), see Accessing a report template properties p. 63.

To create a report chapter:

Chapter creation

1. From the **Chapters** creation wizard, in the **Available Report Chapters** pane, click **New**  .

   The **Creation of report Chapter** wizard page appears.



2. Enter the chapter **Name**.

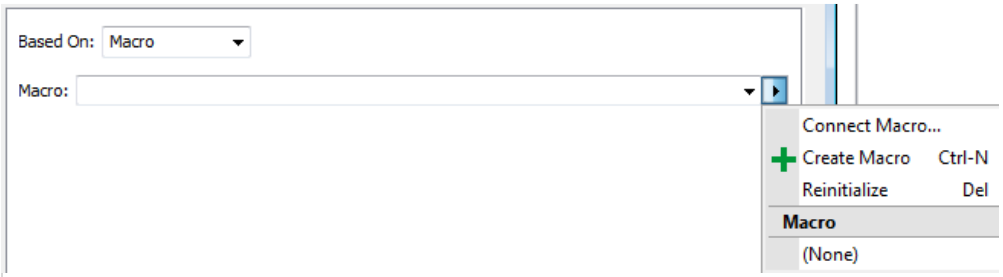
The Chapter creation wizard is based on either:

- a view, or
- a macro

3. In the **Based On** field, select the chapter creation type.

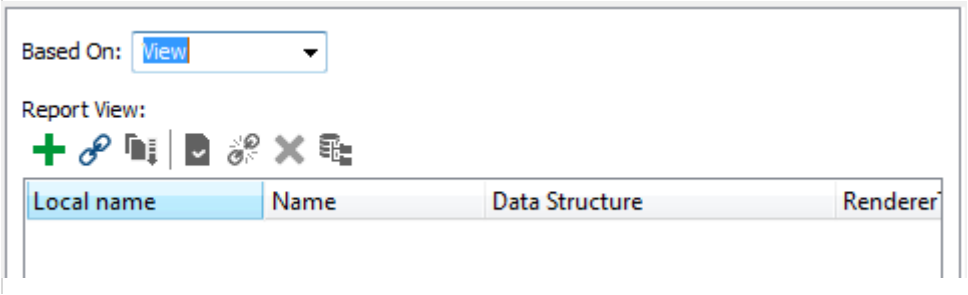
   When you select:

   - the macro-based chapter creation, the wizard enables to connect/create the macro and the chapter creation is finished. For more details on macros, see Defining report chapters with macros p. 74.



the view-based chapter creation, the **Report View** pane appears, see

- Creating a view p. 55.



6.3 Creating a view

The **Creation of View** wizard enables to create and configure the view:

- its renderer type.
- the data structure and its headers.

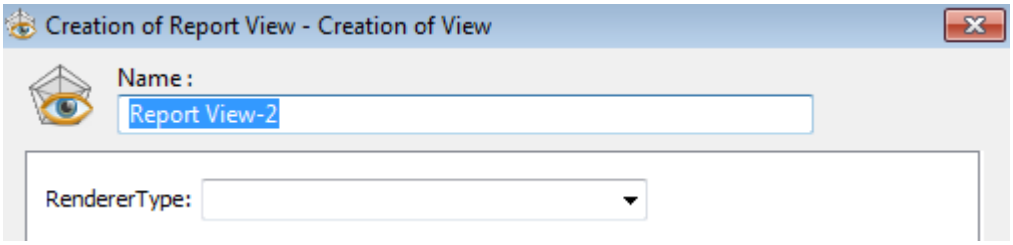
☞ You can also create a view from an existing report chapter, from the Report chapter properties (**Characteristics** tab), see Accessing a report chapter properties p. 64.

To create the view:

View creation

1. From the **Creation of report Chapter** wizard page, in the **Report view** pane, click **New**  .

   The **Creation of View** wizard page appears.



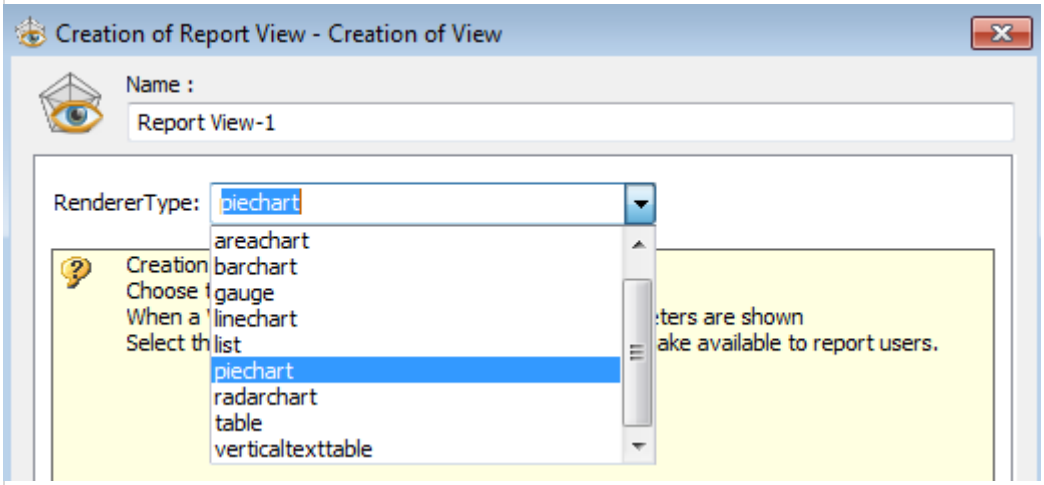
2. In the **Name** field, enter a name for the view.

Renderer Type choice

See Renderer types p. 44

3. (Optional) In the **RendererType** field, select the renderer type.

   ☞ If your view is intended to include other views leave the **RendererType** field empty and go to step 6 to create a grouped view.



4. Click **Next**.

   The **Report Headers** wizard appears.

5. Go to step 9 to define the Report headers.

Grouped views

To define the view,

6. (optional) In the **Number of columns** field, enter the number of columns to define how the grouped views are displayed.

which is a container of several views and define how these views are displayed with the use of columns (see

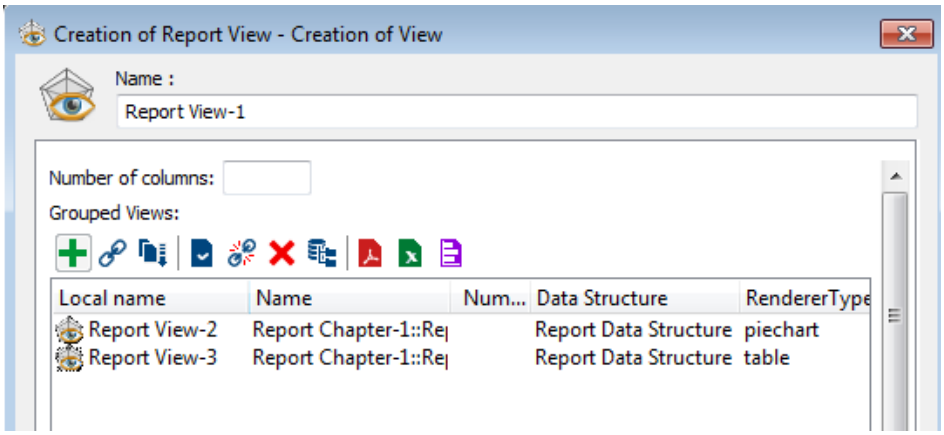
View p. 43).

☞ Leave the field empty to display the views in a single column.

7. In the **Grouped Views** pane, click **New** to create each included view.

   Configure each included view as detailed for a view (Name, Renderer Type, number of columns if it includes other views).

   In the following example, "Report View-1" view includes "Report view-2" and "Report view-3" views, which are displayed vertically on a single column.



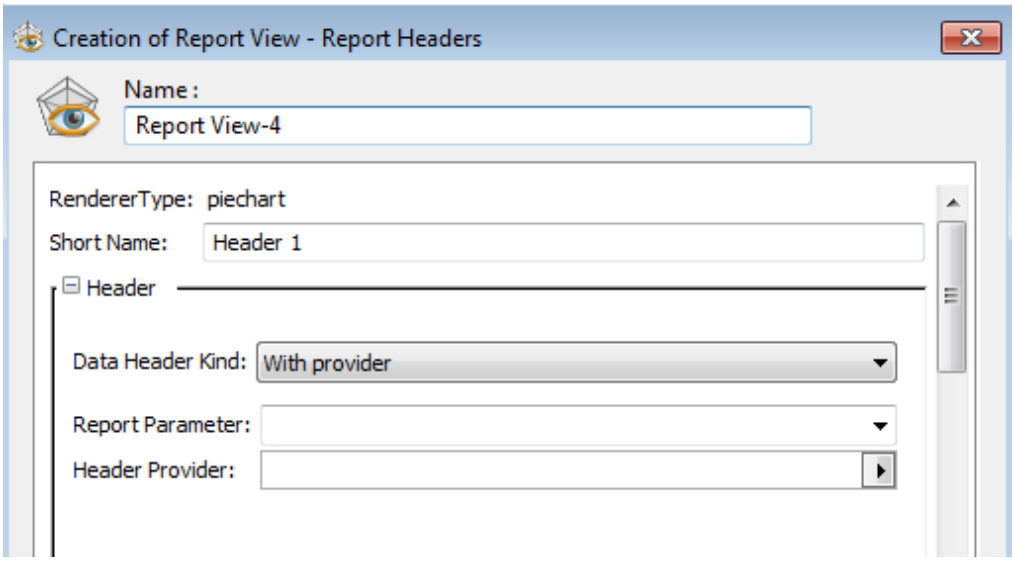
8. To modify the **Grouped Views** display go to Modifying the Grouped Views display p. 78.

Report Header definition

As many wizards are displayed as needed headers.

You can link a computation to header items for only one header item of the data structure.

9. In the **Report Headers** wizard, in the **Short Name** enter the header name.



10. In the **Header** pane, from the **Data Header Kind** field, select how the header is retreived:
    - **With provider**, see Getting header items with Provider p. 57.
    - **Explicit items**, see Getting header items with Explicit Items p. 58.
    - **Explicit items with computation**, see Getting header items with Explicit Items with computation p. 59.
11. Click **Finish**.

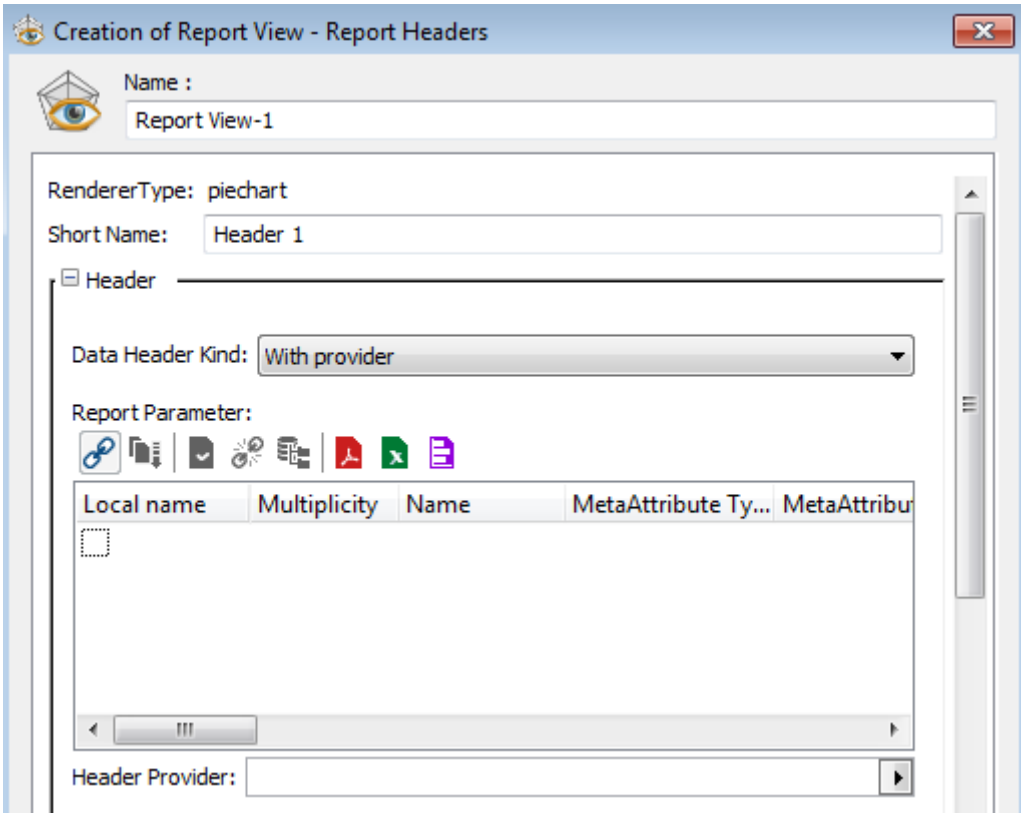
    The **Creation of DataSet Content Computation** wizard appears, go to Creating

the report data computation p. 60.

6.3.1 Getting header items with Provider

To get header items with provider:

1. In the **Header** pane, from the **Data Header Kind** field, select **With provider**.



2. (optional) From the **Report Parameter** pane, connect a parameter (you can connect several parameters) if the header item is based on (one of) the parameter defined as report template input and which is based on an occurrence (see Setting up the report template p. 51).
3. In the **Header Provider** field, if in step 2 the **Report parameter**:
   - is defined, the parameter type enables to define the list of queries, which have this parameter type or the list of MetaAssociationEnds, which start from this object type, select either:
     - **Connect a query** or
     - **Connect a MetaAssociationEnd**

     Example: if in the **Report Parameter** field you selected « Portfolio », the "APM – Get applications from portfolio" query enables to retrieve these header items, which are these portfolio applications.
   - is not defined, the provider is either:
     - a tabulated value MetaAttribute (or a TaggedValue)

       Select **Connect a MetaAttribute** or **Connect a TaggedValue**.

       or
     - a query without parameter

       Select **Connect a query/New query**.

Example: if the provider is the “Audit plan category” attribute, the header items are made up of the different values of the audit plan categories.

	Compliance	Efficiency	...	...
Audit Plan 2010	10	10	13	7
Audit Plan 2011	10	15	8	5
Audit Plan 2012	11	18	7	4

4. Click **Finish**.

   The **Creation of DataSet Content Computation** wizard appears.

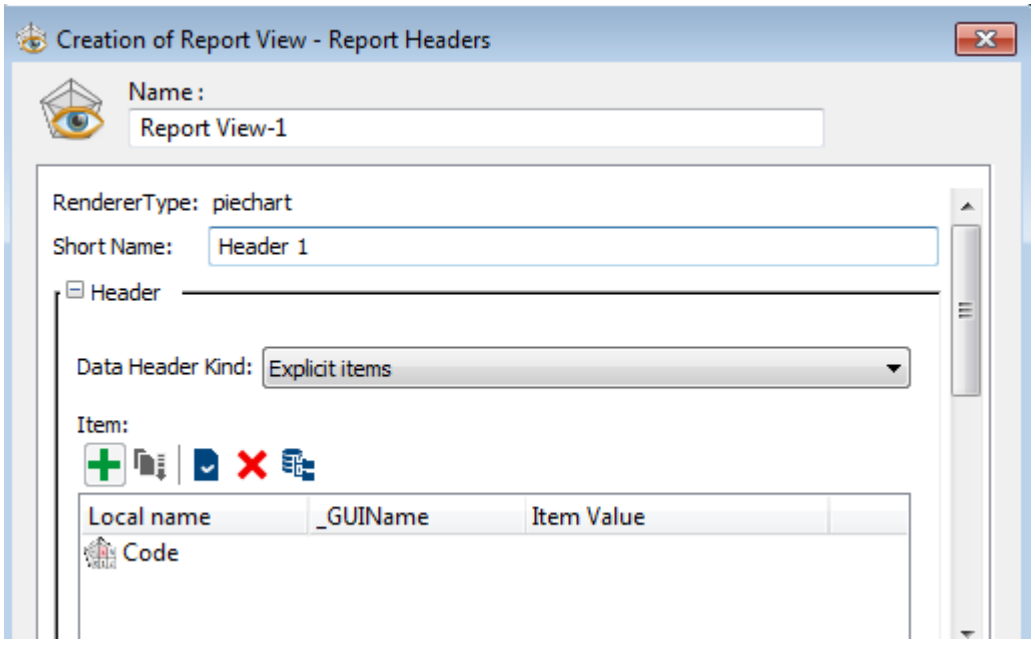
5. Go to Creating the report data computation p. 60.

6.3.2 Getting header items with Explicit Items

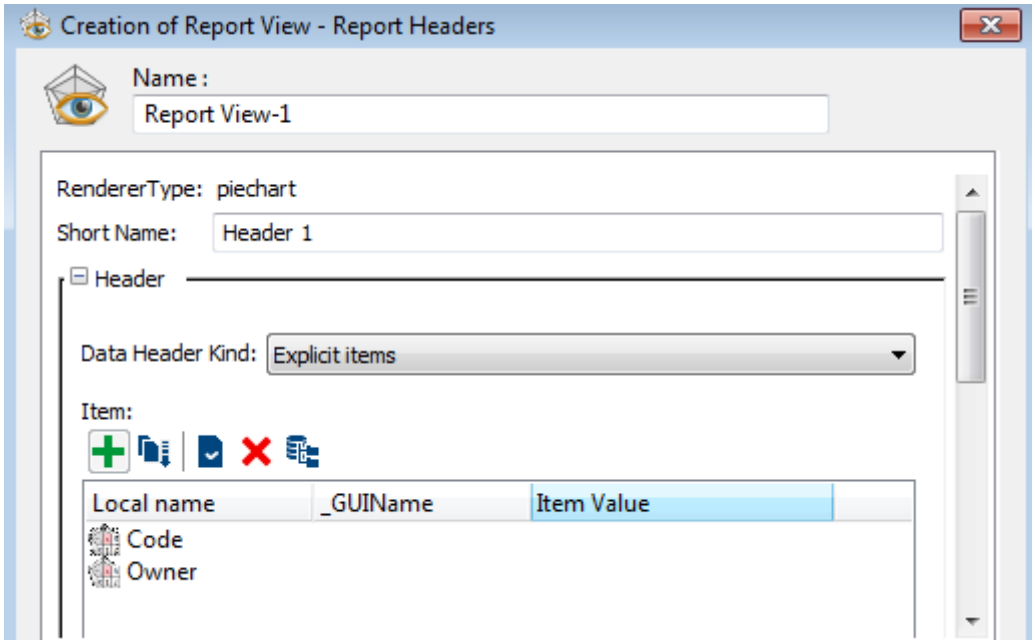
To get header items with explicit items:

1. In the **Header** pane, from the **Data Header Kind** field, select **Explicit items**.
2. In the **Item** pane, click **New**  .

   The data header item is created.
3. In **Local name**, enter the data header name.

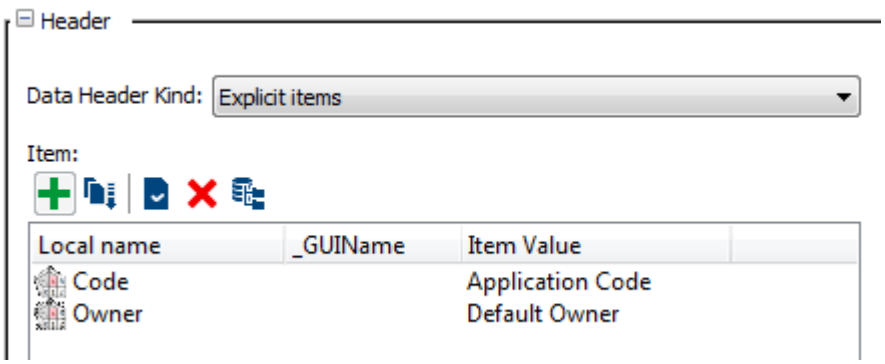


4. (optional) In **_GUIName** enter the data header name to be displayed in the GUI (axis, legend) else by default the **Item Value** name is displayed.



5. Click in the **Item Value** field, and select **Connect** to enter the data header item value
6. In the **Query** window, select the data on which is based the item value. This value can be:
   - an occurrence property (MetaAttribute, TaggedValue).

     `Example: the "Code" header value is defined through the "code" attribute of the "Application" MetaClass.`
   - a MetaModel data (MetaAttributeValue, MetaClass, MetaAssociationEnd)



7. Click **Finish**.

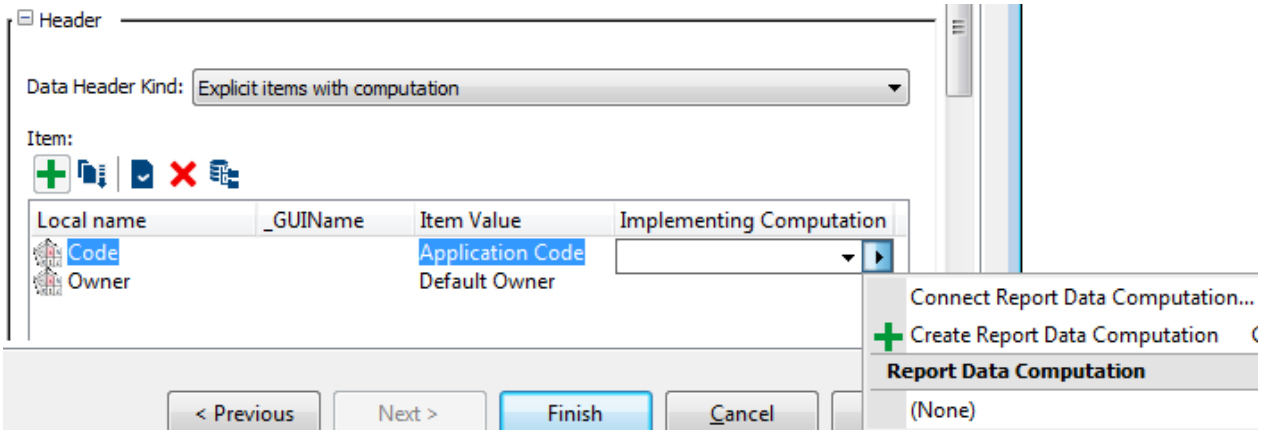
   The **Creation of Report Data Computation** wizard appears.
8. Go to Creating the report data computation p. 60.

6.3.3 Getting header items with Explicit Items with computation

When the header is detailed and the computation depends on these header items, you have to define a computation for each of these header items (you can reuse a computation associated to another item of this header).

To get header items with explicit items with computation:

1. Proceed as detailed for Getting header items with Explicit Items p. 58
2. Click in the **Implementing Computation** field, and select **Create Report Data Computation**.



The **Creation of Report Data Computation** wizard appears.

3. Go to Creating the report data computation p. 60.

6.4 Creating the report data computation

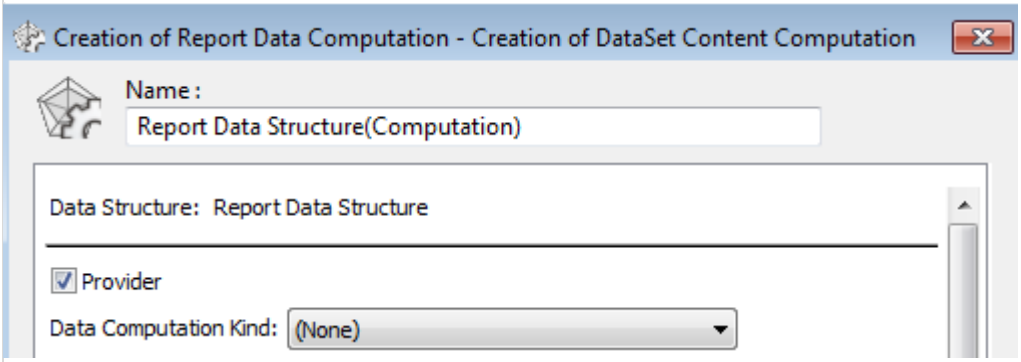
The dataset Computation enables the computation of the values that will be provided to the renderer.

To create the report data computation:

To create the dataset content computation

This wizard defines the input data processing and/or computation, which enable to retrieve the values stored in the data structure and provide them to the renderer.

1. (Optional) From the **Creation of DataSet Computation** wizard, in the **Name** field, enter a name for your computation.



The object instances used for the computation can be those of a given:

- report parameter
- header
- item

else, require a new search in the repository (**Provider** selected)

When a parameter is associated with the provider, this wizard enables to select/create a query (or a MetaAssociationEnd).

2. Select **Provider** if data are collected through a provider, else leave it clear if data are input data.

3. In **Data Computation Kind**, select the computation type:
   - **Attribute Value**, which gets the value of the attribute defined as input
   - **Condition**, which executes a MetaTest on the object instance defined as input. The Metatest is available for single header structures or when the computation is specific to a header item.
   - **Method**, which executes the computation code
4. Click **Next**.

   The **DataSet Content Input** wizard page appears.

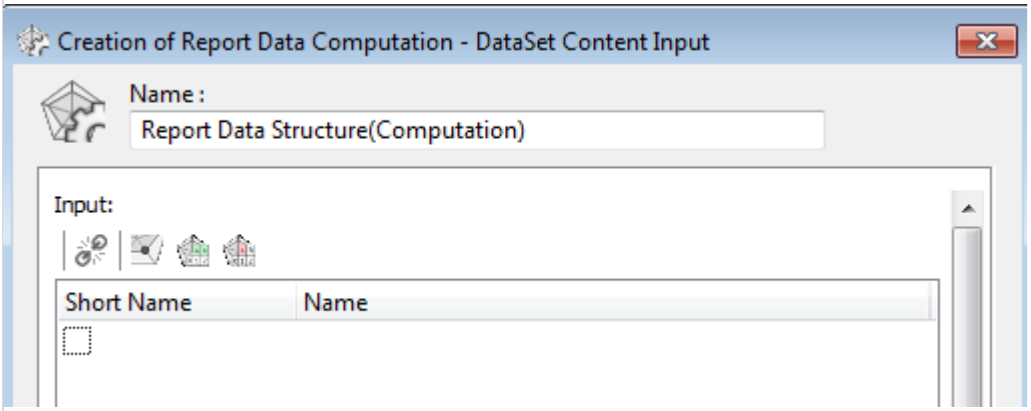
To define the dataset content input for computation

The **Input** defines which object instances will be used as input by the provider and/or method executed to compute the values provided to the renderer type.

Input can be based on:

- report parameters, based on occurrences
- data structure headers
- items (e.g.: other items of the same header)
- remains empty.

5. (Optional) From the **DataSet Content Input** wizard page, modify the computation **Name**.



6. (optional) In the **Input** pane, click:
   - to connect a report parameter, and/or
   - to connect a report header, and/or
   - to connect an item
7. Click **Next**.

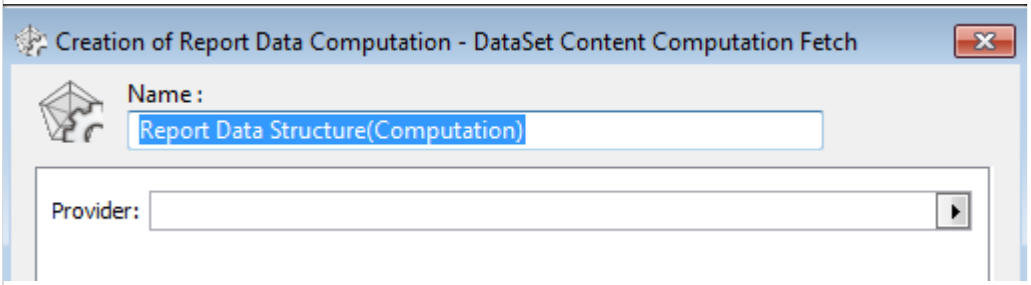
   The **DataSet Content Computation Provider** wizard page appears.

To define the dataset content provider

The Provider can be either a:

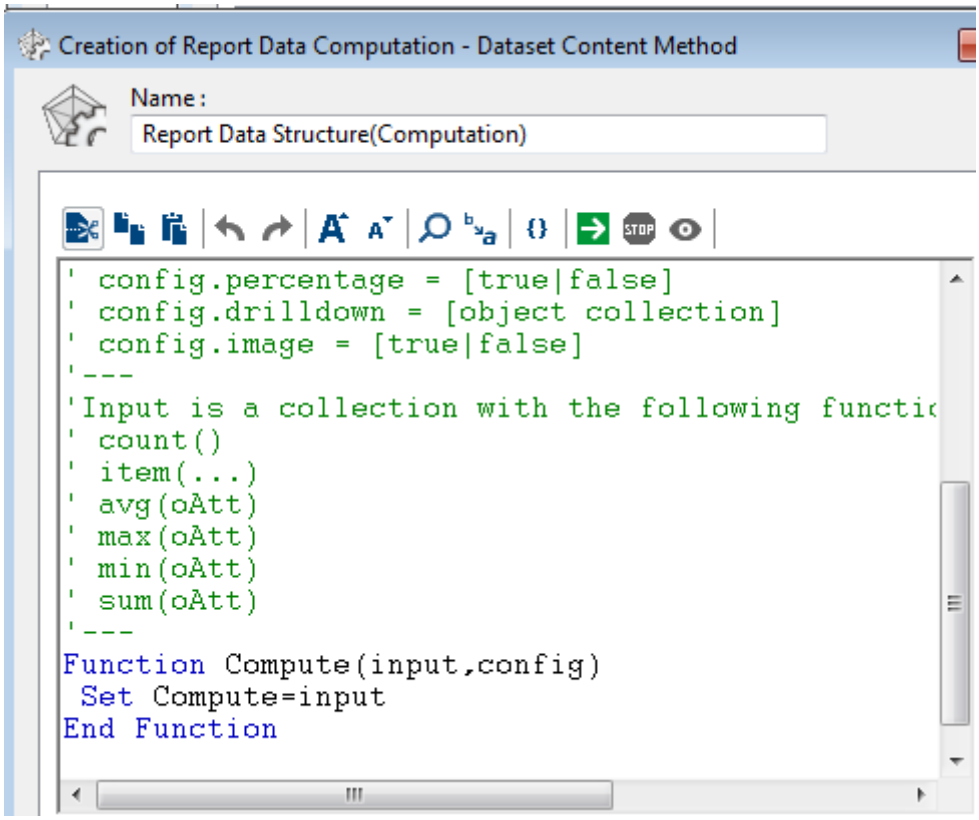
- MetaAssociationEnd (case where input is a header item)
- query, which includes at least one of the following parameters (or none if there is no input):
  - report parameter
  - header
  - item

8. The **DataSet Content Computation Provider** wizard page, in the **Provider** field, select the provider type:
   - **Connect a Query**
   - **Connect a MetaAssociationEnd**



9. Click **Finish**.
10. The **DataSet Content Method** wizard page appears.

To define the dataset content method



11. Click **Finish**.

    The report view is created and defined. You can create other reports.

12. Click **OK**.

    The report chapter is created and defined. You can create other chapters.

13. Click **Finish**.

7 Customizing a report template

7.1 Report template properties

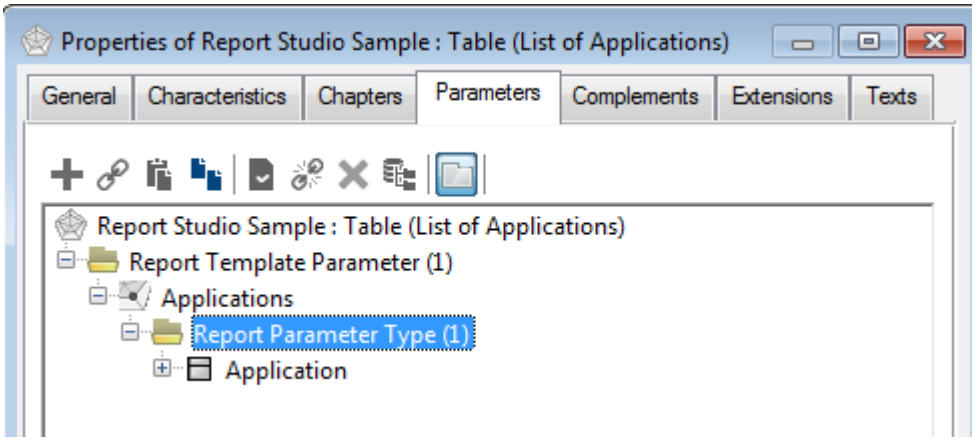
7.1.1 Accessing a report template properties

To access the report template properties:

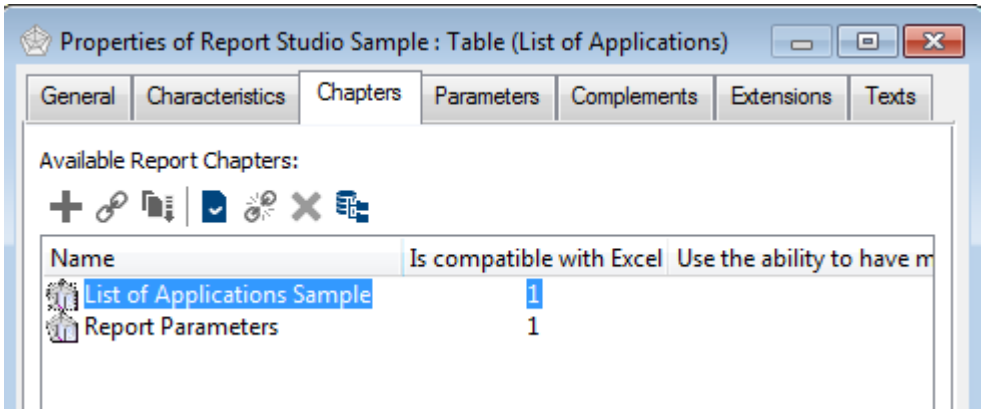
1. From the **Utilities** Navigation window, expand **Report Templates** folder.
2. Right-click the report template and select **Properties**.

   Dedicated tabs enable access to:

   - input **Parameters**



   - report **Chapters**



   You can:

   - add new parameters in the report template, see Defining new parameters in a report template p. 67.
   - customize parameter display, see Customizing Parameter Display p. 69.
   - define report chapters, see Defining report chapters with macros p. 74.

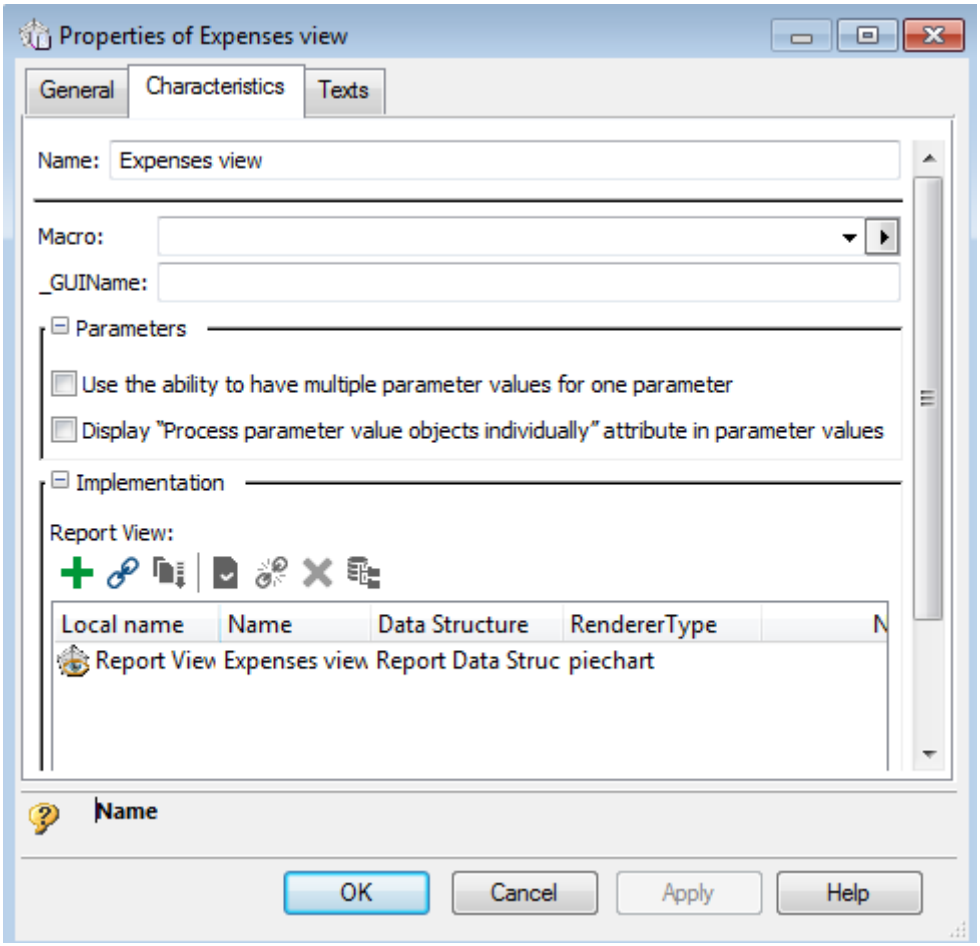
7.1.2 Accessing a report chapter properties

To access the report chapter properties:

1. From the report template properties, select **Chapter** tab.

   ☞ Else, from the **Utilities** Navigation window, expand **Report Templates** folder and the report template to which the report chapter is linked.

2. Right-click the report chapter and select **Properties**.



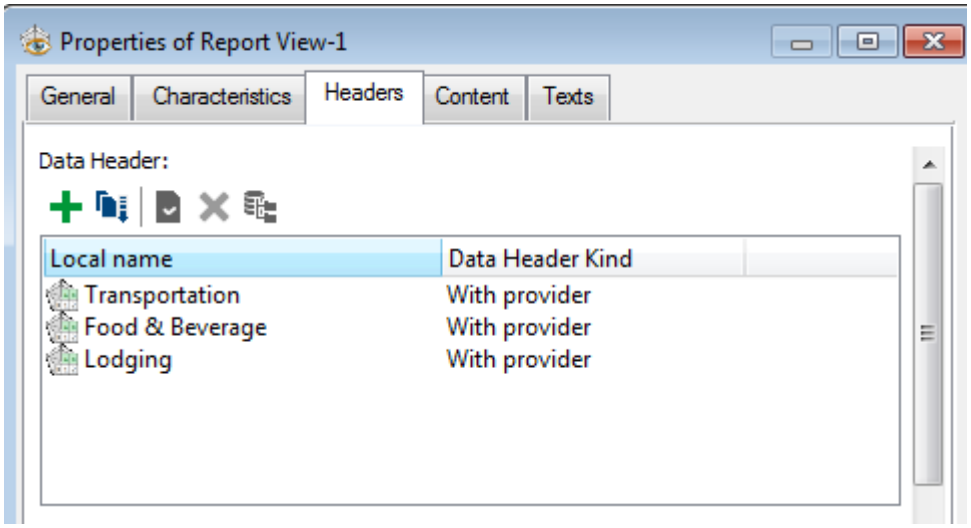
You can define the report chapter characteristics, see:

- Processing parameter value objects individually or collectively p. 71
- Authorizing multiple values in a parameter p. 72
- Defining report chapters with macros p. 74
- Accessing the View properties p. 64
- Managing report snaphots p. 7.1.465

7.1.3 Accessing the View properties

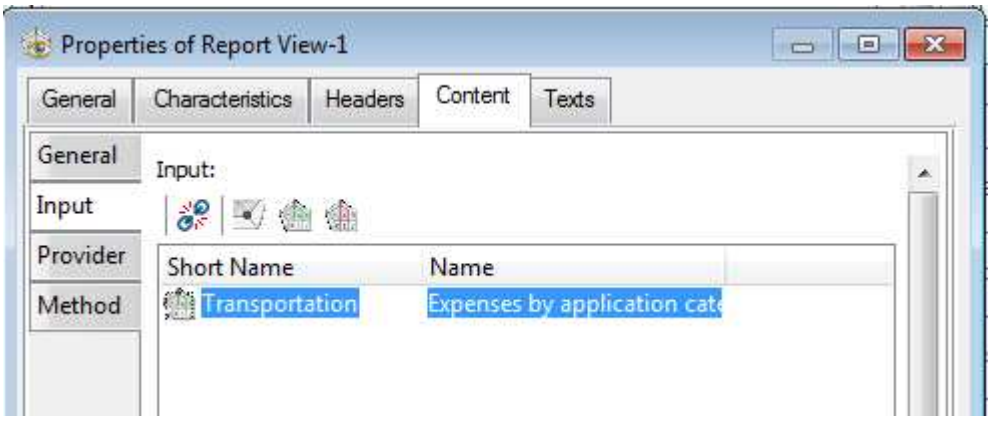
To access the View properties:

1. From the report chapter properties, select **Characteristics** tab.
2. In the **Implementation** pane, right-click the view and select **Properties**.



The View properties give access to:

- the report headers (**Headers** tab)
- the report value collect mode (**Content** tab)
  - input data (**Input** sub tab)
  - data collection mode (**Provider** sub tab)
  - report value computation mode (**Method** sub tab)



7.1.4 Managing report snaphots

Report snapshot creation is available for Java Reports or Report Studio.

Report snapshot creation is useful for a fast report chapter display and is also used for export operations.

When the report snapshot creation is not activated the report chapter display is fully recalculated at each refresh, which can be time consuming.

For each report chapter, you can activate or deactivate the report snapshot creation and define its type.

To manage a report snapshot:

1. From the report chapter properties, select **Characteristics** tab.
2. In the **Snapshot** pane, from the **Report snapshots** drop down list select:
   - "Yes" to activate the report snaphot creation.

- “By Format” to activate a report snapshot creation. In this case a report snapshot for each format is created (html, pdf, excel).
- “No” to deactivate the report snapshot creation.

3. From the **Snapshot Type** drop down list select:
   - “Snapshot by user” (by default) for example when data reading access (confidentiality) or data writing access are used.
   - “Snapshot by profile” for example when data is profile dependent (i.e.: a portfolio of a profile).
   - “Global Snapshot”, when the report chapter is available for all the users.
4. Click **OK**.

7.2 Defining new parameters in a report template

Report template parameters enable to define required elements to build a report. For each parameter you must specify the object type that can be instantiated for the report as well as its multiplicity.

7.2.1 Creating a parameter

To create a parameter:

1. Right-click the report template and select **New > Report Parameter**.

   The Report parameter creation wizard appears with its owner name.

2. Click **Next**.
3. Enter the parameter **Name** and a comment if required.
4. Specify its **Multiplicity**.

   📖 The **Multiplicity** of a report parameter enables definition of the number of values that can be associated with the parameter when creating a report. If multiplicity is 1 or 0..1, only one parameter value can be connected.

5. Select the report parameter type:
   - **Parameter with MEGA objects** enables connection of MetaClasses that can define the parameter. For example, for a parameter named "Application Architecture", you can connect an applications folder, an application, a query returning applications.
   - **Parameter with simple types** enables connection of "Tagged Values".

   Parameter types correspond to report values. Assigned MetaClasses inherit the "System report value" abstract MetaClass.

   ☞ For details on abstract MetaClasses, see *HOPEX Studio - MEGA Studio* Technical Article, “Abstract Metamodel” section.

7.2.2 Parameter with simple types

A parameter with simple types defines the types that can take values in the report (e.g.: boolean, date, string). MEGA proposes by default a set of simple types, represented by tagged values.

📖 Tagged values play the same role as MetaAttributes but do not belong to the MEGA metamodel.

To create a parameter that filters display of certain objects in the report, use an optional boolean:

1. In the parameter creation window, name the parameter (for example "Display Owner").
2. Click **Next**.
3. Select multiplicity "0..1" and select **Parameter with simple types**.
4. Click **Next**.
5. In the **Tagged value** pane, click **Connect** .
6. In the dialog box that appears, find and select "Boolean Report Value" among the tagged values available for a report.
7. Click **Connect**.

   The tagged value appears in the parameter creation dialog box.

8. Click **Finish**.

Results in generated report

To create a report from the report template you have created:

1. Right-click the report template and select **New > Report**.

   The report creation wizard opens.
2. Enter the report name and click **Next**.
3. Among the report parameters, the **Display Owner** parameter appears in the form of a check box.
4. Define the objects of parameters and click **Finish** to launch the report.

7.3 Customizing Parameter Display

7.3.1 Grouping parameters

You can group the display of report parameters. A group of parameters corresponds to a MetaAttributeValue that is defined on the "Report Parameter" MetaClass and then connected to the parameters of report templates concerned.

> Consider a report template "Org-Unit Characteristics", which contains three parameters: "Status", "Street Number" and "City". "Street Number" can include one or no value, with a multiplicity of "0..1". The "City" parameter should display a mandatory value, with a multiplicity of "1".

For consistency and visibility, we want parameters "Street Number" and "City" to be displayed in an "Address" group.

To create the "Address" group:

1. Open the properties of the "Report Parameter" MetaClass.
2. From the **MetaAttribute** tab, open the properties of the “Displayed in Group” MetaAttribute
3. Select the **Characteristics** tab, then the **Standard** sub tab.
4. In the **MetaAttributeValues for enumerated MetaAttribute Only** pane, create the MetaAttribute Value "Address".
5. Specify an **Internal Value** as well as values in each data language.
6. Exit MEGA and compile the metamodel to take account of modifications.

To connect the "Address" group to the parameters of the "Org-Unit Characteristics" report template:

1. In the "Org-Unit Characteristics" report template, select the "Street Number" parameter and open its properties.
2. Select the **Characteristics** tab.
3. In the **Displayed in Group** field, select the "Address" MetaAttributeValue.
4. Carry out the same procedure for the "City" parameter.

Note that if both parameters have the same order number, the "City" parameter appears in the report before the "Street Number" parameter, the multiplicity "1" being positioned before the multiplicity "0..1". For more details, see Defining object display order in generated reports p. 70.

To position "Street Number" before "City"", modify the MEGA order number of the "Street Number" parameter:

1. In the "Street Number" properties, select the **General tab** and then the **Administration** sub tab.
2. In the **Order** field, enter an order number lower than that of the "City" parameter.
3. Click **OK**.

Results in generated report

To create a report from the "Org-Unit Characteristics" report template:

1. Right-click the "Org-Unit Characteristics" report template and select **New > Report**.

   The report creation wizard opens.

2. Indicate the report name and click **Next**.

   "Street Number" and "City" parameters are displayed under the "Address" group.

3. Specify values for mandatory parameters and click **Finish**.

7.3.2 Defining object display order in generated reports

Parameters without groups

For parameters not containing groups, display order depends on:

1. **MEGA** order number: objects of a list have an order number taking value "9999" by default. You can modify this value for each object (from the object properties, **General > Administration** tabs).
2. **Multiplicity**: for parameters with the same MEGA order number, display order depends on multiplicity:
   1. Multiplicity 1
   2. Multiplicity 0..1
   3. Multiplicity 1..*
   4. Multiplicity * and NONE (the two are equivalent)
3. **Alphabetical order**: for the same multiplicity, it is alphabetical order that is taken into account.

Parameters with groups

The parameter group display order depends on group alphabetical order.

In each group, parameters are displayed in MEGA order.

If parameters of the group have the same order number, their display order depends on their multiplicity:

1. Multiplicity 1
2. Multiplicity 0..1
3. Multiplicity 1..*
4. Multiplicity * and NONE (both are equivalent)

For the same parameter multiplicity, it is alphabetical order that is taken into account.

Parameters with and without group

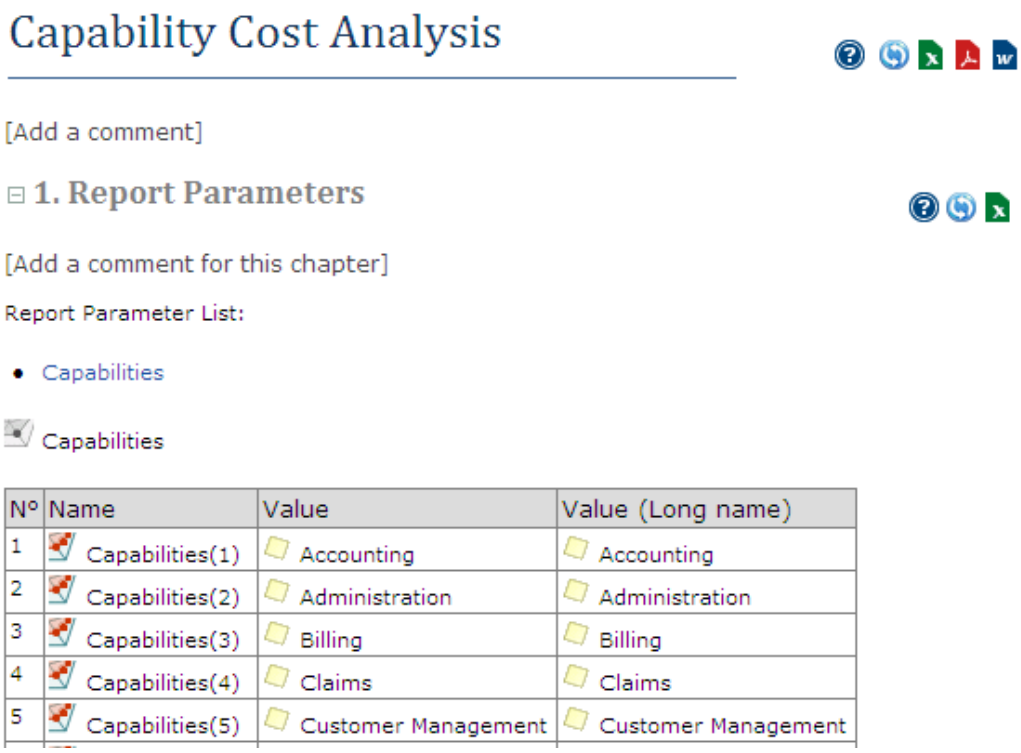
Parameters not in groups appear in first position. They are followed by groups.

7.4 Processing parameter value objects individually or collectively

7.4.1 Objects processed individually

Report parameters can include several objects. By default, each object is processed individually in report chapters.

For example, a capability cost analysis report contains a "Capabilities" parameter in which capabilities to be analyzed are defined. Each capability corresponds to an entry in the report.



Capability Cost Analysis

[Add a comment]

1. Report Parameters

[Add a comment for this chapter]

Report Parameter List:

- Capabilities

Capabilities

N°	Name	Value	Value (Long name)
1	Capabilities(1)	Accounting	Accounting
2	Capabilities(2)	Administration	Administration
3	Capabilities(3)	Billing	Billing
4	Capabilities(4)	Claims	Claims
5	Capabilities(5)	Customer Management	Customer Management

7.4.2 Objects processed collectively

When creating a report, selected objects can be grouped in the parameter value so that the report chapter processes them collectively.

For example, below, all capabilities are grouped under the same value.

Capabilities

N°	Name	Value	Value (Long name)
1	Capabilities	Accounting	Accounting
		Administration	Administration
		Billing	Billing
		Claims	Claims
		Customer Management	Customer Management
		Human Resources	Human Resources

To group objects at parameter definition, you must deactivate the **Process Values Individually** attribute. This attribute is active but hidden by default.

To display this attribute in the report creation wizard:

1. Open properties of the report template, from which you want to create reports.
2. From the **Chapters** tab, open the properties of the report chapter that will process the parameter values.
3. In its **Characteristics** tab, select the **Display "Process parameter value objects individually" attribute in parameter values definition** option.
4. Click **OK**.

   When you create a report, the attribute appears under each parameter. Clear it when you do not want parameter values to be processed individually.

7.5 Authorizing multiple values in a parameter

By default, a parameter contains an input value, which can contain one or several MEGA objects.

When defining a parameter, you can create several values, each assembling a set of MEGA objects. The report chapter takes each value (group of objects) as an entry point.

> For example, by default an application functional analysis report contains the "Functional Scope" parameter, which groups one or several city plans under the same value.
>
> On this "Functional Scope" parameter you can create two values: "Functional Scope 1" which includes a set of city planning areas, and "Functional Scope 2" which contains another set of city planning areas.

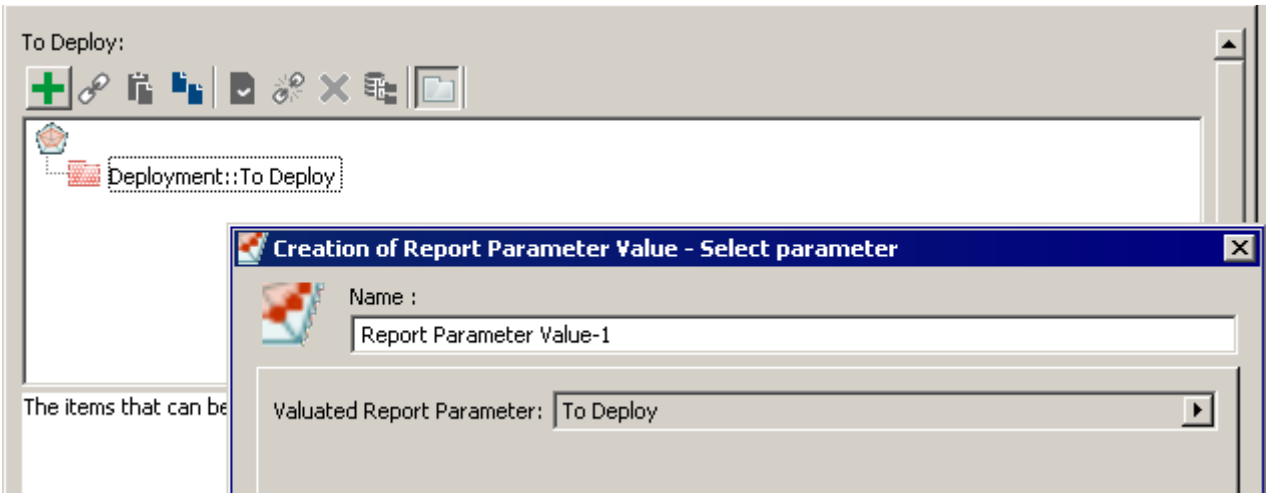
This functionality is specific to each report chapter and is not used by them all.

To authorize multiple values in parameter definition:

1. Open properties of the report chapter that will process the parameter values.
2. In the **Characteristics** tab, select “Use the ability to have multiple parameter values for one parameter”.

7.5.1 Result in generated report

When you create a report, the parameter value definition window displays a different interface.



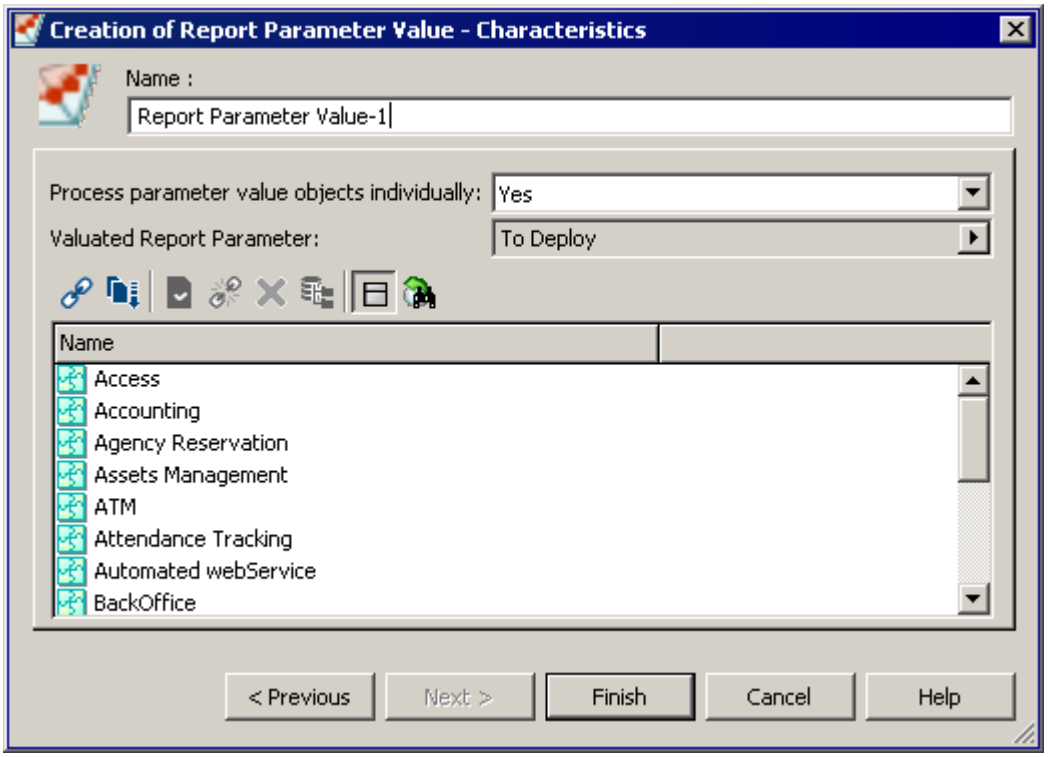
To create a parameter value:

1. Select the parameter, and click **Create**.
2. In the **Name** field, enter parameter value name.
3. Click **Next**.

   A query dialog box opens.
4. Select the value objects.
5. Click **OK**.

   The objects appear in the value creation window.

   ☞ You can decide to process objects individually or collectively in the chapter that processes the value, see Processing parameter value objects individually or collectively p. 71.



6. Click **Finish**.

You can create other values in the same way.

7. Click **Next** then **Finish** to launch the report.

7.6 Defining report chapters with macros

You can define report chapters with a macro, connected to the report template through a report generator.

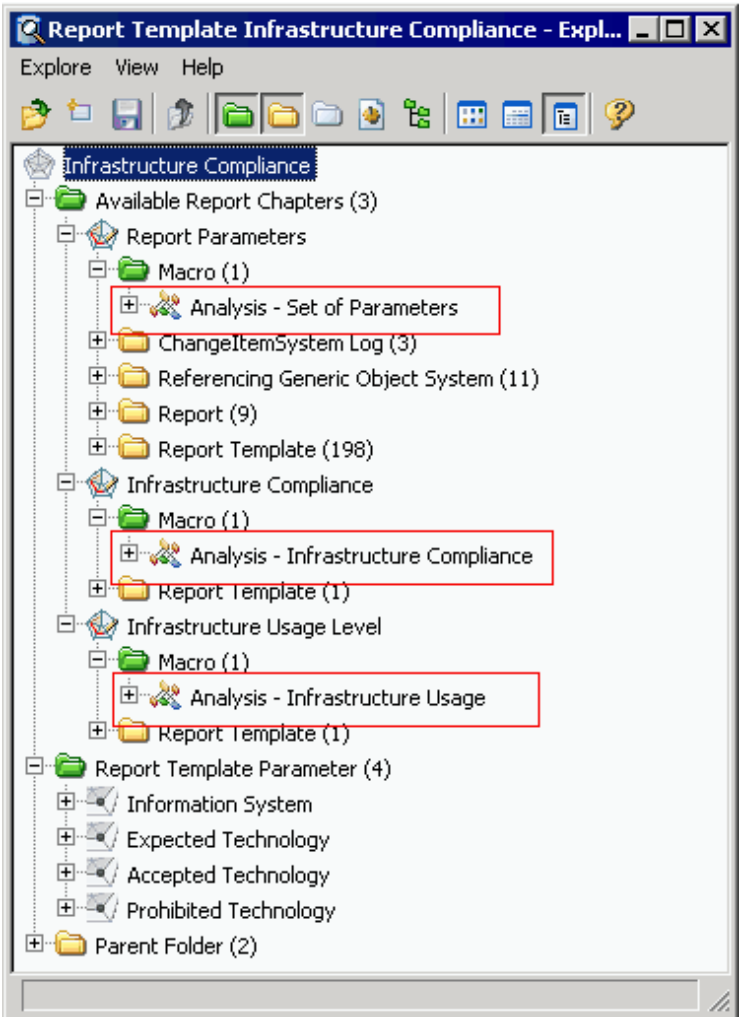
Example of the "Infrastructure Compliance" report template. For example this report template can show the usage level of a technology set, or compliance of an information sub-system set with a technical infrastructure group.

This report template generates three report chapters:

- Report Parameters
- Infrastructure Compliance
- Infrastructure Usage Level

These three report chapters are defined respectively in the analysis template by the following macros:

- Analysis - Set of parameters
- Analysis - Infrastructure Compliance
- Analysis - Infrastructure Usage



To build your report chapters, you can use an existing macro on condition that parameters are compatible. **MEGA** provides in particular the macro" Analysis – Set of Parameters" which describes report template parameters and the way in which they are interpreted by the report chapter.

☞ Macros have a comment available in the form of a system note.

You can also create new report chapter macros.

7.6.1 Creating a report chapter macro

To create a report chapter macro use a report generator. You can create macros either in VB Script or Java format.

To create a report chapter macro:

1. In MEGA menu bar, select **Tools > Options**.
2. In the **Options** tree, select **Repository** and set the **Metamodel Access** option to "Expert".
3. Right-click the report template to which the macro relates and select **New > Report Chapter**.
4. In the **Name** field, enter the report chapter name.
5. In the **Based on** field, select “Macro”.
6. In the **Macro** field, click the right-oriented arrow and **Create Macro**.

   The macro creation wizard appears.
7. Select **Create a Java Macro** and click **Next**.
8. In the **Setup JAVA Macro**:
   a. In the **Class Name** field, enter the class defined in Eclipse when developing the Java macro.
   b. In the **Package Name** field, enter the macro path (for example "com.mega.modeling.analysis.reports" for those supplied by MEGA).
9. Click **Finish**.

   The macro appears in the navigator, under the **Report Chapter** associated with the report template.

JAVA macros

To create a Java macro, see the following Technical Articles:

- *Writing Java Report Chapters*
- *All about starting with APIs*
- *API annex*, which includes the *API - JavaDoc*

  A Java API help is also delivered with MEGA in JavaDoc format.

To read this Help:

1. In **<MEGA Installation>\java**, expand **doc** file.
2. Unzip "mj-api.doc.zip" and "mj_anls.doc.zip in the shared repository.
3. Open "index.html" page.

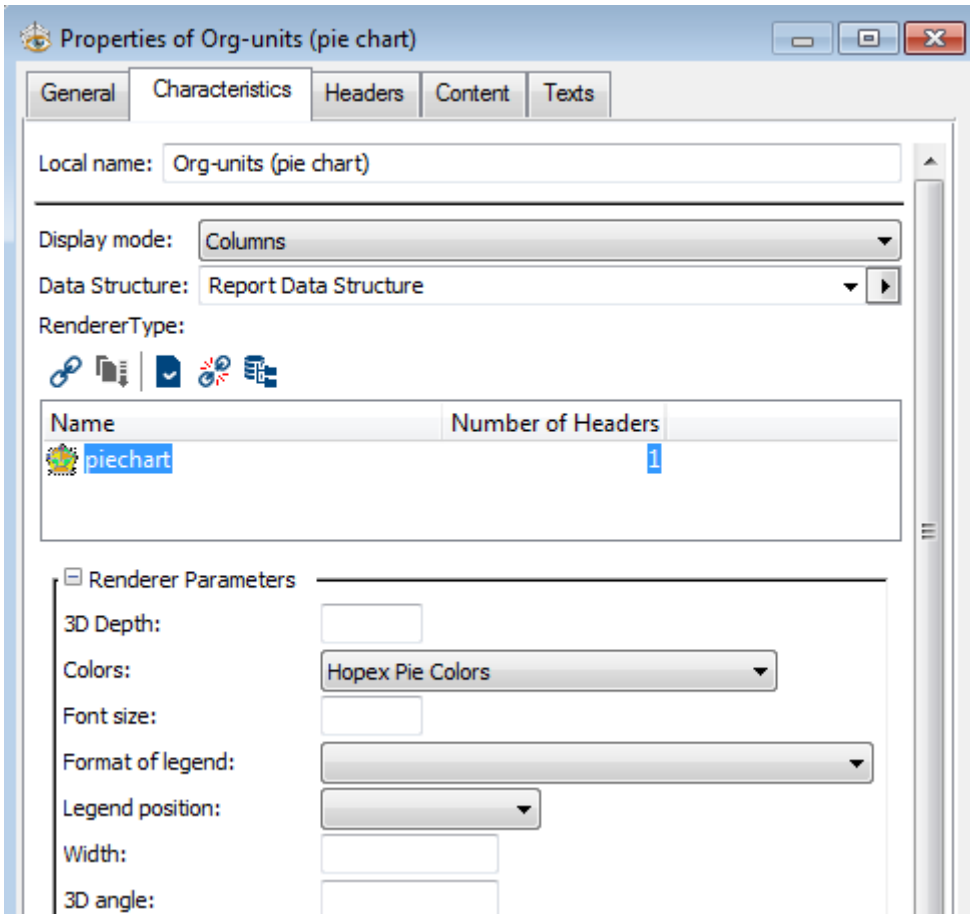
7.7 Customizing the renderer display

7.7.1 Renderer view

Each of the renderers used in a view can be customized. Some of these parameters have:

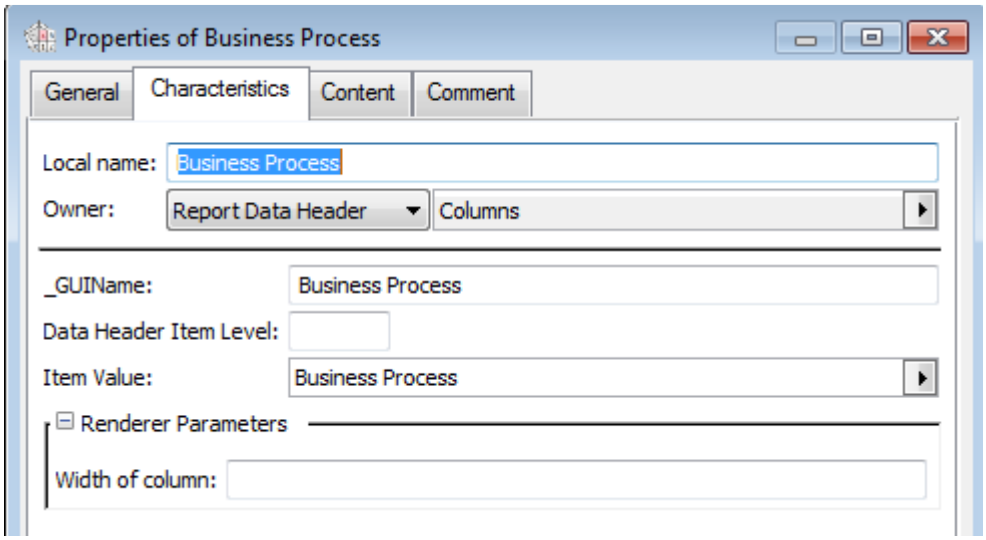
- a global effect on the whole view.

  Example: the **Renderer Parameters** in the renderer properties.



- a local effect on part of the view.

  Example: the **Renderer Parameters** in a header item properties.



Report Studio enables to define the display and formatting parameters of the Renderer that the user can access.

To do so, renderer parameters can be associated with some chapter parameters and to a specific view. You can define the parameters available for the user. These parameters can be either global chapter parameters (report parameters are only linked to the chapter) or parameters that apply only to specific views (report parameters are linked to the chapter and the view).

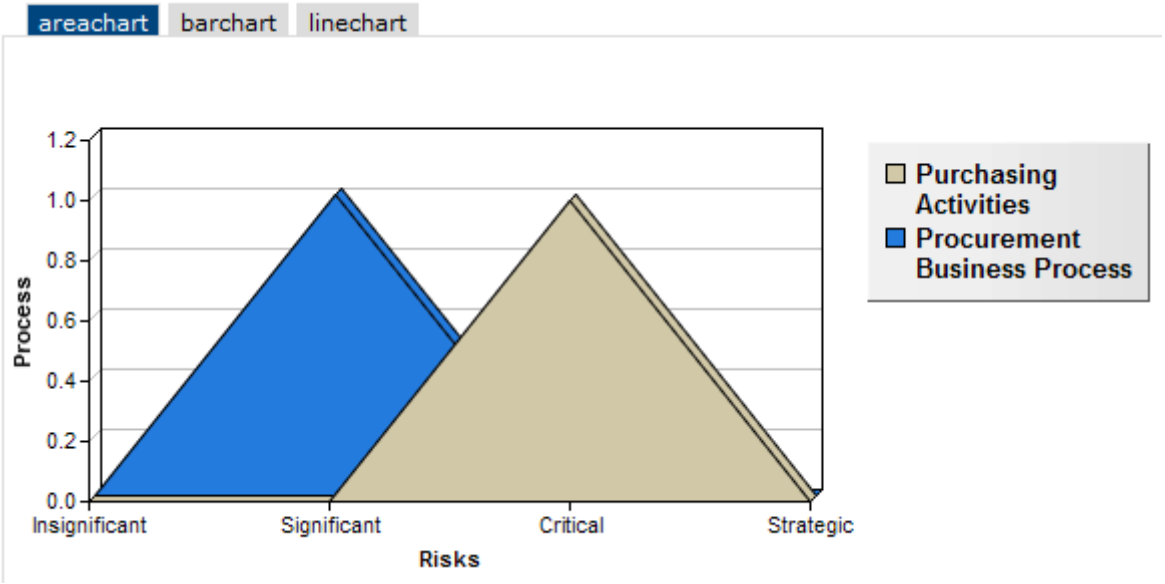
Note: This impacts the parameter access from the report displayed to the end user.

7.7.2 Multi-renderer views

A view can display several renderers from the same data structure.

☞ This is possible only for renderers with the same header numbers.

The following example shows three displays of the same data: **areachart** tab, **barchart** tab, and **linechart** tab.



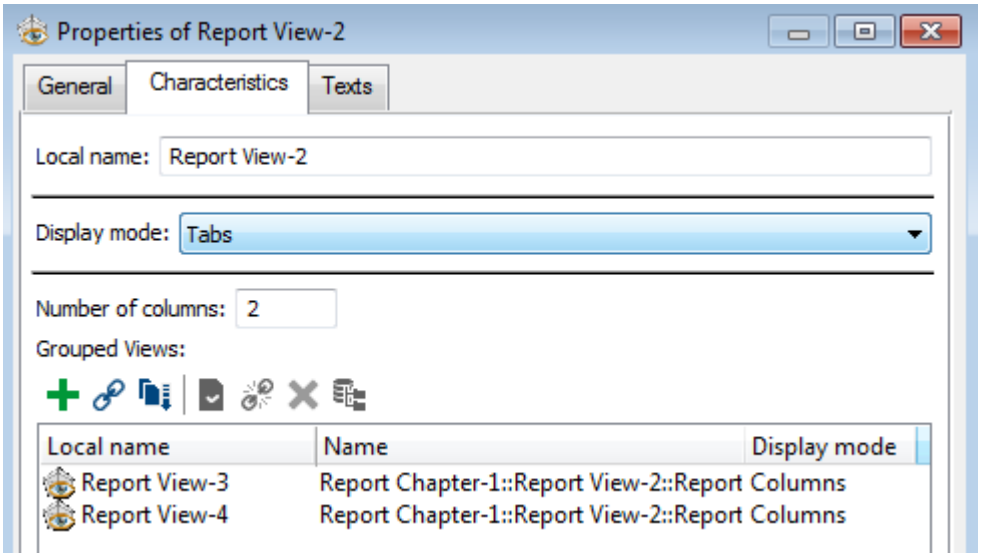
7.7.3 Modifying the Grouped Views display (in tabs)

By default the **Grouped Views** are displayed in columns. You can modify this column display for a tab display, especially when views are too large to fit in the same row.

To modify the Grouped Views display:

1. Access the **Report View** properties.
2. From the **Characteristics** tab, in the **Display mode** field, select “Tabs”.

   Note: The **Number of columns** parameter is then not considered)



7.8 Duplicating a report template

The recommended way to create a new report template similar to an existing one is to duplicate a template.

To duplicate a template:

1. **Prerequisite:** In the **Repository** options, check that the “Definition of path of MetaAssociation” option is set to “Compatibility up to MEGA 2009”.
2. Right-click the report template and select **Duplicate**.

7.9 Compatibility

7.9.1 Mixt reports

A report template can include a set of chapters based on the different implementing technologies.

```
Example: a view-based chapter then a java code-based chapter, etc.
```

8 Report modeling rules

When creating new report templates you need to follow the modeling rules that ensure the use and consistency of the reports built using these report templates.

Reminder:

A rule applies to a MEGA repository object and defines a check on this object.

A regulation enables classification of rules according to a specific context or domain.

> ☞ For details on the rules, see *MEGA Common Features* user guide, “Documentation” part, "Generating Documentation with MEGA" chapter, "Checking objects" section.

Report-specific rules are listed in the “Report Regulation” regulation.

To access the Report Regulation rules:

3. From Mega menu bar, select View > Navigation Windows > MetaStudio.
4. Expand the **Repository Consistency** folder.
5. Expand the **Modeling Regulation** folder.
6. Explore the **Report Regulation**.

You can define an active regulation, which is always visible on the reports and report templates you create.

You can also temporarily apply a regulation on a report to check its consistency.

10 Customizing a report color palette

With HOPEX Studio you can create your own report color palette. For this purpose you have to duplicate an existing palette and modify it.

You can create a new palette for another library.

To create a color palette:

1. Connect to MEGA with **MEGA Customizer** profile/business role.
2. From Mega menu bar, select **View > Navigation Windows > Utilities**.
3. Expand **HOPEX Palette** folder.
4. Right-click the palette that will provide the basis for your palette and select **Duplicate**.
5. (If needed) In the **Library** pane, select the option for duplicating the palette in another library and from the drop-down list select the target library.
6. Select the duplication name format for your palette (Prefix or Suffix).
7. Click **OK**.

   The color palette is duplicated. You can customize it.
8. Access the color palette properties.
9. From the **Characteristics** tab, in the **Palette Type**, select “Report Palette”.
10. From the **Color set** tab, modify the color as needed.
11. Click **OK**.

11 Glossary

Report View: a view is chapter area displaying a set of values (Data Structure) in a graphical representation or not (renderer type).

Report Data Structure: structure of data including the values organized and processed by the renderer types.

Report Header: the data structure includes headers that correspond to the header tables or to the legend of the displayed chart.

Data Header Kind attribute defines how the header items are retrieved.

Report Header Item: a header is described through items of different types that need to be described or specifically processed.

Report Data Computation: enables the computation of the values that will be provided to the renderer.

Data Computation Kind attribute defines the computation mode of these values.

Report Data Provider: enables to collect the data to be processed (e.g.: query)

Renderer Type: defines the different display and formatting available for the dataset included in the reports (e.g.: table, bar chart, pie chart, area chart, etc.).

Writing Java report renderers

Java report renderers allow the extension of the report engine, offering more rendering possibilities to Java report chapters. This technical article presents how such renderers are modeled and written.

INTRODUCTION AND PREREQUISITES

This technical article presents how renderers are modeled and implemented in Java.

It requires a configured Java development environment and that you be familiar with Java report chapter implementation and the data structure which reports generate.

If not, please refer to the following technical articles:

- MEGA Plug-ins with Java
- Writing Java report chapters
- Java report data structure

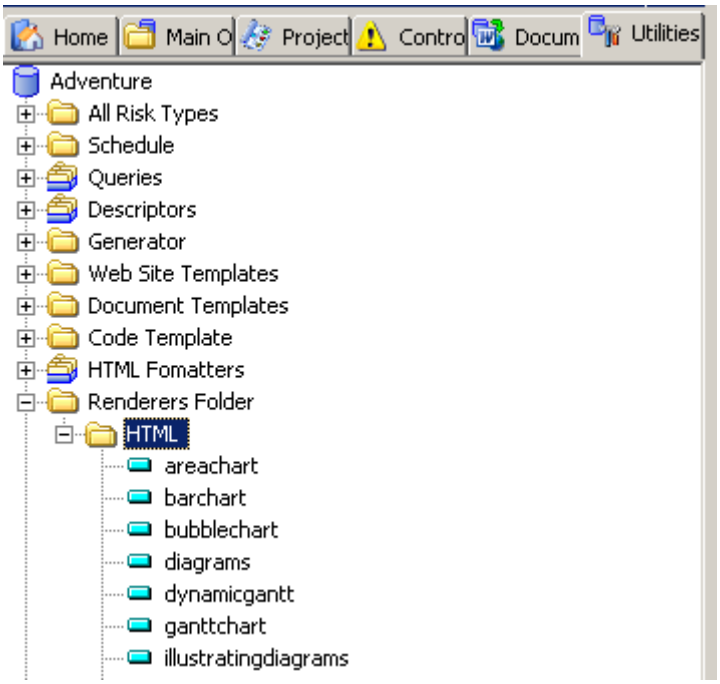
For each rendering format which should be handled (HTML, PDF, etc.), there is one renderer implementation.

> In Mega 2009 SP5, HTML and PDF renderers are taken into account. However, RTF is generated from HTML.
> In Mega 2009 SP4, only HTML renderers are taken into account.

RENDERER MODELING

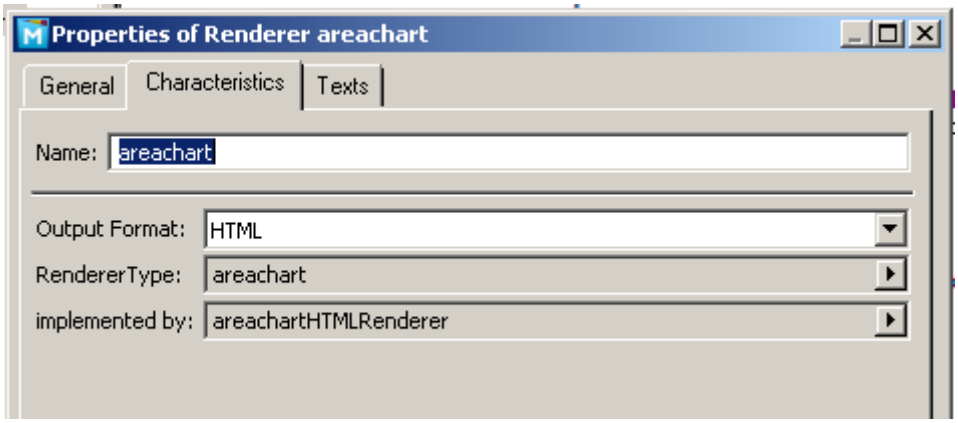
Renderers are modelled in Mega and called through an object identifier.

They are available in the Renderers Folder of the Utilities navigation window:



Each renderer is specific to a format and type e.g. HTML areachart renderer.

This is specified in renderer properties:



- Output Format: format handled by this renderer.

- Renderer Type: type of rendering produced. The renderer type is used in the analysis report to specify which rendering to apply to the dataset.
- Implemented by: the implementation macro, in Java (see next chapter for its implementation).

The call of the renderer is handled by the analysis engine depending on generation format (HTML, PDF, etc.).

RENDERER IMPLEMENTATION

Interface to implement

Renderers must implement the `com.mega.modeling.analysis.ViewRenderer` interface. This requires the implementation of a single Generate function.

This results in the following structure:

```
public class MyHTMLRenderer implements ViewRenderer {

  @Override

  public boolean Generate(final Object document, final ReportContent
reportContent, final Item i, final MegaCOMObject megaContext, final
Object userData, final MegaRoot root) {

      …

  }

}
```

The aim of the function is to append the document object with the rendering of Item i, using the data contained in the ReportContent object.

In this situation, the Item is usually the View object created by the analysis report to link a dataset to this renderer.

> In the case of HTML, the document object is a StringBuffer.
> In the case of PDF, the document object is an aspose.pdf.Pdf.

A Mega Context object, Mega Root and an optional userData are also made available.

This function produces a boolean that indicates:

- True: capable of rendering in this context with the data supplied, and has done so by appending the document object.
- False: not capable of rendering and has not appended the document object. The analysis engine should try to find another suitable renderer.

Implementation example

This example uses ChartDirector 4.1, a graph library provided with Mega capable of rendering many different types of charts.

> Further documentation on ChartDirector can be obtained from ASE at:
> http://download2.advsofteng.com/v4/cdjavadoc_html_v4.zip

HTML Renderer Code

```
import ChartDirector.BarLayer;
import ChartDirector.Chart;
import ChartDirector.LegendBox;
import ChartDirector.XYChart;
import com.mega.modeling.analysis.AnalysisRenderingToolbox;
import com.mega.modeling.analysis.ViewRenderer;
import com.mega.modeling.analysis.content.Dataset;
import com.mega.modeling.analysis.content.Dimension;
import com.mega.modeling.analysis.content.Item;
import com.mega.modeling.analysis.content.ReportContent;
import com.mega.modeling.analysis.content.Value;
import com.mega.modeling.analysis.content.View;
import com.mega.modeling.api.MegaCOMObject;
import com.mega.modeling.api.MegaRoot;


/**
 * Renderer for bar chart in HTML
 * Accepted dimensionalities: 2
 * @author NLE
 */
public class barchartHTMLRenderer implements ViewRenderer {
  @Override
  public boolean Generate(final Object document, final ReportContent
reportContent, final Item i, final MegaCOMObject megaContext, final Object
userData, final MegaRoot root) {
```

```
    if (i instanceof View) {
      final Dataset d = reportContent.getDataset(((View) i).getDatasetId());
      if (d.getDimensionCount() == 2) {
        AnalysisRenderingToolbox.setChartDirectorLicense(root);

        // The data for the chart
        final Dimension dim1 = d.getDimension(0);
        final Dimension dim2 = d.getDimension(1);

        // The labels & data for the chart
        final int labelCount = dim1.getItemCount();
        final int areaCount = dim2.getItemCount();

        final String[] labels = new String[labelCount];
        for (int ii = 0; ii < labelCount; ii++) {
          labels[ii] = AnalysisRenderingToolbox.getItemText(dim1.getItem(ii),
root);
        }

        // Create a XYChart object of size 400 x 240 pixels
        final XYChart c = new XYChart(600, 260);

        // Add a title to the y-axis

c.yAxis().setTitle(AnalysisRenderingToolbox.getCodeTemplate(dim2.getCodeTemplate
ID(), root));

        // Add a title to the x-axis

c.xAxis().setTitle(AnalysisRenderingToolbox.getCodeTemplate(dim1.getCodeTemplate
ID(), root));

        // Set the x axis labels
```

```
c.xAxis().setLabels(labels);

// Set the plot area and size.
c.setPlotArea(40, 30, 340, 190);

// Add a legend box at top right corner using 10 pts Arial Bold
// font. Set the background to silver, with 1 pixel 3D border effect.
final LegendBox b = c.addLegend(570, 30, true, "Arial Bold", 10);
b.setAlignment(Chart.TopRight);
b.setBackground(Chart.silverColor(), Chart.Transparent, 1);

//colors
String sColors = ((View) i).getParameter("colors");
String[] tColors = null;
if ((sColors != null) && (sColors.length() > 1)) {
  tColors = sColors.split(",");
}

// Add a multi-bar layer with data sets and 3 pixels 3D depth
final BarLayer layer = c.addBarLayer2(Chart.Side, 3);
for (int j = 0; j < areaCount; j++) {
  final double[] data = new double[labelCount];
  for (int ii = 0; ii < labelCount; ii++) {
    final Item curItem = d.getItem((ii + 1) + "," + (j + 1));
    if (curItem instanceof Value) {
      data[ii] = (Double) ((Value) curItem).getValue();
    } else {
      data[ii] = 0.0;
    }
  }
  if (tColors != null) {
```

```
            layer.addDataSet(data, (int) Long.parseLong(tColors[j], 16),
AnalysisRenderingToolbox.getItemText(dim2.getItem(j), root));

          } else {

            layer.addDataSet(data, AnalysisRenderingToolbox.getHexaColor(j),
AnalysisRenderingToolbox.getItemText(dim2.getItem(j), root));

          }

        }


        // Get filename

        final String imgurl =
root.currentEnvironment().toolkit().getStringFromID(root.currentEnvironment().to
olkit().generateID()) + ".jpg";

        // Generate chart


c.makeChart(AnalysisRenderingToolbox.getGenerationFolderWrite(megaContext, root)
+ imgurl);

        // append html

        AnalysisRenderingToolbox.getShowHideStart(root, (StringBuffer) document,
(View) i, d, megaContext);

        ((StringBuffer) document).append("<p><img src=\"" +
AnalysisRenderingToolbox.getGenerationFolderRead(megaContext, root, imgurl) +
"\"/></p>");

        AnalysisRenderingToolbox.getShowHideEnd((StringBuffer) document, (View)
i);

        return true;

      }

      return false;

    }

    return false;

  }

}
```

PDF Renderer Code

```
import java.io.File;

import ChartDirector.BarLayer;
```

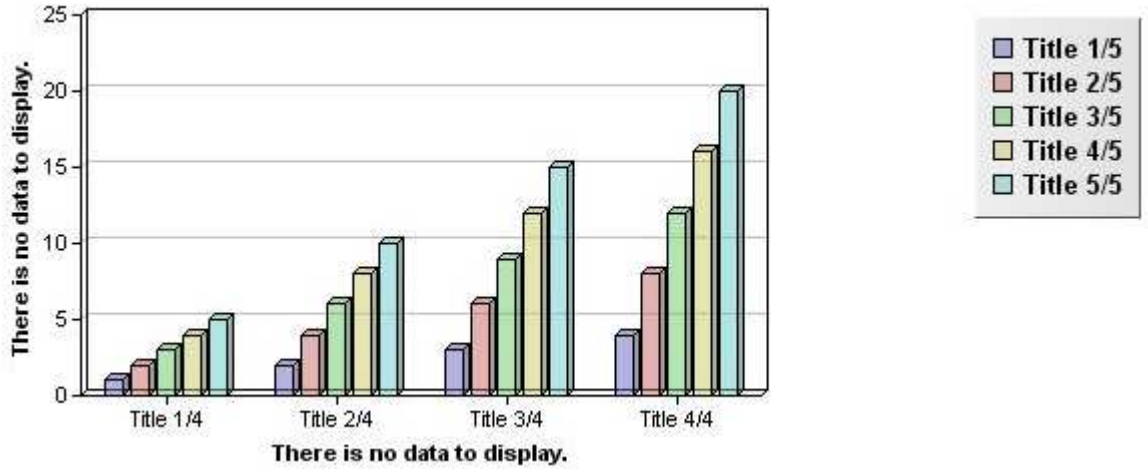
```
import ChartDirector.Chart;
import ChartDirector.LegendBox;
import ChartDirector.XYChart;
import aspose.pdf.Image;
import aspose.pdf.Pdf;
import aspose.pdf.Section;
import com.mega.modeling.analysis.AnalysisRenderingToolbox;
import com.mega.modeling.analysis.ViewRenderer;
import com.mega.modeling.analysis.content.Dataset;
import com.mega.modeling.analysis.content.Dimension;
import com.mega.modeling.analysis.content.Item;
import com.mega.modeling.analysis.content.ReportContent;
import com.mega.modeling.analysis.content.Value;
import com.mega.modeling.analysis.content.View;
import com.mega.modeling.api.MegaCOMObject;
import com.mega.modeling.api.MegaRoot;


/**
 * Renderer for bar chart in PDF
 * Accepted dimensionalities: 2
 * @author NLE
 */
public class barchartPDFRenderer implements ViewRenderer {
  @Override
  public boolean Generate(final Object document, final ReportContent
reportContent, final Item i, final MegaCOMObject megaContext, final Object
userData, final MegaRoot root) {
    if (i instanceof View) {
      final Dataset d = reportContent.getDataset(((View) i).getDatasetId());
      if (d.getDimensionCount() == 2) {
        AnalysisRenderingToolbox.setChartDirectorLicense(root);
```

```
        // The data for the chart
        final Dimension dim1 = d.getDimension(0);
        final Dimension dim2 = d.getDimension(1);

        // The labels & data for the chart
        final int labelCount = dim1.getItemCount();
        final int areaCount = dim2.getItemCount();

        final String[] labels = new String[labelCount];
        for (int ii = 0; ii < labelCount; ii++) {
          labels[ii] = AnalysisRenderingToolbox.getItemText(dim1.getItem(ii),
root);
        }

        // Create a XYChart object of size 400 x 240 pixels
        final XYChart c = new XYChart(600, 260);

        // Add a title to the y-axis

c.yAxis().setTitle(AnalysisRenderingToolbox.getCodeTemplate(dim2.getCodeTemplate
ID(), root));

        // Add a title to the x-axis

c.xAxis().setTitle(AnalysisRenderingToolbox.getCodeTemplate(dim1.getCodeTemplate
ID(), root));

        // Set the x axis labels
        c.xAxis().setLabels(labels);

        // Set the plot area and size.
        c.setPlotArea(40, 30, 340, 190);
```

```
        // Add a legend box at top right corner using 10 pts Arial Bold
        // font. Set the background to silver, with 1 pixel 3D border effect.
        final LegendBox b = c.addLegend(570, 30, true, "Arial Bold", 10);
        b.setAlignment(Chart.TopRight);
        b.setBackground(Chart.silverColor(), Chart.Transparent, 1);

        //colors
        String sColors = ((View) i).getParameter("colors");
        String[] tColors = null;
        if ((sColors != null) && (sColors.length() > 1)) {
          tColors = sColors.split(",");
        }

        // Add a multi-bar layer with data sets and 3 pixels 3D depth
        final BarLayer layer = c.addBarLayer2(Chart.Side, 3);
        for (int j = 0; j < areaCount; j++) {
          final double[] data = new double[labelCount];
          for (int ii = 0; ii < labelCount; ii++) {
            final Item curItem = d.getItem((ii + 1) + "," + (j + 1));
            if (curItem instanceof Value) {
              data[ii] = (Double) ((Value) curItem).getValue();
            } else {
              data[ii] = 0.0;
            }
          }
          if (tColors != null) {
            layer.addDataSet(data, (int) Long.parseLong(tColors[j], 16),
AnalysisRenderingToolbox.getItemText(dim2.getItem(j), root));
          } else {
            layer.addDataSet(data, AnalysisRenderingToolbox.getHexaColor(j),
AnalysisRenderingToolbox.getItemText(dim2.getItem(j), root));
          }
```

```
        }


        // Get filename
        final String imgurl =
root.currentEnvironment().toolkit().getStringFromID(root.currentEnvironment().to
olkit().generateID()) + ".jpg";
        // Generate chart
        File f = new
File(AnalysisRenderingToolbox.getGenerationFolderWrite(megaContext, root) +
imgurl);
        f.deleteOnExit();
        c.makeChart(f.getAbsolutePath());
        // append PDF
        Section sec1 = ((Pdf) document).getSections().add();
        sec1.setIsNewPage(false);
        //Create an image object in the section
        Image img1 = new Image(sec1);

img1.getImageInfo().setTitle(AnalysisRenderingToolbox.getCodeTemplate(d.getCodeT
emplateID(), root));
        img1.getImageInfo().setFixWidth(400.0f);
        //Add image object into the Paragraphs collection of the section
        sec1.getParagraphs().add(img1);
        //Set the path of image file
        img1.getImageInfo().setFile(f.getAbsolutePath());
        return true;
      }
      return false;
    }
    return false;
  }

}
```

Example result

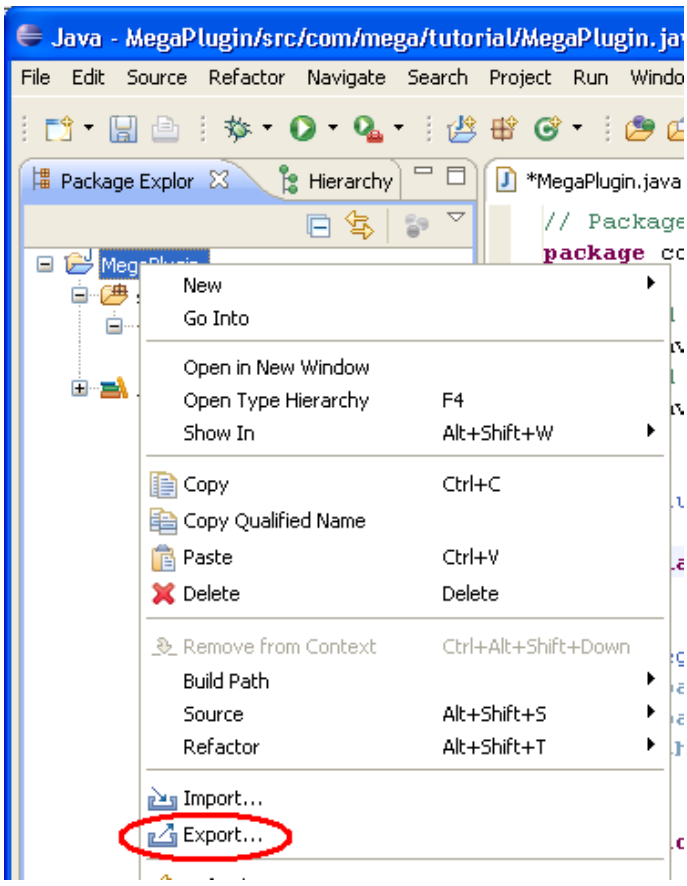


USING YOUR IMPLEMENTATION FROM THE MEGA RENDERER MACRO

Compile

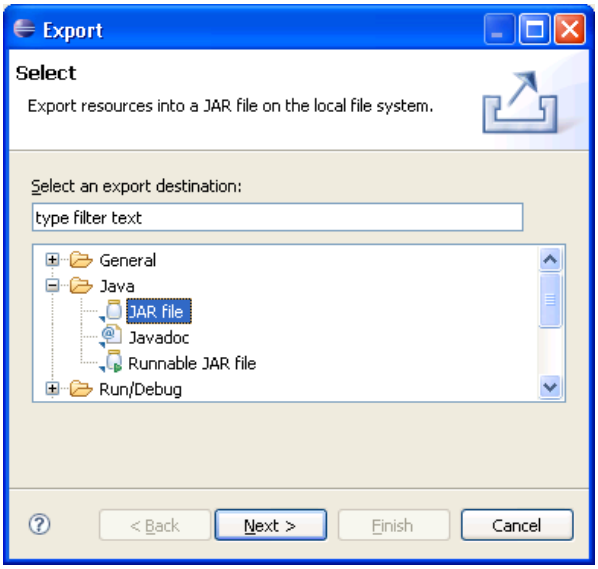
In order to use your Java Report Renderer it must be exported in a .jar in the java/lib directory of your MEGA installation.

Compilation of the Java component in the form of a JAR file is via the "Export" menu of the Java project:



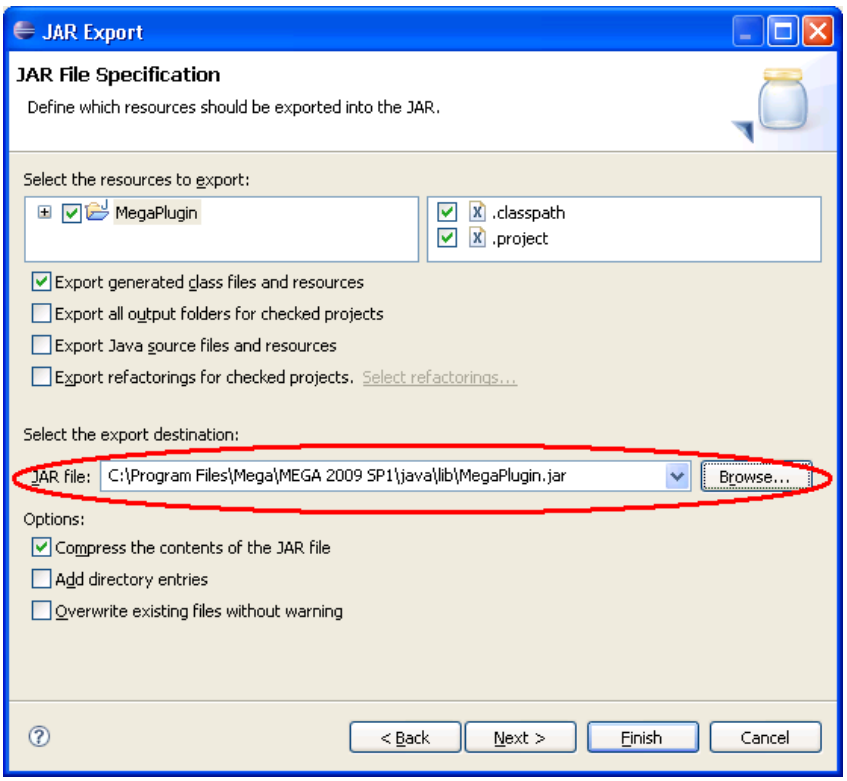
A JAR can contain as many Java Renderer implementing classes as you wish.

"JAR File" export in the Java directory should be selected:



Indicate the location of the JAR file to be generated and click "Finish":

The JAR file **must** be generated (or copied after generation) in the "java\lib" directory of the MEGA installation site.

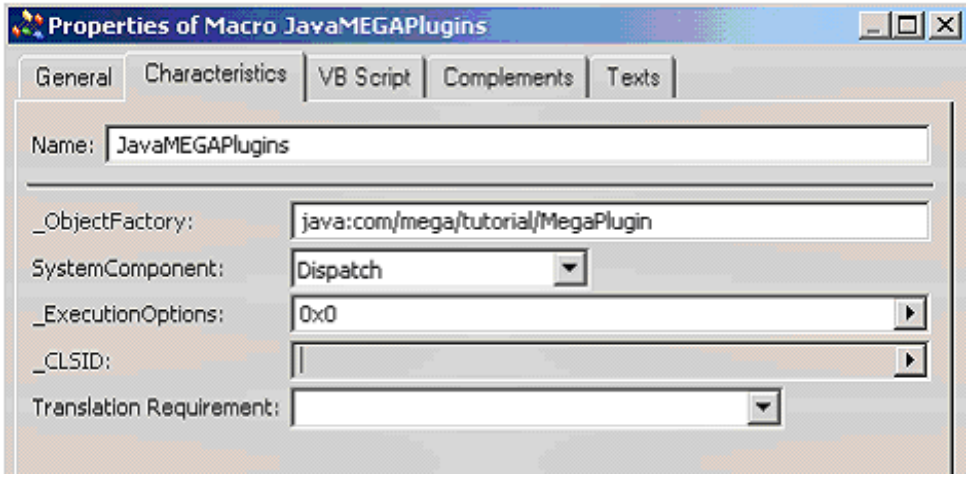


The name of the JAR file itself is not significant; you should use a name that makes sense in your project.

Configuring the macro

Once you have added the JAR file to the "java\lib" directory, restart Mega and edit the properties of your renderer macro in order to reference your Java class.

In the macro properties dialog box, assign the "Dispatch" value to the "SystemComponent" attribute, then specify the "_ObjectFactory" attribute using the renderer Java class. For exemple, the value "java:com/mega/tutorial/MegaPlugin" identifies the "MegaPlugin" class of the "com.mega.tutorial" package.



The VB Script of the macro should be kept empty.

Writing Java report chapters

Starting from Mega 2009 SP4 it is possible to write report chapters in Java. This article explains how to do so, the new architecture and the differences with VBS analyses.

INTRODUCTION TO JAVA REPORT CHAPTERS

Generalities

Starting with MEGA 2009 SP4, a report chaptercan be written in Java.

It is highly recommended to write new reports in this language in order to benefit from the platform improvements and also to better handle PDF/RTF document generation. The old report platform will not be improved whereas the Java platform will be.

Compared to VB Script reports, Java reports are written differently as described in this article.

Both use the same metamodel described in the product documentation.

Data and view separation

VB Script report chapter macros generate HTML.

Java report chapter macros generate an object structure describing datasets (report data) and the type of view to apply to these datasets.

For more details on this structure, you should refer to the technical article “Java report data structure”.

Conversion from this object structure to a report output (HTML or PDF for example) is handled by the report engine using specific renderers. Renderers are the subject of another technical article “Writing Java report renderers”.

> This new architecture allows for better management of different output formats and more flexibility and modularity. It is therefore highly recommended to write new report chapter macros in Java, using the method described hereafter.

PREREQUISITES

Report modeling

This technical article does not cover modeling of reports, report templates, report parameters and report chapters.

Modeling is the same as for VB Script reports. You should refer to product documentation on this subject.

Setup a Java development environment

A Java report chapter macro is a Java Plug-in which uses Mega APIs.

You should first refer to the technical article “Creating MEGA Plug-ins with Java” to set up a Java development environment adapted to MEGA.

Referencing jars and javadoc

Following the method described in “Creating MEGA Plug-ins with Java” you should reference the following jars in your development environment:

- mj_toolkit.jar
- mj_api.jar
- mj_anls.jar

To allow for contextual help in Eclipse, you should also reference the corresponding javadocs:

- mj_api.doc.zip
- mj_anls.doc.zip

WRITING THE MACRO IMPLEMENTATION

Interfaces to implement

Report chapters

A Java report chapter must implement one of the following interfaces:

- `com.mega.modeling.analysis.AnalysisReport`
- `com.mega.modeling.analysis.AnalysisReportWithContext`

The `getReportContent` method is the only method required to implement these interfaces. Its parameters are:

- a Mega Root,
- a Map of parameter Lists referenced by the HexaIdAbs of the "Analysis Parameter" repository object,
- an optional userData object,
- and in the case of `AnalysisReportWithContext` an extra Analysis object which provides contextual information.

It produces the report content in the form of a ReportContent object. This is an instance of the ReportContent Java class, to which all data and view information is given in order to pass it to the analysis engine and its renderers.

More information on ReportContent objects is available in the "Java analysis data structure" technical article and in the engine Javadoc.

This typically results in either of the following structures:

```
public class MyReport implements AnalysisReport {

  @Override

  public ReportContent getReportContent(final MegaRoot root, final Map<String,
List<AnalysisParameter>> parameters, final Object userData) {

    …

  }

}
```

```
public class MyReport implements AnalysisReportWithContext {

  @Override

  public ReportContent getReportContent(final MegaRoot root, final Map<String,
List<AnalysisParameter>> parameters, final Analysis analysis, final Object
userData) {

    …

  }

}
```

Callbacks

Callback calls allow for user interactivity in tables and trees, by allowing the execution of a Callback function when the user clicks on a cell. The callback function subsequently updates the cell content.

The Java implementation of the macro that handles callback calls must implement the `com.mega.modeling.analysis.AnalysisCallback` interface.

This can be the same macro and Java class as the report chapter itself or a different macro.

The `Callback` method is the only method required to implement the AnalysisCallback interface. Its parameters are:

- a Mega Root,
- the HTML content of the cell that was clicked,
- MegaCollections of the line and column objects of the cell,
- an optional userData.

It produces the new content to be rendered in the clicked cell, in the form of an HTML string.

For example:

```
public class MyReportCallback implements AnalysisCallback {

  @Override

  public String Callback(final MegaRoot root, final String
HTMLCellContent, final MegaCollection ColumnMegaObjects, final
MegaCollection LineMegaObjects, final Object userData) {

    …

  }

}
```

Implementing the macro

General steps

The implementation of the getReportContent function follows the following general steps:

- Create a new ReportContent object that will contain all report data and views:

```
final ReportContent reportContent = new ReportContent("");
```

- Create one or more Datasets (Java objects that contain all the data to be rendered) and add them to the ReportContent, getting the ID of the dataset for future use:

```
final Dataset d = new Dataset("");
final int datasetID = reportContent.addDataset(d);
```

- Create one or more Dimensions (the equivalent of x, y, etc. axis in a diagram) and add them to the Dataset(s):

```
final Dimension dim = new Dimension("");
d.addDimension(dim);
```

- Create one or more items and add them to the Dimension(s).This is the equivalent of line or column headers in a table.

```
dim.addItem(new Text("Some text…", false));
```

- Create one or more items and add them to the Dataset(s):

```
d.addItem(new Value((double) n),"1”);
```

- Create one or more Views (or Texts or MegaObjectProperties), using the ID of the dataset they should represent, and add them to the ReportContent. A view links a dataset to a renderer in order to define where and how the dataset should be rendered.

```
final View v = new View(datasetID);
reportContent.addView(v);
```

- Add one or more renderers to the View(s) to specify how the view dataset should be shown (here, as a table):

```
v.addRenderer(AnalysisReportToolbox.rTable);
```

- Return the now complete ReportContent:

```
return reportContent;
```

More information on the data structure of ReportContent, Dataset, Dimension, View, Item, etc. is available in the “Java report data structure” technical article and in the engine Javadoc.

Example

A basic working example is as follows:

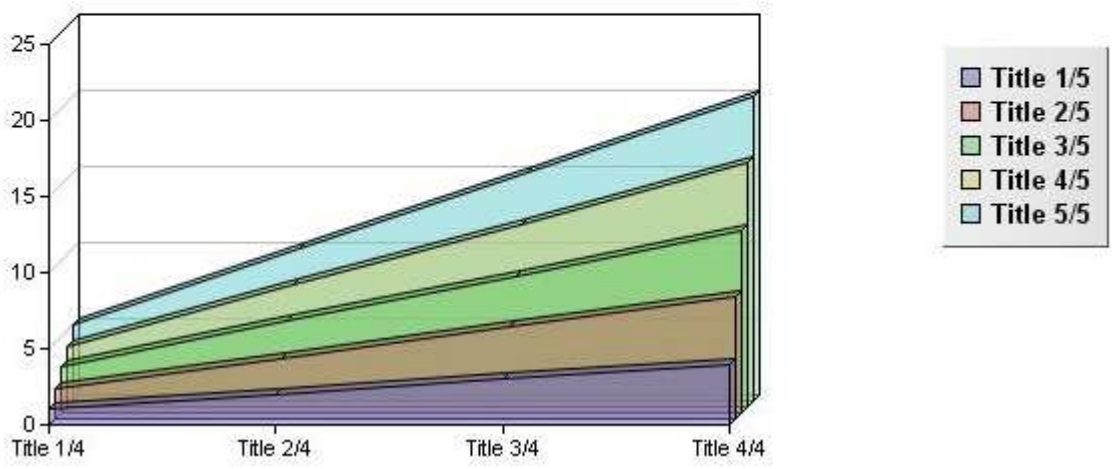
```
import java.util.List;
import java.util.Map;


import com.mega.modeling.analysis.AnalysisParameter;
```

```
import com.mega.modeling.analysis.AnalysisReport;
import com.mega.modeling.analysis.AnalysisReportToolbox;
import com.mega.modeling.analysis.content.Dataset;
import com.mega.modeling.analysis.content.Dimension;
import com.mega.modeling.analysis.content.ReportContent;
import com.mega.modeling.analysis.content.Text;
import com.mega.modeling.analysis.content.Value;
import com.mega.modeling.analysis.content.View;
import com.mega.modeling.api.MegaRoot;


/**
 * This is a basic demonstration report.
 * @author NLE
 */


public class MyReport implements AnalysisReport {

  public ReportContent getReportContent(final MegaRoot root, final Map<String,
List<AnalysisParameter>> parameters, final Object userData) {
    final ReportContent reportContent = new ReportContent("");


    // Creating a 2D Dataset and its dimensions
    final Dataset d2 = new Dataset("");
    final Dimension dim21 = new Dimension("");
    final Dimension dim22 = new Dimension("");
    // Set the dimension sizes
    // (compulsory only when no Items are added to the dimension)
    dim21.setSize(4);
    dim22.setSize(5);
    // Add the dimensions to the Dataset
    d2.addDimension(dim21);
    d2.addDimension(dim22);
```

```
    // Filling in the dataset and its dimensions
    // Arbitrary data is used here
    for (int i = 1; i <= 4; i++) {
      dim21.addItem(new Text("Title " + i + "/4", false));
      for (int j = 1; j <= 5; j++) {
        d2.addItem(new Value((double) i * j), i + "," + j);
      }
    }
    for (int j = 1; j <= 5; j++) {
      dim22.addItem(new Text("Title " + j + "/5", false));
    }

    // Add the dataset to the report
    final int datasetID = reportContent.addDataset(d2);

    // Add a table and list view of the dataset
    // List will only be used if table cannot be used
    final View v1 = new View(datasetID);
    v1.addRenderer(AnalysisReportToolbox.rTable);
    v1.addRenderer(AnalysisReportToolbox.rList);
    reportContent.addView(v1);

    // Add an area chart view, with the same dataset (not duplicated)
    final View v2 = new View(datasetID);
    v2.addRenderer(AnalysisReportToolbox.rAreaChart);
    reportContent.addView(v2);

    return reportContent;
  }
}
```

This code will typically result in the following rendering:

	Title 1/5	Title 2/5	Title 3/5	Title 4/5	Title 5/5
Title 1/4	1.0	2.0	3.0	4.0	5.0
Title 2/4	2.0	4.0	6.0	8.0	10.0
Title 3/4	3.0	6.0	9.0	12.0	15.0
Title 4/4	4.0	8.0	12.0	16.0	20.0



Going further

You should refer to the Javadoc for all possible options and possibilities. The Javadoc is accessible contextually in Eclipse if you configured it as suggested in the above chapter, as well as in HTML format in a zip file "mj_anls.doc.zip" in the "java\doc" directory of your MEGA installation.

You can of course make use of all MEGA Java APIs (documented in the "mj_api.doc.zip" javadoc) to handle MegaObjects and MegaCollections.

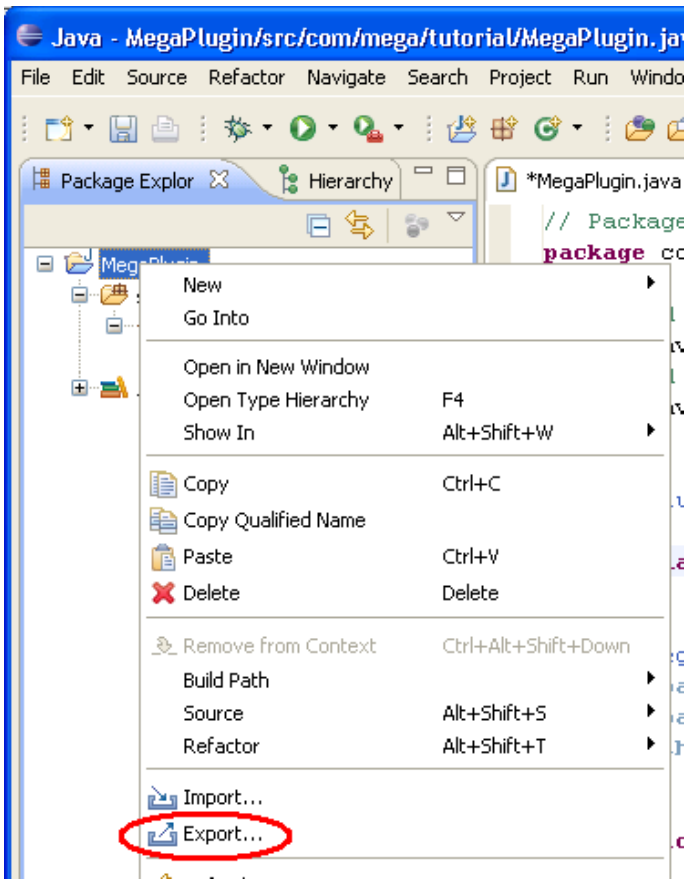
A more complex example is provided at the end of this document.

USING YOUR IMPLEMENTATION FROM THE MEGA REPORT MACRO

Compile

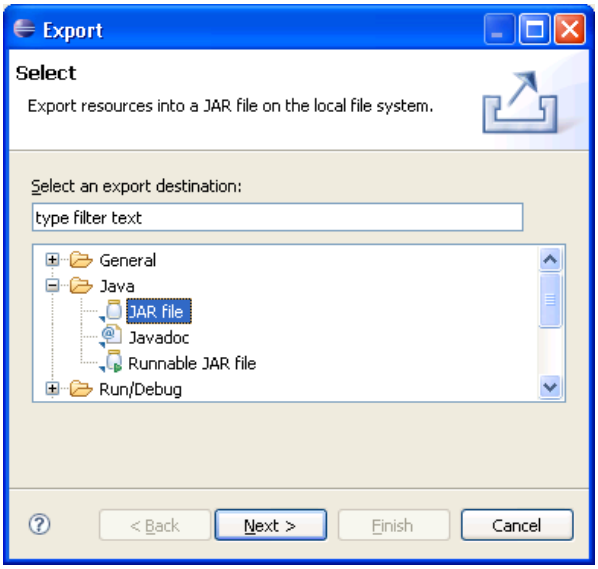
In order to use your Java report chapter, it must be exported in a .jar in the java/lib directory of your MEGA installation.

Compilation of the Java component in the form of a JAR file is via the "Export" menu of the Java project:



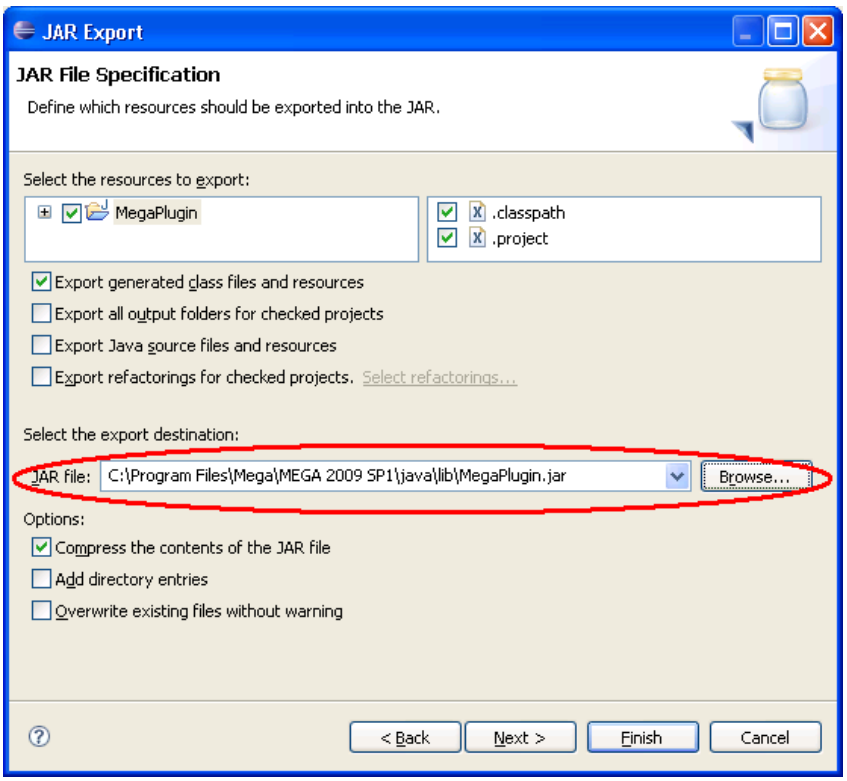
A JAR can contain as many Java report chapter implementing classes as you wish.

"JAR File" export in the Java directory should be selected:



Indicate the location of the JAR file to be generated and click "Finish":

The JAR file **must** be generated (or copied after generation) in the "java\lib" directory of the MEGA installation site.

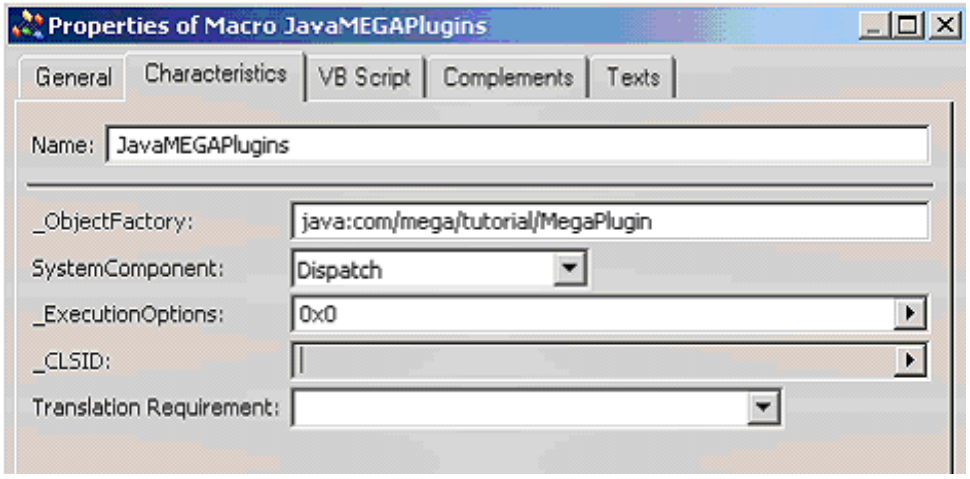


The name of the JAR file itself is not significant; you should use a name that makes sense in your project.

Configure the Macro

Once you have added the JAR file to the "java\lib" directory, restart Mega and edit the properties of your report macro in order to reference your Java class.

In the macro properties dialog box, assign the "Dispatch" value to the "SystemComponent" attribute, then specify the "_ObjectFactory" attribute using the report Java class. For example, the value "java:com/mega/tutorial/MegaPlugin" identifies the "MegaPlugin" class of the "com.mega.tutorial" package.



The VB Script of the macro should be left empty.

CODE EXAMPLE

```
import java.util.Date;
import java.util.List;
import java.util.Map;

import com.mega.modeling.analysis.AnalysisCallback;
import com.mega.modeling.analysis.AnalysisParameter;
import com.mega.modeling.analysis.AnalysisReport;
import com.mega.modeling.analysis.AnalysisReportToolbox;
import com.mega.modeling.analysis.content.Dataset;
import com.mega.modeling.analysis.content.Dimension;
import com.mega.modeling.analysis.content.Item;
import com.mega.modeling.analysis.content.MegaObjectProperty;
import com.mega.modeling.analysis.content.ReportContent;
import com.mega.modeling.analysis.content.Text;
import com.mega.modeling.analysis.content.Value;
import com.mega.modeling.analysis.content.View;
import com.mega.modeling.api.MegaCollection;
import com.mega.modeling.api.MegaObject;
import com.mega.modeling.api.MegaRoot;
import com.mega.modeling.api.MegaAttribute.OutputFormat;
import com.mega.toolkit.errmngt.ErrorLogFormater;

/**
 * This is a demonstration report.
 * @author NLE
 */
```

```
public class MyReport implements AnalysisReport, AnalysisCallback {

  @Override

  public ReportContent getReportContent(final MegaRoot root, final
Map<String, List<AnalysisParameter>> parameters, final Object userData) {

    // Error Management exemple

    final ErrorLogFormater err = new ErrorLogFormater();

    err.openSession(root);

    // Do not forget to update this line

    err.addSessionInfo("Component", "(Java) New Analysis Engine:
TestReport: getReportContent");


    // Initialize the report content

    final ReportContent reportContent = new ReportContent("");


    try {
```

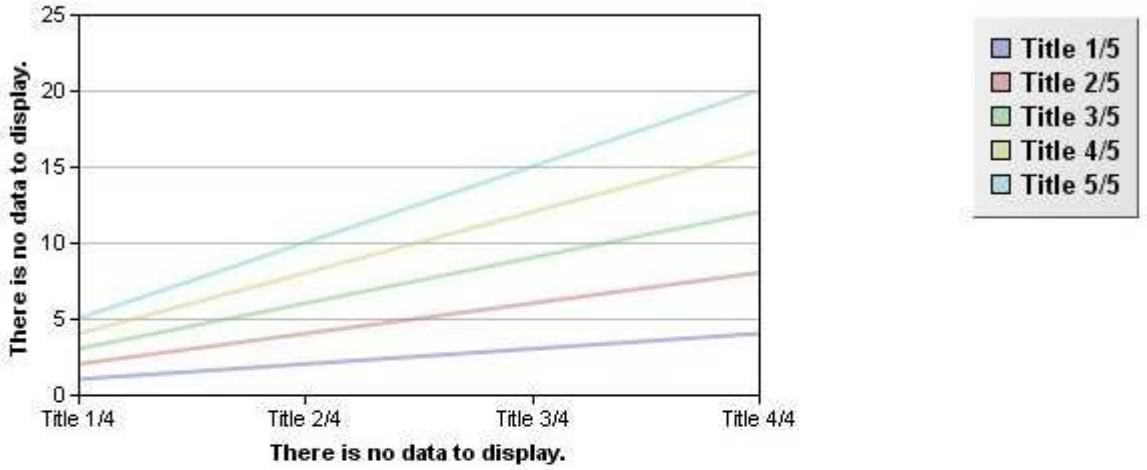
Demo 1: Charts using 2D datasets

```
      final Dataset d2 = new Dataset("~LshNx7Mw6zE0[No Data To
Display]");

      final Dimension dim21 = new Dimension("~LshNx7Mw6zE0[No Data To
Display]");

      final Dimension dim22 = new Dimension("~LshNx7Mw6zE0[No Data To
Display]");

      dim21.setSize(4);

      dim22.setSize(5);

      d2.addDimension(dim21);

      d2.addDimension(dim22);


      // Filling in the dataset (here with arbitrary data)
```

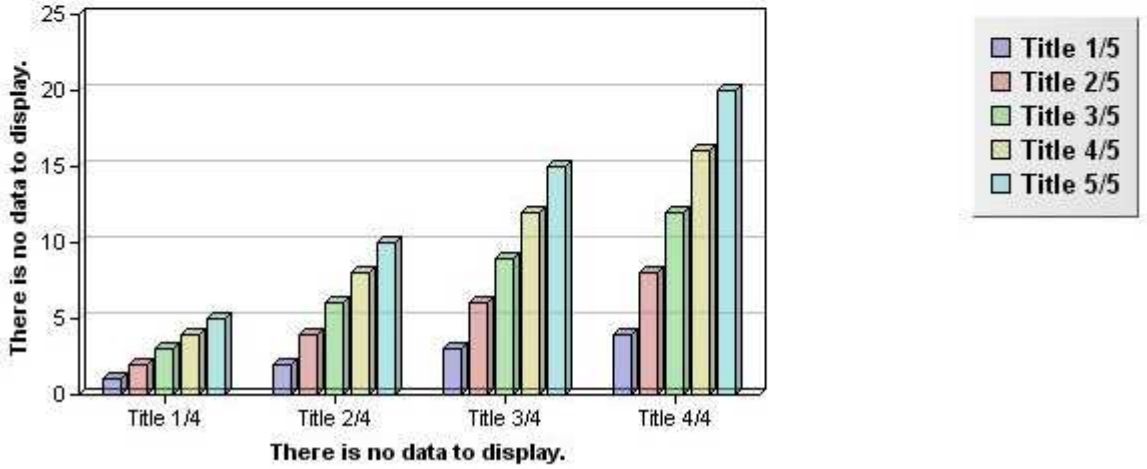
```
for (int i = 1; i <= 4; i++) {
  dim21.addItem(new Text("Title " + i + "/4", false));
  for (int j = 1; j <= 5; j++) {
    d2.addItem(new Value((double) i * j), i + "," + j);
  }
}
for (int j = 1; j <= 5; j++) {
  dim22.addItem(new Text("Title " + j + "/5", false));
}

// Add the Dataset to the report
final int datasetID = reportContent.addDataset(d2);

// Add the Dataset to many different views
final View v21 = new View(datasetID, true, false);
v21.addRenderer(AnalysisReportToolbox.rAreaChart);
reportContent.addView(v21);

final View v22 = new View(datasetID);
v22.addRenderer(AnalysisReportToolbox.rLineChart);
reportContent.addView(v22);

final View v23 = new View(datasetID);
v23.addRenderer(AnalysisReportToolbox.rBarChart);
reportContent.addView(v23);

final View v24 = new View(datasetID);
v24.addRenderer(AnalysisReportToolbox.rRadarChart);
reportContent.addView(v24);
```

⊞ There is no data to display.

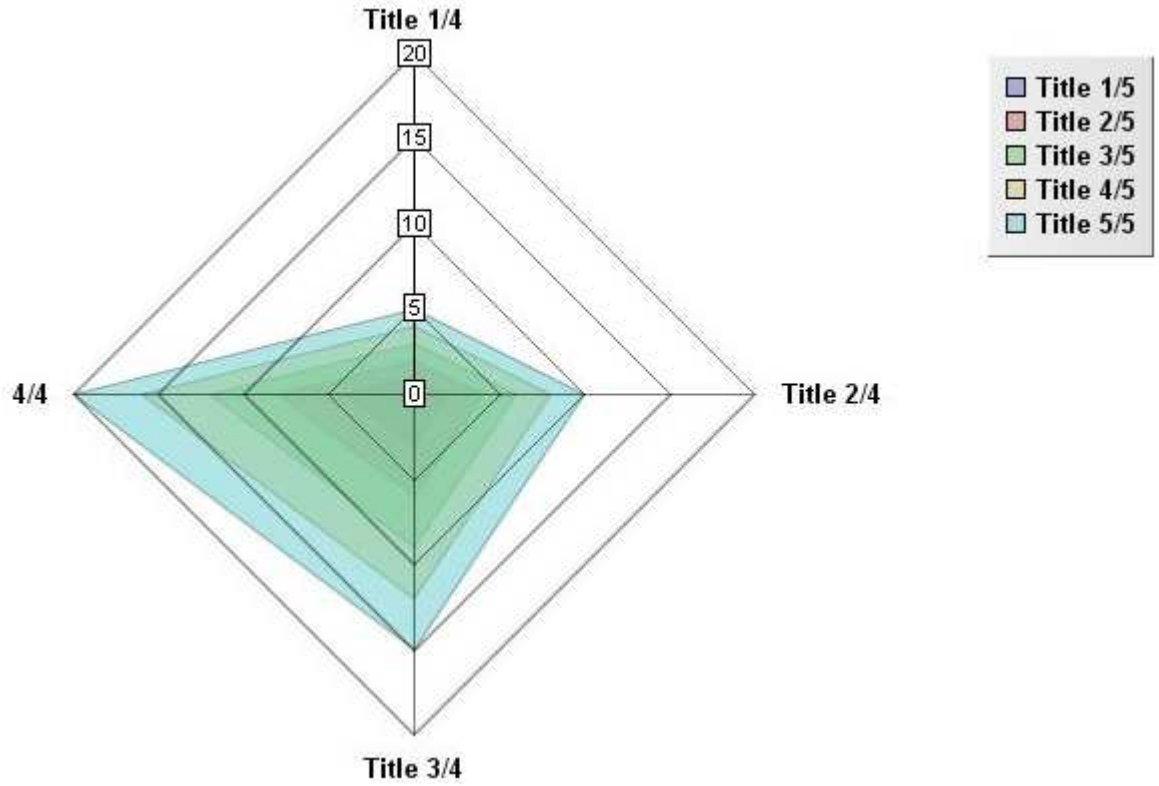
There is no data to display.



There is no data to display.



There is no data to display.



Demo 2: Parameters in tables with vertical headers

```
      // Going through parameters
      for (final String paramType : parameters.keySet()) {
        reportContent.addText(new Text("" +
root.getObjectFromID(paramType).getAttribute("~Z20000000D60[Short
Name]").getFormated(OutputFormat.html, ""), false, 3));


        // Going through its values
        for (final AnalysisParameter paramValue :
parameters.get(paramType)) {
          reportContent.addText(new
Text(paramValue.getParameterObject().getAttribute("~Z20000000D60[Short
Name]").getFormated(OutputFormat.html, ""), false, 4));
```

```
            // and the actual individual values
            final Dataset paramDataset = new Dataset("");
            final Dimension dim1 = new Dimension("");
            final Dimension dim2 = new Dimension("");
            dim2.setSize(1);
            final Item title = new Text("Column Title", false);
            // Set the title column header to allow ordering of column
            title.setOrderable(true);
            dim2.addItem(title);
            int i = 1;
            for (final MegaObject value : paramValue.getValues()) {
              paramDataset.addItem(new
MegaObjectProperty(value.megaField(), "~Z20000000D60[Short Name]"), i +
",1");
              i++;
            }
            dim1.setSize(i - 1);
            paramDataset.addDimension(dim1);
            paramDataset.addDimension(dim2);


            final int paramDatasetID =
reportContent.addDataset(paramDataset);


            final View paramView1 = new View(paramDatasetID, true, false);

paramView1.addRenderer(AnalysisReportToolbox.rVerticalTextTable);
            reportContent.addView(paramView1);


          }
        }
```

Prohibited Technology

Prohibited Technology

Accepted Technology

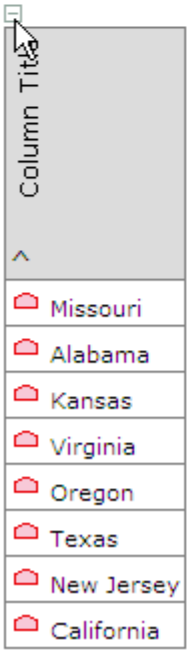
Accepted Technology

Expected Technology

Expected Technology

Information System

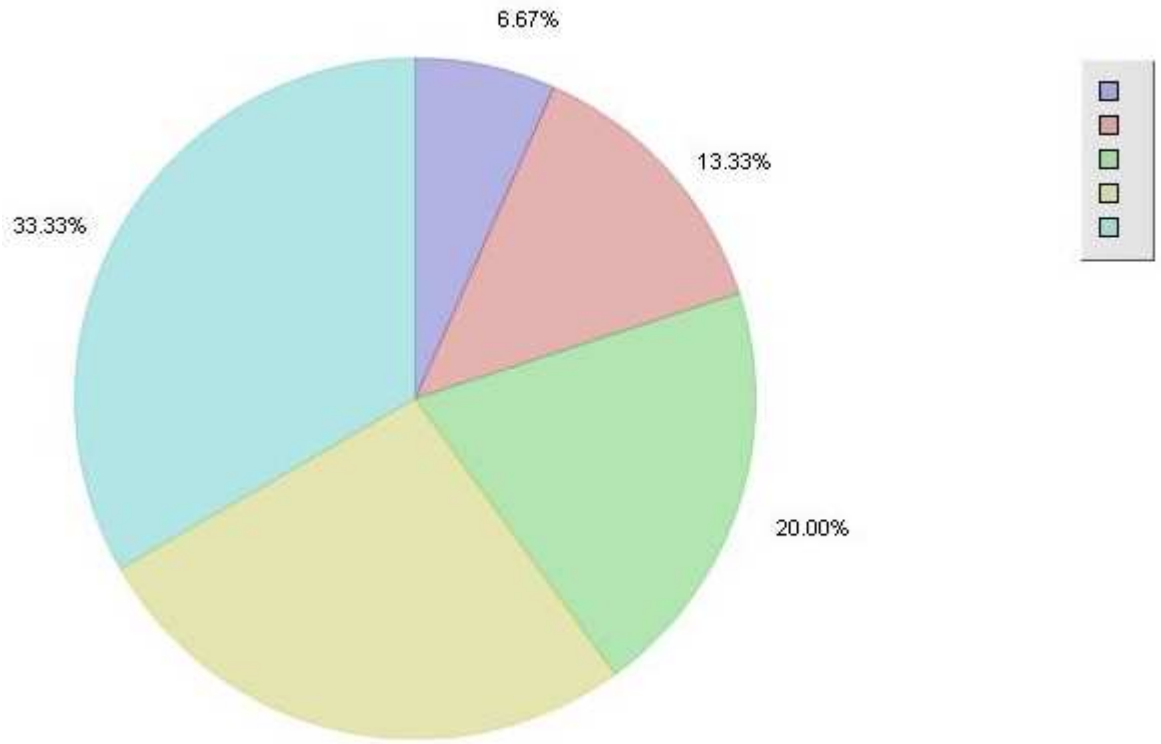
American States



Demo 3: Pie chart

```
final Dataset d1 = new Dataset("");
```

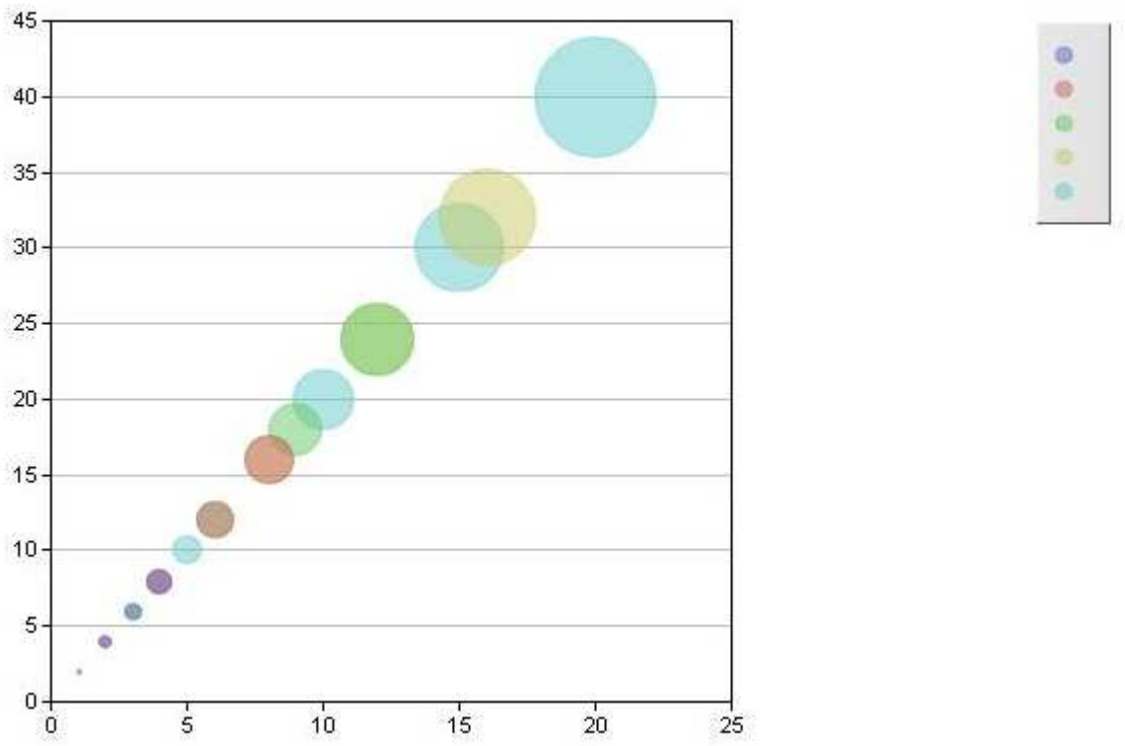
```
final Dimension dim11 = new Dimension("");
dim11.setSize(5);
d1.addDimension(dim11);
for (int i = 1; i <= 5; i++) {
  d1.addItem(new Value((double) i), i + "");
}
final View v1 = new View(reportContent.addDataset(d1));
v1.addRenderer(AnalysisReportToolbox.rPieChart);
reportContent.addView(v1);
```



Demo 4: Bubble chart

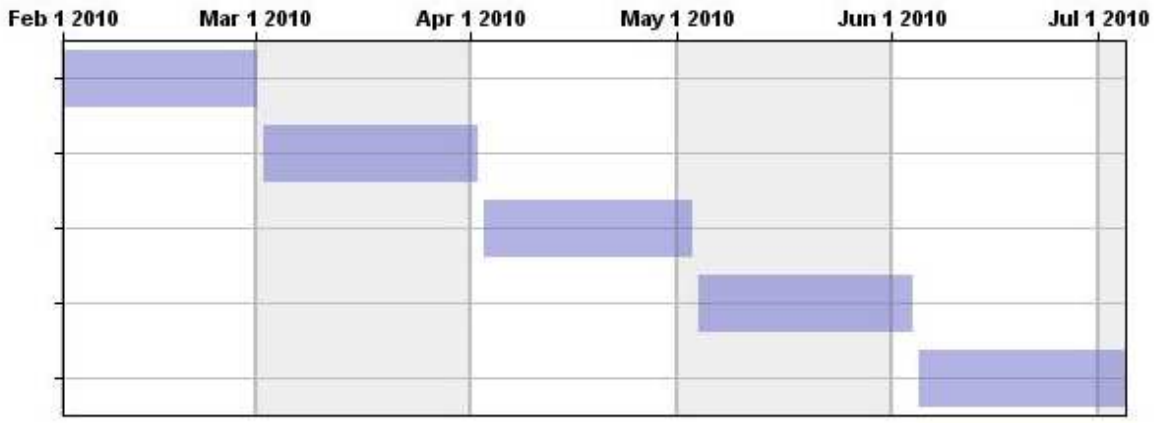
```
final Dataset d3 = new Dataset("");
final Dimension dim31 = new Dimension("");
final Dimension dim32 = new Dimension("");
```

```
      final Dimension dim33 = new Dimension("");
      dim31.setSize(5);
      dim32.setSize(4);
      dim33.setSize(3);
      d3.addDimension(dim31);
      d3.addDimension(dim32);
      d3.addDimension(dim33);
      for (int i = 1; i <= 5; i++) {
        for (int j = 1; j <= 4; j++) {
          for (int k = 1; k <= 3; k++) {
            d3.addItem(new Value((double) i * j * k), i + "," + j + "," +
k);
          }
        }
      }
      final View v3 = new View(reportContent.addDataset(d3));
      v3.addRenderer(AnalysisReportToolbox.rBubbleChart);
      reportContent.addView(v3);
```



Demo 5: Simple Gantt chart

```
      final Dataset dGantt = new Dataset("");
      final Dimension dimGantt = new Dimension("");
      dimGantt.setSize(5);
      dGantt.addDimension(dimGantt);
      for (int i = 1; i <= 5; i++) {
        dGantt.addItem(new Value(new Date(2010 - 1900, i, i), new
Date(2010 - 1900, i + 1, i)), i + "");
      }
      final View vGantt = new View(reportContent.addDataset(dGantt));
      vGantt.addRenderer(AnalysisReportToolbox.rGanttChart);
      reportContent.addView(vGantt);
```



Demo 6: Tree of parameters

```
      final Dataset paramDataset = new Dataset("");
      // Callback: set the Macro to be called back, in this exemple you
should
      // reference the macro referencing this Java class
      paramDataset.setCallback("~Jq(Ipv4W4P50[Analysis - Set of
Parameters]");
      final Dimension dim1 = new Dimension("");
      final Dimension dim2 = new Dimension("");
      dim2.setSize(1);
      int i = 0;
      for (final String paramType : parameters.keySet()) {
        dim1.addItem(new
MegaObjectProperty(root.getObjectFromID(paramType).megaField(),
"~Z20000000D60[Short Name]"), 1);
        i++;
        paramDataset.addItem(new
MegaObjectProperty(root.getObjectFromID(paramType).megaField(),
"~210000000900[Name]"), i + ",1");
        // Going through its values
```

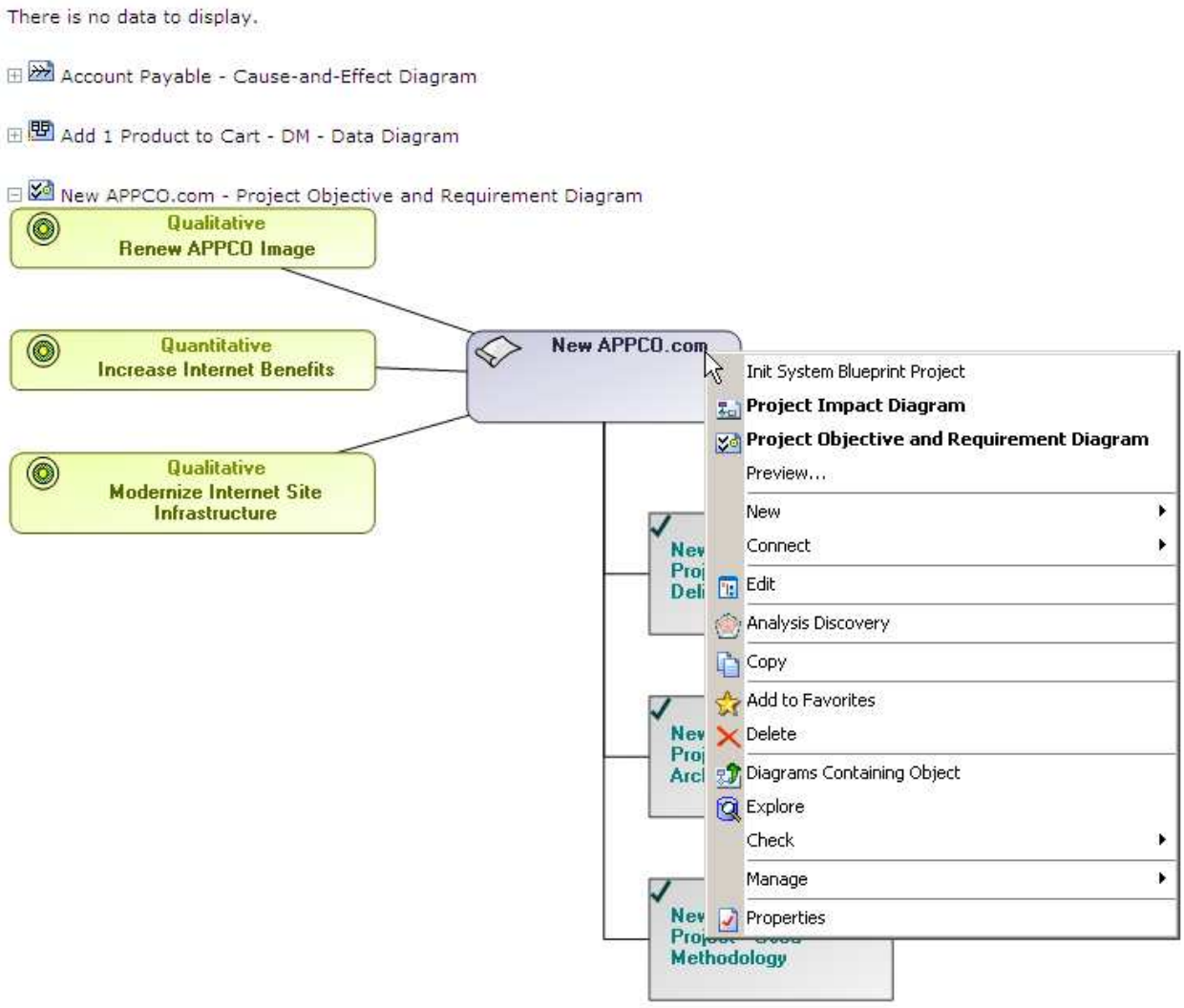
```
        for (final AnalysisParameter paramValue :
parameters.get(paramType)) {

          dim1.addItem(new
MegaObjectProperty(paramValue.getParameterObject().megaField(),
"~Z20000000D60[Short Name]"), 2);

          i++;

          paramDataset.addItem(new
MegaObjectProperty(paramValue.getParameterObject().megaField(),
"~210000000900[Name]"), i + ",1");

          // and the actual individual values!!

          for (final MegaObject value : paramValue.getValues()) {

            dim1.addItem(new MegaObjectProperty(value.megaField(),
"~Z20000000D60[Short Name]"), 3);

            i++;

            paramDataset.addItem(new
MegaObjectProperty(value.megaField(), "~210000000900[Name]"), i + ",1");

          }


        }
      }

      paramDataset.addDimension(dim1);

      paramDataset.addDimension(dim2);

      final View treeView = new
View(reportContent.addDataset(paramDataset));

      treeView.addRenderer(AnalysisReportToolbox.rTree);

      treeView.addRenderer(AnalysisReportToolbox.rTable);

      reportContent.addView(treeView);
```

Prohibited Technology	Infrastructure Compliance::Prohibited Technology
Accepted Technology	Infrastructure Compliance::Accepted Technology
Expected Technology	Infrastructure Compliance::Expected Technology
Information System	Infrastructure Compliance::Information System
American States	.Net Architecture Compliance::American States
Missouri	Missouri
Alabama	Alabama
Kansas	Kansas
Virginia	Virginia
Oregon	Oregon
Texas	Texas
New Jersey	New Jersey
California	California

Demo 7: Clickable diagrams

```
	final Dataset dDiags = new Dataset("~LshNx7Mw6zE0[No Data To Display]");
	final Dimension dimD1 = new Dimension("");
	dDiags.addDimension(dimD1);

	// filling in the dataset
	dimD1.addItem(new MegaObjectProperty("~uGEJcPMZ4fM0[Account Payable - Cause-and-Effect Diagram]", ""));
	dimD1.addItem(new MegaObjectProperty("~B9QnnNye9940[Add 1 Product to Cart - DM - Data Diagram]", ""));
	dimD1.addItem(new MegaObjectProperty("~vU3v8M(a9L50[New APPCO.com - Project Objective and Requirement Diagram]", ""));
	final int datasetDiagID = reportContent.addDataset(dDiags);

	final View vDiag = new View(datasetDiagID);
```

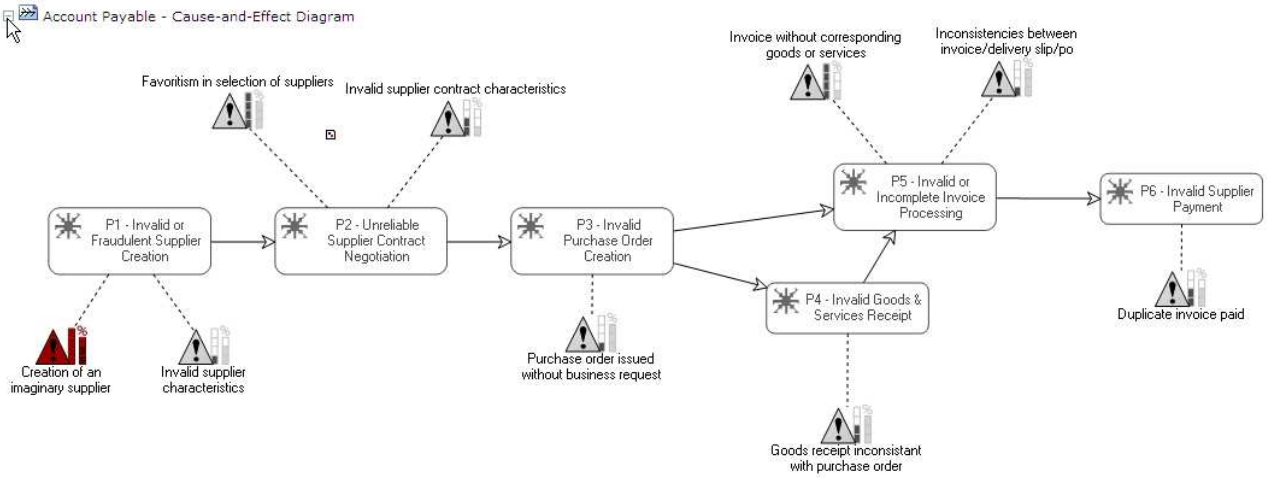
```
vDiag.addRenderer(AnalysisReportToolbox.rDiagrams);
reportContent.addView(vDiag);
```



Demo 8: Clickable illustrating diagrams

```
final Dataset diDiags = new Dataset("");
final Dimension dimiD1 = new Dimension("");
diDiags.addDimension(dimiD1);
```

```
      // filling in the dataset with the objects we want to find in the
diagram
      // and the color to apply to them
      dimiD1.addItem(new MegaObjectProperty("~4YRLwW5V49o0[Creation of an
imaginary supplier]", ""));
      final int datasetiDiagID = reportContent.addDataset(diDiags);
      diDiags.addItem(new Value((short) 200, (short) 0, (short) 0), "1");

      final View viDiag = new View(datasetiDiagID);
      viDiag.addRenderer(AnalysisReportToolbox.rIllustratingDiagrams);
      reportContent.addView(viDiag);

      // End of error management
    } catch (final Exception e) {
      err.logMessage("Report generation failed:");
      err.logError(e);
      err.closeSession();
    }
    return reportContent;
  }

  @Override
  public String Callback(final MegaRoot root, final String
HTMLCellContent, final MegaCollection ColumnMegaObjects, final
MegaCollection LineMegaObjects, final Object UserData) {
    // Exemple of callback in a table or tree
    return "[TEST] Was called back successfully, was[" + HTMLCellContent
+ "]";
  }
}
```


Account Payable - Cause-and-Effect Diagram
Favoritism in selection of suppliers
Invalid supplier contract characteristics
Invoice without corresponding goods or services
Inconsistencies between invoice/delivery slip/po
P1 - Invalid or Fraudulent Supplier Creation
P2 - Unreliable Supplier Contract Negotiation
P3 - Invalid Purchase Order Creation
P5 - Invalid or Incomplete Invoice Processing
P6 - Invalid Supplier Payment
P4 - Invalid Goods & Services Receipt
Creation of an imaginary supplier
Invalid supplier characteristics
Purchase order issued without business request
Duplicate invoice paid
Goods receipt inconsistant with purchase order

Customizing Diagrams

Sommaire

CREATING AND EDITING SHAPES

The **Shapes Editor** tool enables you to create and modify shapes that can be used in diagrams.

☛ *The **Shapes Editor** is available in **MEGA Windows Front-End** only.*

A shape is composed of drawing elements and is saved in an individual file. This file can be treated like any other file.

Shapes are used by reference: if you make changes to a given shape, the shape is changed automatically in all diagrams that use it. In the diagrams, each object type is represented by a shape, so each object type has the same shape in all diagrams. This ensures consistency of presentation.

The graphical features of **MEGA** allow you to customize the default shapes. You can also use these features during diagramming to improve legibility.

```
An example would be to highlight an org-unit by surrounding
it with a border. Similarly, you can insert text at certain
places to explain an important point, without this comment
being assigned to a particular element.
```

☛ *For more details on graphical shapes handling, see the **MEGA Common Features** guide.*

The following functionalities are common to all products in **MEGA**:

Shapes Editor and Shapes

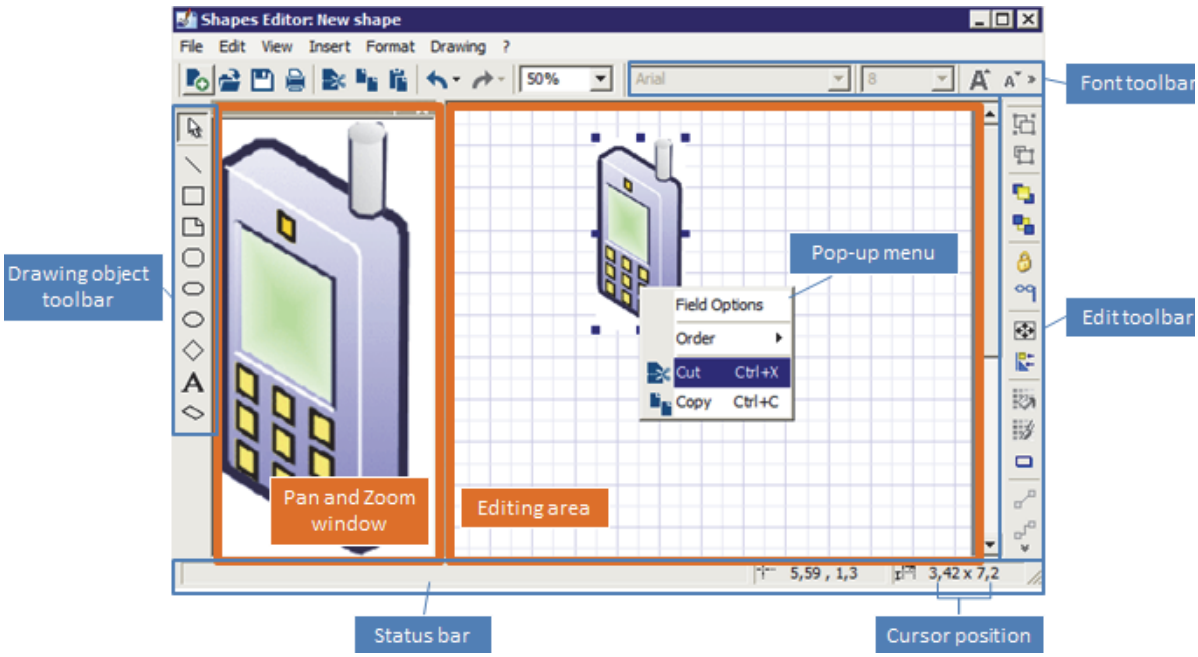
The **Shapes Editor** tool enables you to create and modify shapes that can be used in diagrams.

☛ *The **Shapes Editor** is available in **MEGA Windows Front-End** only.*

Opening the Shapes Editor

To opening the **Shapes Editor**:

1. From **MEGA** menu bar, select **Tools > Shape Editor**.
   The **Shapes Editor** opens.



The **Shapes Editor** includes:

- an editing area, to edit shapes

  ☛ *To edit shapes, see "Editing Shapes", page 103.*
- a pan and zoom window, which can be docked, fixed, enlarged, or hidden
- toolbars, which can be displayed or hidden

  ☛ *To display/hide a toolbar or status bar, see "Shapes Editor toolbars", page 103.*
- a status bar, which displays the horizontal and vertical position of the cursor in the editing area.

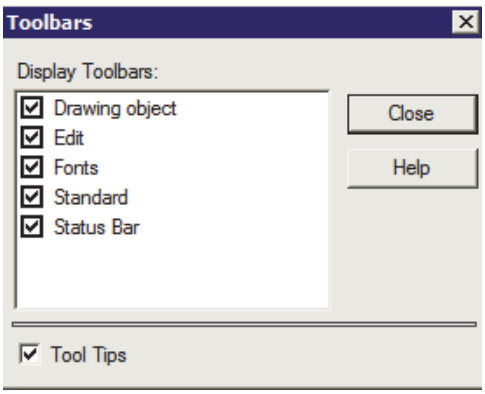
Shapes Editor toolbars

Toolbars include buttons that allow you to directly execute the menu commands.

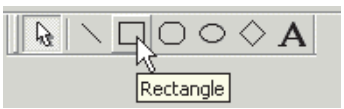
To enable/disable the display of toolbars:

1. From the **Shapes Editor** menu bar, select **View > Toolbar**.

   ☛ *Alternatively, you can right-click a toolbar.*



2. Unselect/Select the toolbars you want to hide/display.
3. (optional) Select/Clear **Tool Tips** option to display/hide the button name when the mouse move over the button.



To move a toolbar:

1. Select the toolbar.
2. Drag and drop it where you want.

Editing Shapes

You can create new shapes or modify the standard shapes provided by **MEGA**.
The shapes are stored in specific folders:

- Mega_Std folder for shapes common to the site
- Mega_Usr folder for the environment specific shapes.

☛ *See the **MEGA Administration - Supervisor** user guide for further information.*

☛ *The shapes in the MEGA_STD folder are read-only. To make changes to these shapes, use the file manager to copy them into the "Mega_Usr" sub-folder of your environment.*

To modify an existing shape:

- From the **Shapes Editor** menu bar, select **File > Open**.

To create a shape:

- From the **Shapes Editor** menu bar, select **File > New**.

To modify drawing objects:

- Right-click the drawing object you want to modify.

> ☛ *Note that the commands in this menu are also available in the **Format** and **Drawing** menus.*
>
> ☛ *For more details on graphical shapes handling, see the **MEGA Common Features** guide.*

Additional remarks on shapes

You can:

- run several instances of the shape editor at the same time, which enable you to access several shapes at the same time.
- call one shape from inside another.

Exchanging files with other software

You can transfer diagram drawings from one function to another by using **Cut**, **Copy** and **Paste**.

Exporting drawings

To export diagram drawings:

- From the **Shapes Editor** menu bar, select **File > Save As**.

Importing drawings

You can import drawings from applications operating under Windows, such as Paint or Photoshop.

Customized shapes should be saved in .mgs format (**MEGA** proprietary format). The shape can then be loaded.

After import, you can only enlarge or reduce the diagram size. Distortion may occur when modifying the size of a diagram dot-for-dot (bitmap).

> ☛ *To modify certain non-vectorial elements contained in .MGS (proprietary format) shapes, it may be necessary to cut and paste to the graphic editor in which they were produced.*

See:

- "Shapes Used in Diagrams", page 105
- "Positioning Objects in a Shape", page 107

SHAPES USED IN DIAGRAMS

For shapes that correspond to objects in the repository, you can define formatting for the object name.

To define object name formatting:

1. Create a text field and enter the value "&Name&" for the name displayed in this diagram.
2. Place it in the foreground.
3. Indicate the format for the text in the **Graphical Options** dialog box (**Format > Colors and Borders** menu).

To associate parts of the shape with the object name:

1. Select the shape parts and the object name (hold [Ctrl] key).
2. In the **Edit** toolbar, click **Group**  .
   In this case, the size of the selected shape parts is proportional to the name size and not to the overall object size.

   ☛ *Certain shapes display additional fields which depend on the type of object they represent.*

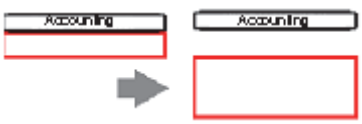
Tips on Using Shapes

Avoiding split shapes

To prevent drawing objects in a shape from drifting apart when you increase their size:

- Align the part that is not proportional to the name with the top left border.

When you increase the height of a shape, the space between the top of the drawing and the non-proportional objects increases with the overall size and the shape may drift apart.



Initial size

It is recommended that you check that the size specified for the shape will allow you to manipulate the shape without systematically distorting it.

Optimal distortion

Use a grid when defining shapes to ensure that any resizing is proportional. To work in finer detail, you can enlarge the shape overall, modify it and then reduce it overall.

Aligning links

Because links are drawn from the center of shapes, their alignment on the grid is simplified when they are of size that is a multiple of double the grid.

Optimizing performance

The display and printing speeds depend on the contents of the shapes:

- Shapes containing bitmaps and metafiles have low printing speeds.
- Thick lines take longer to display and greatly increase the size of the print file.
- Grouped elements slow down display of shapes. They should be reserved for the part that must be proportional to the name.
- Circles and rounded rectangles take longer to display than shapes with angles.

Reinitializing Shapes

You can reassign to drawing objects and links the graphic characteristics specified in shapes contained in the "Mega_Std" or "Mega_Usr" folder.

To reassign the repository graphic characteristics:

1. In the diagram menu, select **Drawing > Reinitialize Shapes**.

Example:

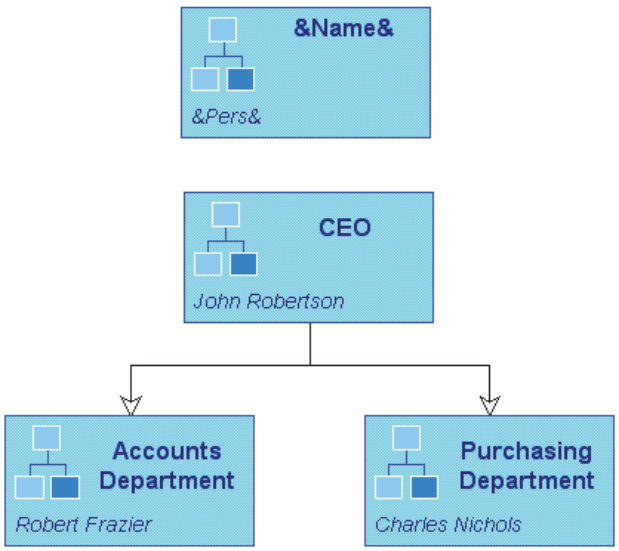
You have modified a shape font in the diagram. These modifications are lost when you perform a **Reinitialize Shapes**.

POSITIONING OBJECTS IN A SHAPE

When default deformation of a shape is not suitable, you can program its automatic deformation using a shape programming language.

This language sets size and position of fields and drawing objects when a shape is deformed. The **Field Positioning Wizard** automatically generates the code of this language.

> 🕮 *Method shape: shape representing methodological concept instances in diagrams (org-unit shape, operation shape).*
>
> 🕮 *Calculated shape: particular method shape object of which content depends on the instance represented. The real content is dynamically calculated when the diagram is loaded.*



Org-unit shape with org-unit and person names

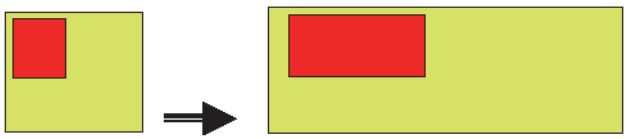
This text is identified by a name (&Name&, &pers&) derived from diagram configuration.

See:

- "Identifying Elementary Objects", page 108
- "Modifying Code Generation Specifications", page 108
- "Code Generation", page 111
- "Tips on Deformation Code Generation", page 112

Identifying Elementary Objects

By default, objects undergo deformation proportionally related to the shape.



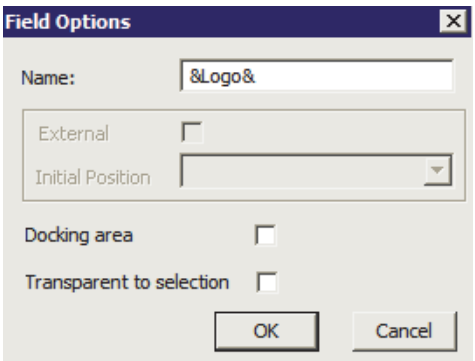
Before running the **Field Positioning Wizard**, objects that should not undergo proportional deformation (particular position, fixed size, etc.) should be named.

Each field name should be unique.

To name a shape object:

1. From the **Shapes Editor** menu bar, select **File > Open**.

   ☛ *To open the **Shapes Editor** see "Opening the Shapes Editor", page 102.*
2. Select a shape.
3. In the **Shapes Editor Editing area**, right-click the shape and select **Field Options**.
   The **Field Options** dialog box appears.



4. In the **Name** field, enter a name for the shape. The name should be prefixed and suffixed by &.

   `Example: &Logo&`

   ☛ *Calculated fields are self-named. The text they contain can be used directly as the object identifier (for example &Name&).*

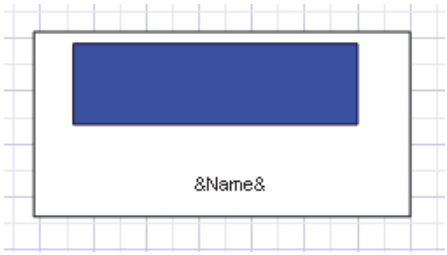
Modifying Code Generation Specifications

When your shape has been drawn and the elements named, you can complete specifications for the corresponding code generation.

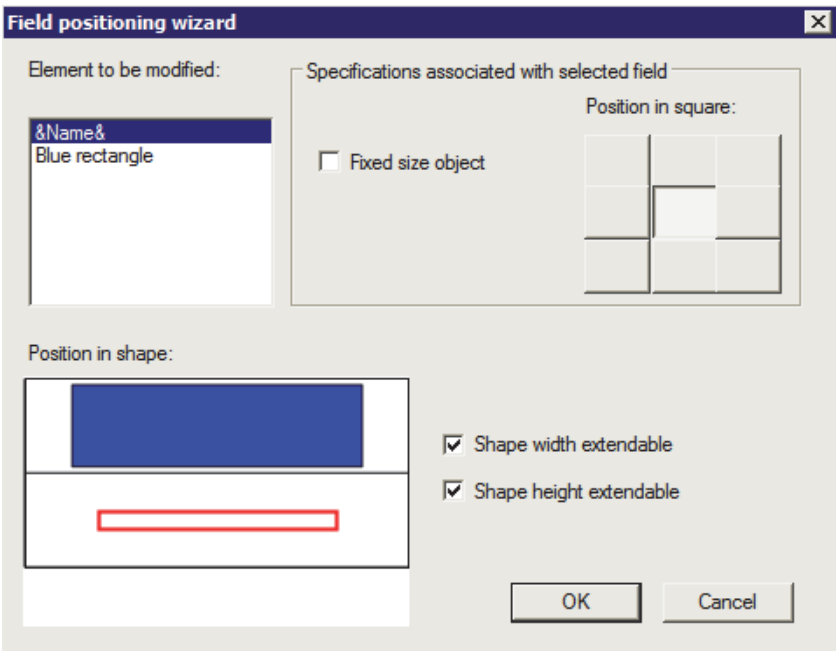
To run the shape code generation wizard:

1. From the **Shapes Editor** menu bar, select **Edit > Field Positioning > Field Positioning Assistant**.
   The **Field positioning wizard** appears.
   All named fields appear in the dialog box. The drawn shape is automatically squared and the different elements of the shape are distributed in the squares.

   For example, for a shape including a blue rectangle and a field &Name&:



   You obtain the following dialog box:



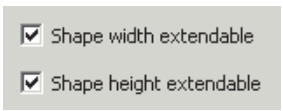
   In the **Element to be modified** pane, the selected element is highlighted in red in the **Position in shape** pane.
   You must correctly position the objects in the **Position in shape** pane.
2. To modify the specifications of selected fields and objects, see:
   - "Shape height and width", page 110
   - "Positioning of the object in its square", page 110
   - "Object deformation", page 111

Shape height and width

By default, height and width of the overall shape are extendable. This means that in the diagram you can stretch the shape so that it exceeds the size of its component objects.

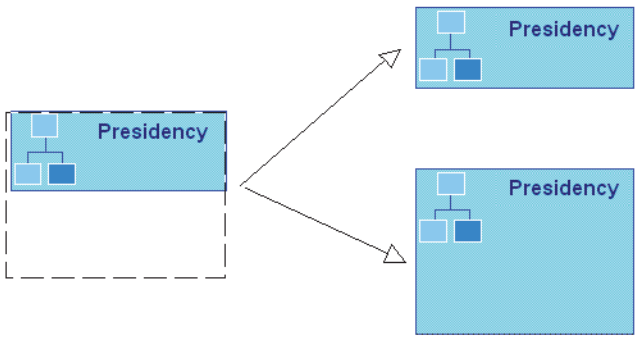
To set height and/or width:

- Clear the corresponding box(es).



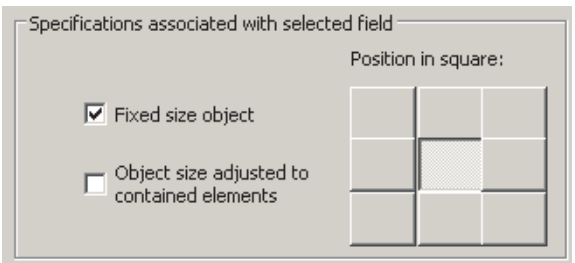
In the following example, the shape is resized to double its height. When the **Shape height extendable** option is:

- ***cleared***: the result is as displayed at ***top*** on right
- ***selected***: the result is as displayed at ***bottom*** on right



Positioning of the object in its square

By default, elements are placed in the center of the square.

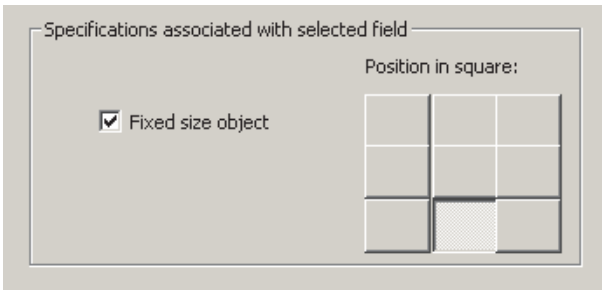


Element placed in center

Elements can be framed vertically and horizontally.
Use of this will be shown in later examples.

Object deformation

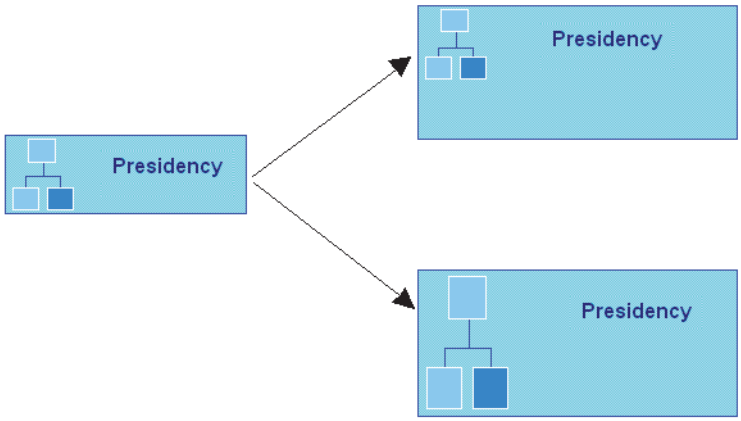
By default, all elements are subject to deformation proportional to that of the shape. It is however possible to set size for certain objects of the shape.



☛ *For calculated fields, the **Fixed size object** option is not taken into account.*

In the following example, height and width of the shape are increased. When the **Fixed size object** option is:

- ***selected***, he result is as displayed at ***top*** on right.
- ***cleared***, he result is as displayed at ***bottom*** on right.



Code Generation

To validate specifications of the shape:

1. In the **Field positioning wizard**, click **OK**.
   The deformation code is automatically generated.

Tips on Deformation Code Generation

Name the fields

Only named fields can intervene in the generated code. It is therefore essential that the different elements of the shape be carefully named. For more details, see "Identifying Elementary Objects", page 108.

Draw accurately

The generator interprets the shape drawn in the shape editor. It considers that two objects are aligned when at least 80% of their width is shared.

MEGA Shapes Programming

A shape is a drawing stored in a file with extension ".MGS". Format of this file is the property of MEGA International. Certain shapes are used to represent methodological concepts used in MEGA diagrams. Others are for purely decorative purposes.

MEGA is a highly customizable platform. This technical article is designed to share knowledge of advanced configuration techniques for deployment or customization of the MEGA platform. Advanced configuration of MEGA will require time and a certain level of expertise with the MEGA platform and other development technologies.

Please be aware that:

- Advanced customizations may require additional development to function correctly after migrating to the next major version of MEGA.
- MEGA Technical Support does not provide assistance with developing, maintaining or upgrading advanced customizations of MEGA.

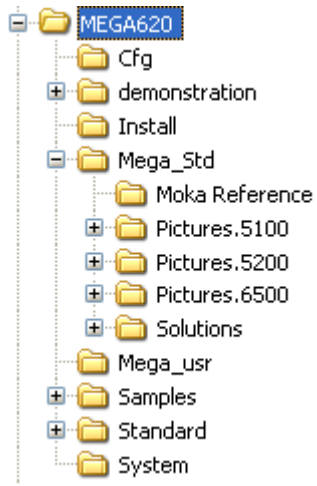
SHAPE STORAGE

From version 6.1, shapes supplied correspond to a given version number. They are stored at site level in directories identified by the MEGA version number with which a new set of shapes has been delivered.

A new diagram always uses the most recent version shapes directory. However, existing diagrams continue to use shapes from the directory of the version with which they were created.

"Method" shapes, "Screen background" shapes and "Decorative" shapes are classified in different directories.

The structure is as follows:

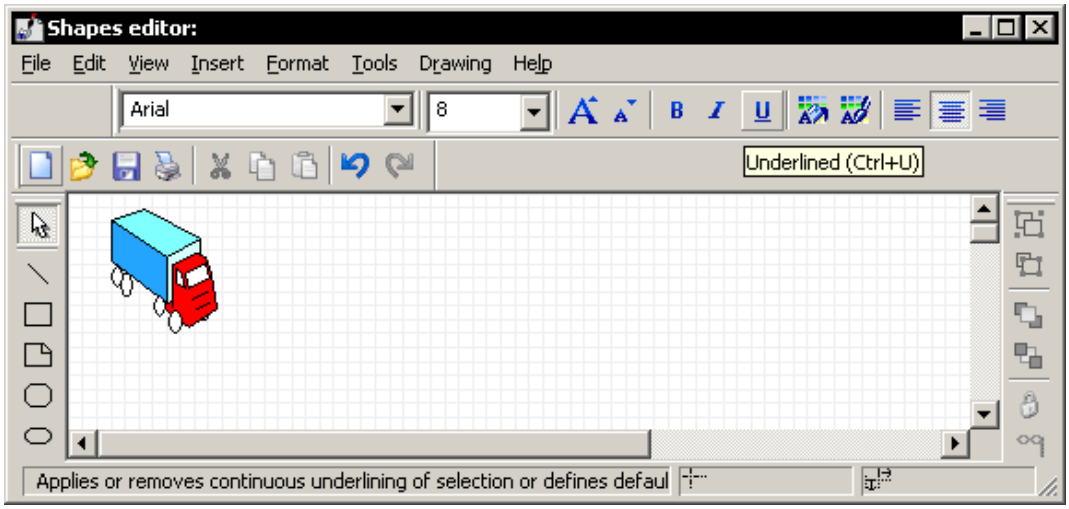


Modification or replacement of these shapes is not recommended since they may be deleted and reinstalled from one version of MEGA to another.

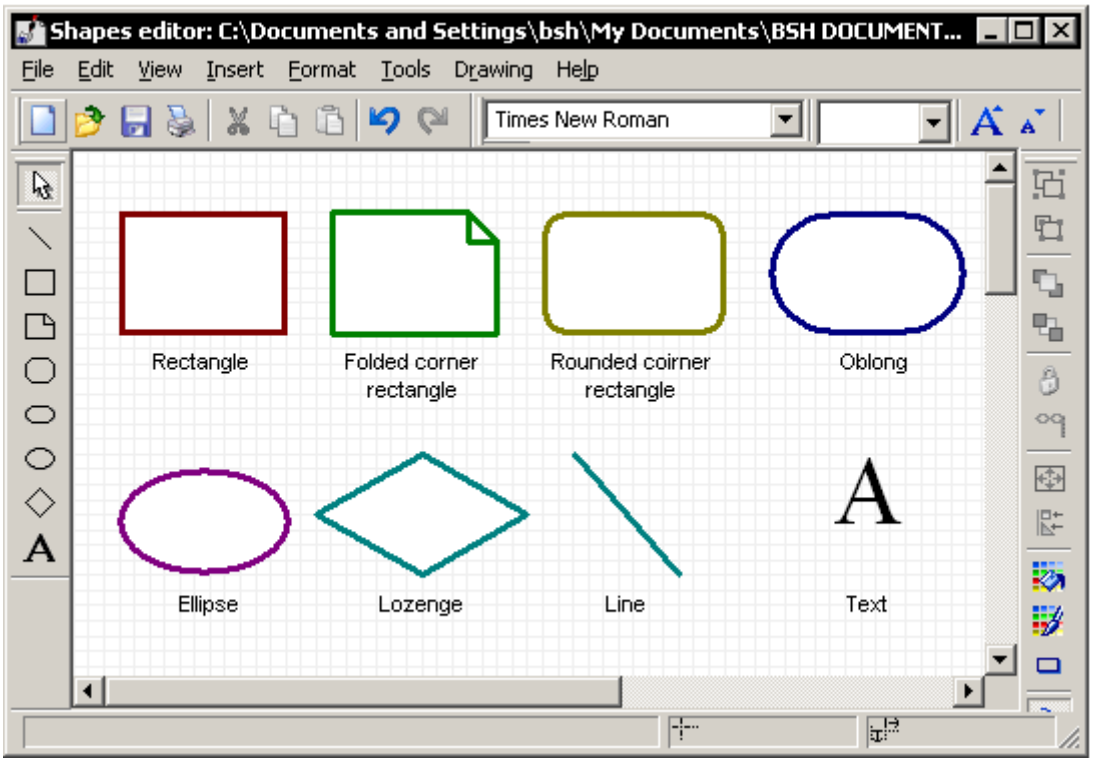
Shapes can however be customized at environment level. To do this, the shape file (.MGS) is copied in the Mega_Usr directory of the environment, and the file "Read-Only" characteristic is cleared.

SHAPE EDITING

The "Shape Editor" enables loading and editing of shapes:



To draw a shape, 8 basic drawing object types are available: line, rectangle, folded corner rectangle, rounded corner rectangle, oblong, ellipse, lozenge, and text, together with images (bmp, wmf, jpeg,..).



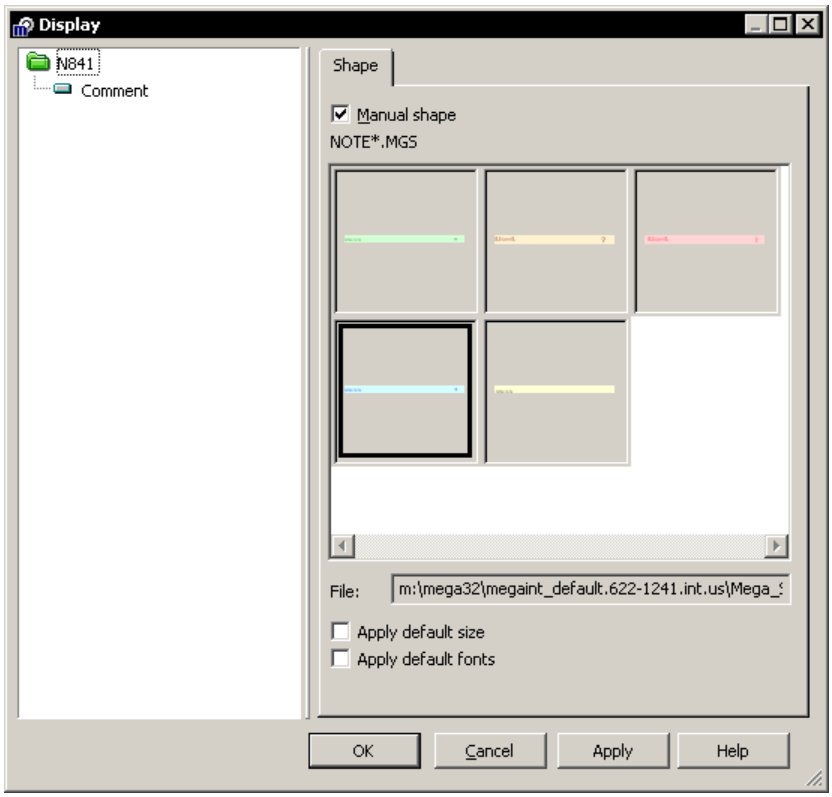
METHOD SHAPES

Method shapes are used to represent methodological concept instances in MEGA Suite diagrams.

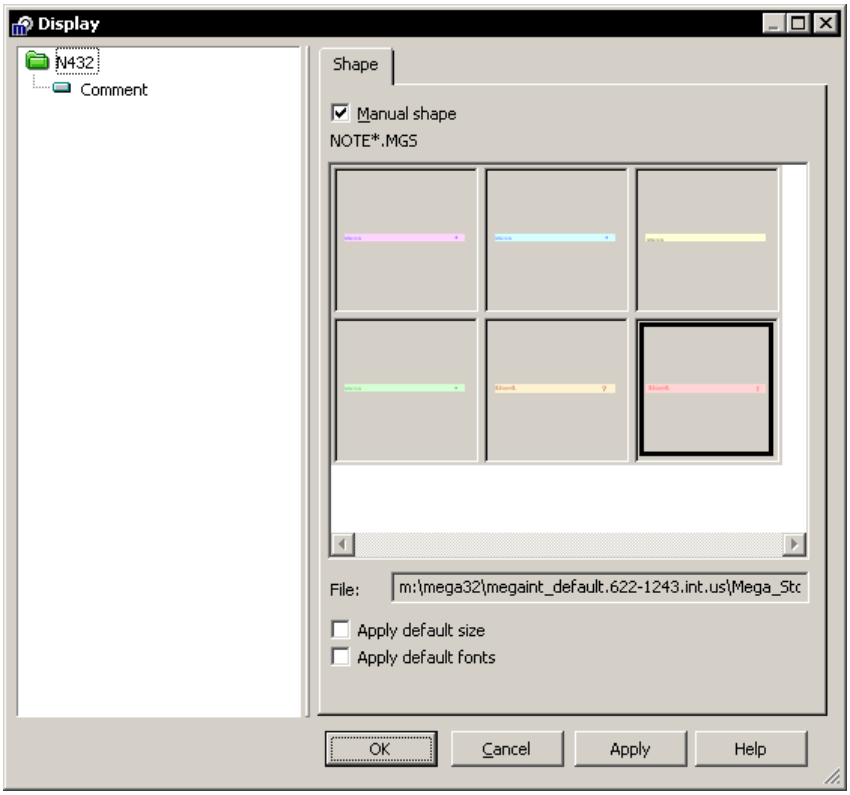
Shape Families

At diagram content definition, each methodological concept is attributed a file name "mask" (eg. *thing**.mgs) enabling grouping under the same prefix of shapes available in this diagram for this concept.

For example, shapes respecting "NOTE*.MGS" format are available for the "Note" concept in diagrams:



As a result, any new shape present in the Mega_Usr directory with the same shape prefix automatically becomes available for this concept:

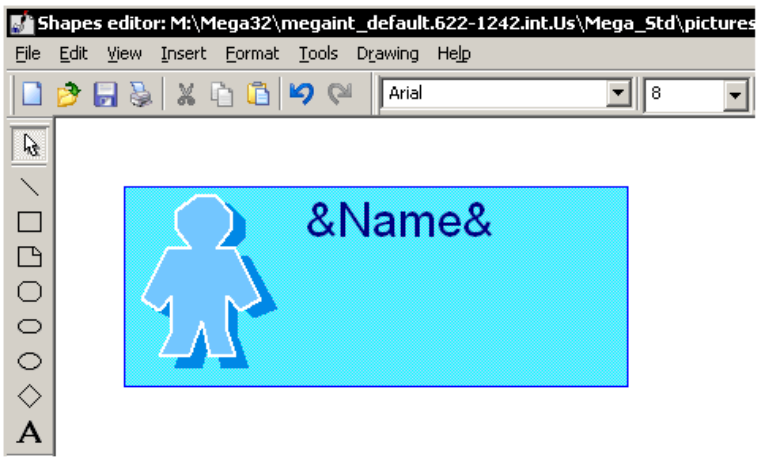


At diagram content definition, shapes to be used by default are also specified.

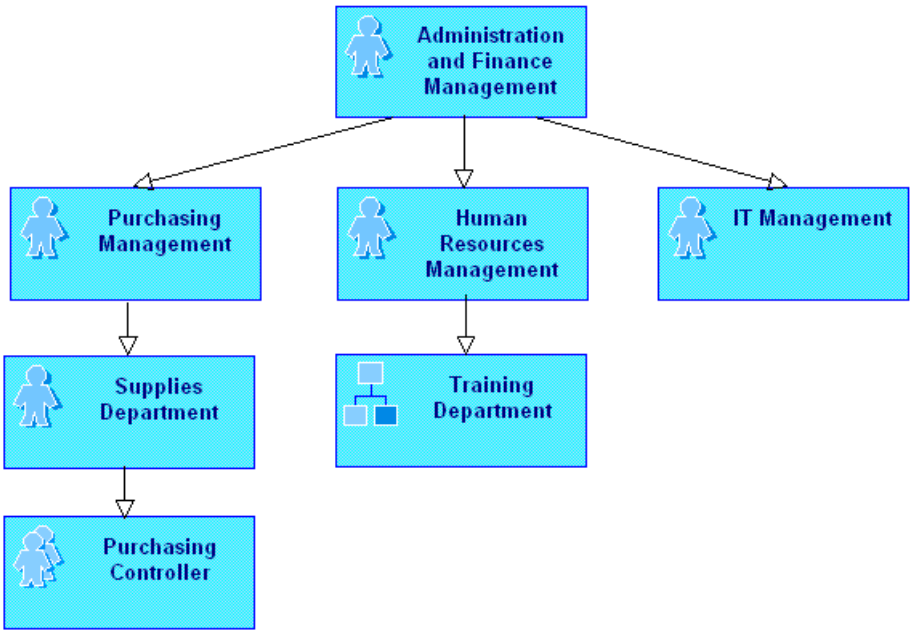
Calculated Fields

In addition to the 8 basic drawing object types, methodological shapes contain particular objects of which content depends on the instance represented. These are "calculated fields".

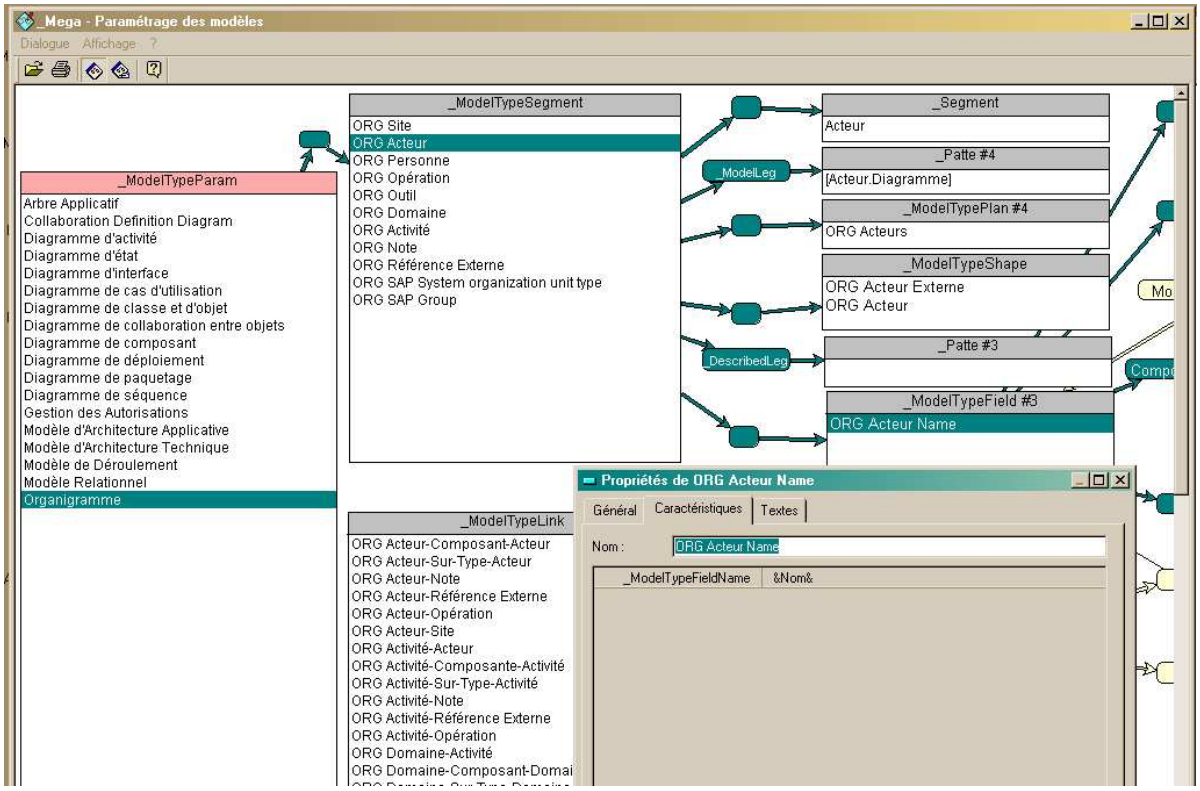
They are represented in shapes by text objects prefixed and suffixed "&":



The real content is dynamically calculated when the diagram is loaded.



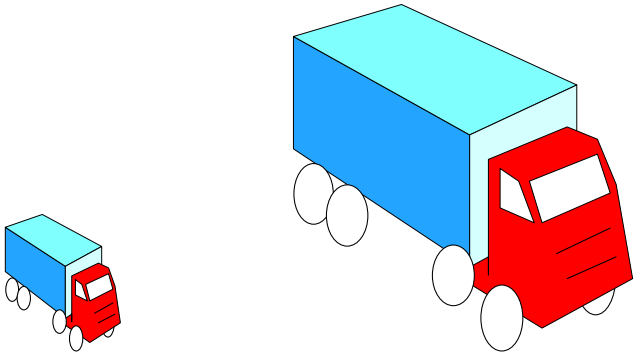
The field calculation mode is specified in diagram content configuration.



DEFAULT DEFORMATION

Proportional Deformation

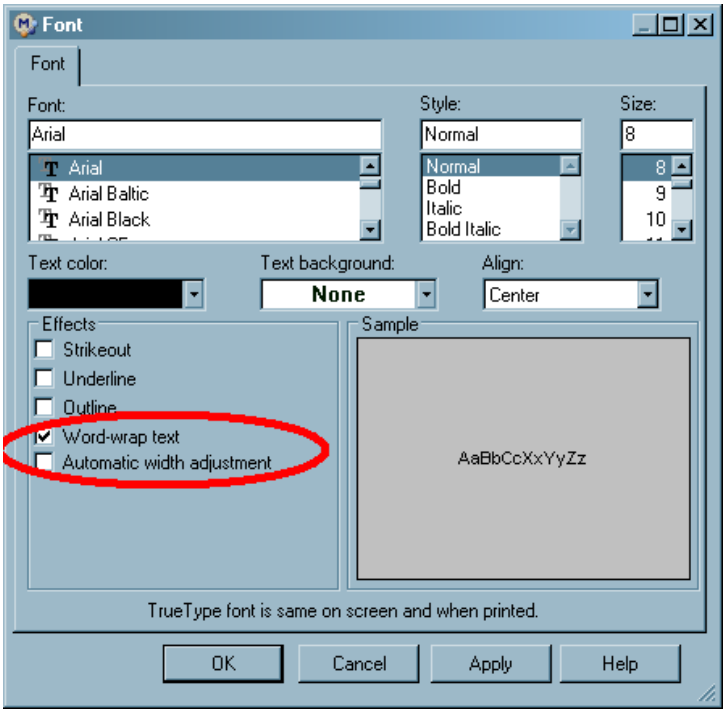
By default, shape deformation conforms to proportional logic. The same reduction or enlargement factor is applied to each object:

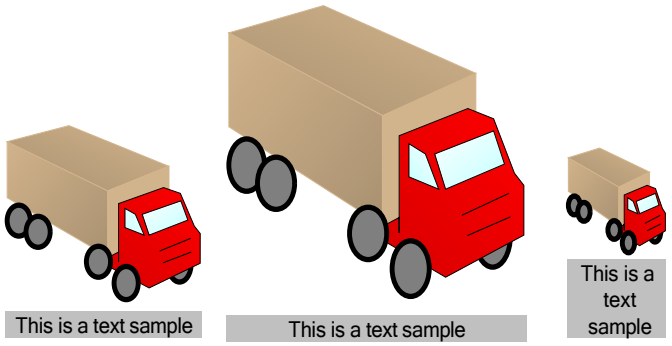
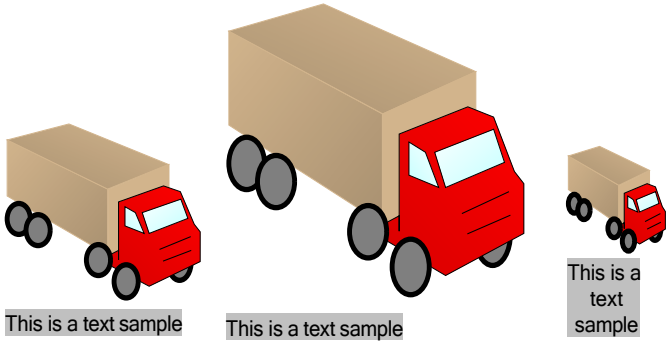
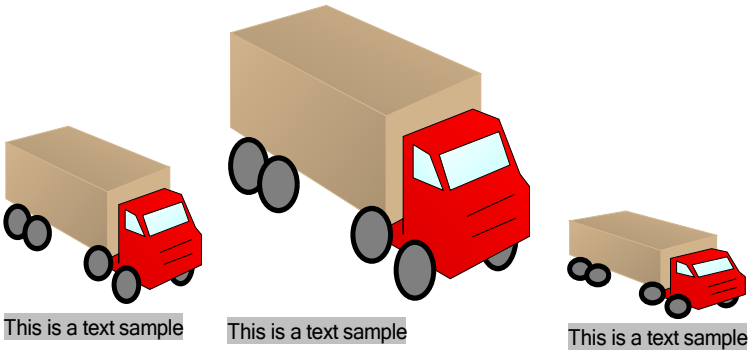
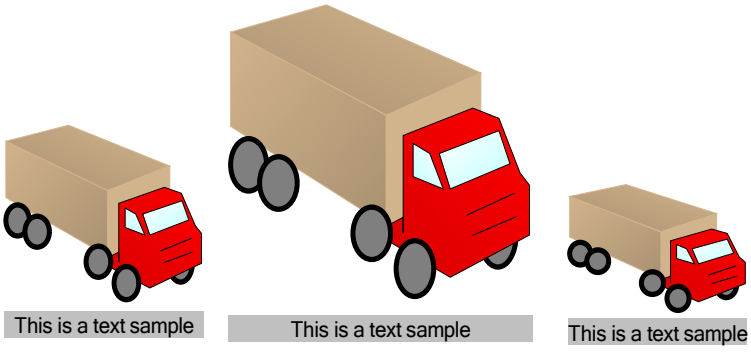


An option allows aspect ratio locking, forcing a shape height and width to be modified by the same factor.

Text Fields Deformation

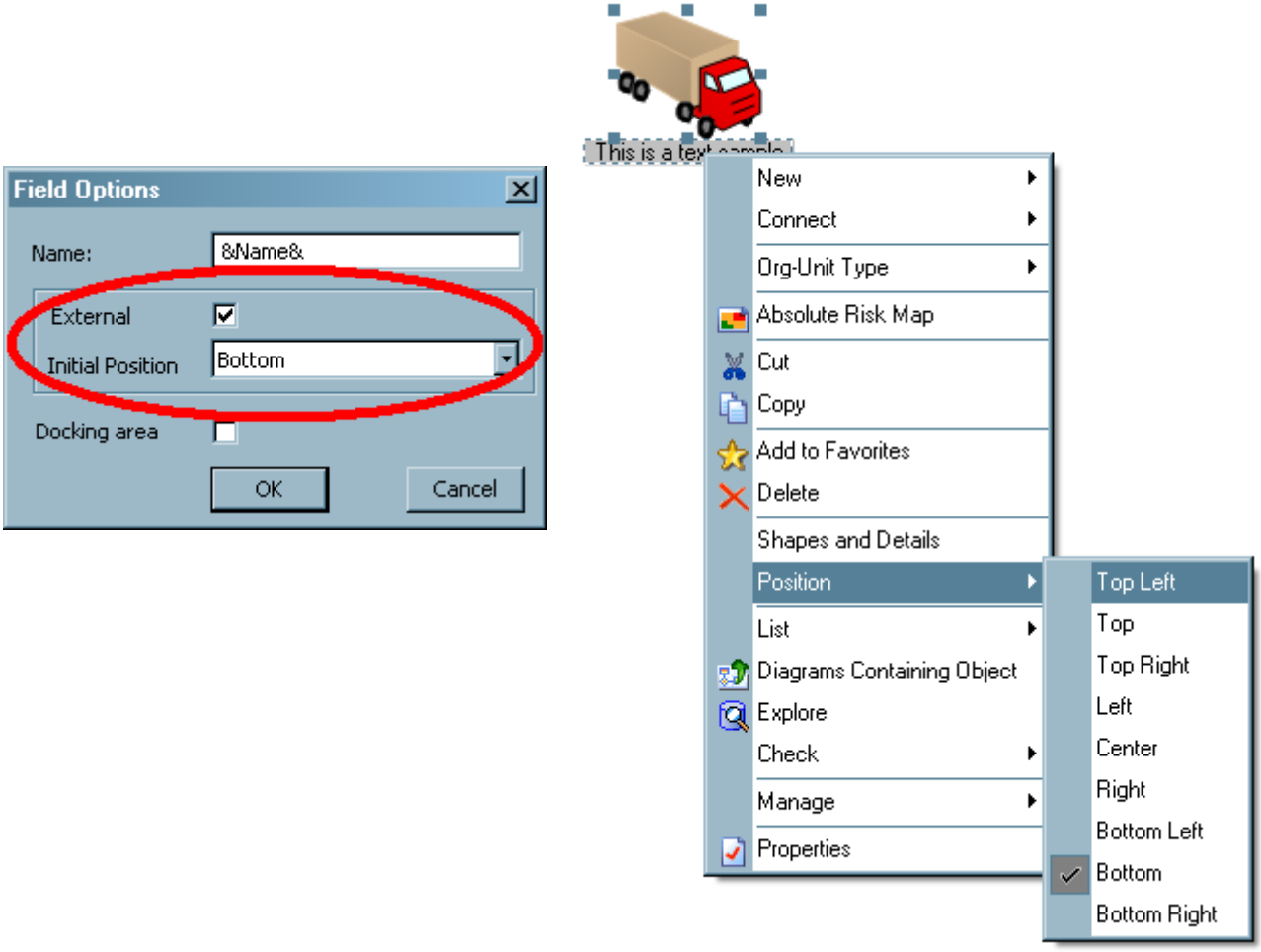
The behavior of text fields can be tweaked by two options.



<table>
<tr><th>Word-wrap text</th><th>Automatic width adjustment</th><th></th></tr>
<tr><td>Yes</td><td>No</td><td rowspan="2"> 
</td></tr>
<tr><td colspan="2">The field width is deformed proportionally while the height is calculated to allow display of the complete text</td></tr>
<tr><td>Yes</td><td>Yes</td><td rowspan="2"> 
</td></tr>
<tr><td colspan="2">The field height is calculated to allow display of the complete text. The field width is then calculated to fit the text width.</td></tr>
<tr><td>No</td><td>Yes</td><td rowspan="2"> 
</td></tr>
<tr><td colspan="2">The text is always displayed on one line. The field width is calculated to fit the displayed text.</td></tr>
<tr><td>No</td><td>No</td><td rowspan="2"> 
</td></tr>
<tr><td colspan="2">The text is always displayed on one line. The field width is deformed proportionally but cannot be smaller than the displayed text.</td></tr>
</table>

External Text Fields

Text fields can be defined to be external to the shape. External fields are ignored during a shape deformation and appear outside the shape frame. The user can modify the position and the width of such fields directly in the diagram.

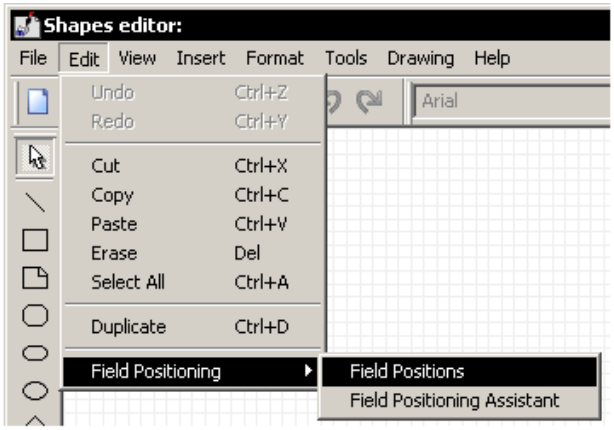


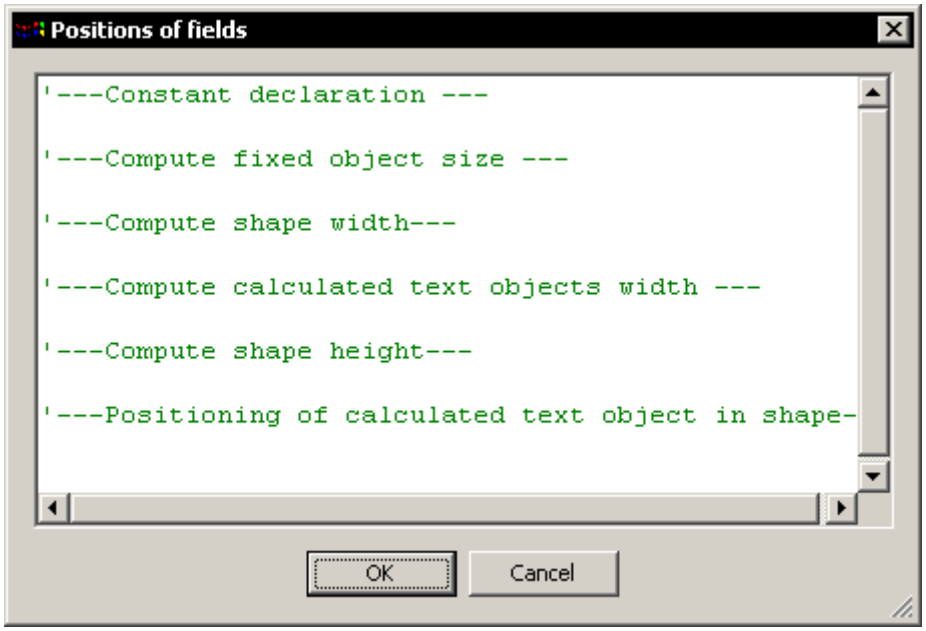
PROGRAMMED DEFORMATION

When default deformation is not suitable, it is possible to program deformation using shape programming language.

Deformation Code Editor

In the shape editor, the "Field positioning" option in the "Edit" menu gives access to a shape deformation program entry space:

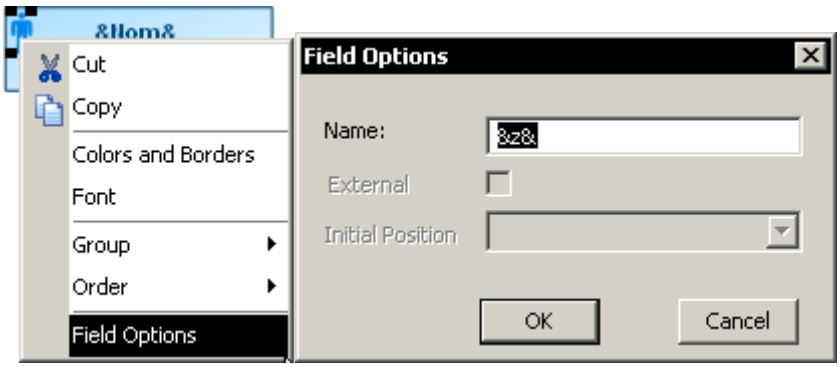




Object Identification

Basic Object Identification

In order to be referenced at programming, each shape object must first be identified by a name. This naming can be via the "Field Options" command in the pop-up menu of each object. The name should be prefixed and suffixed by "&".



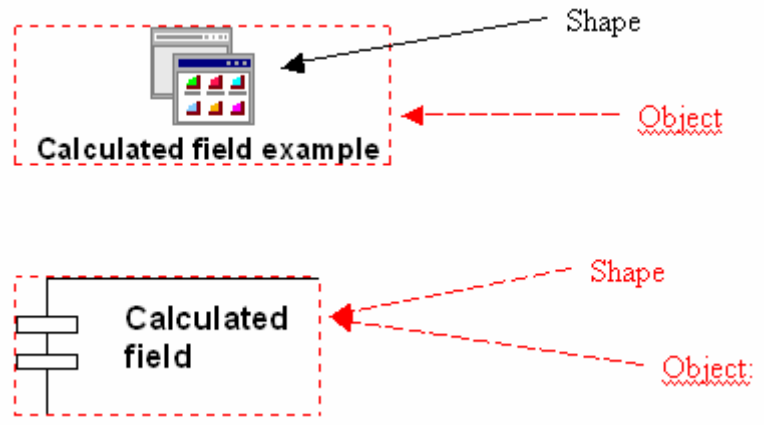
Identifying Calculated Fields

Calculated fields are "self-named". The text they contain can be directly used as the object identifier. (Example: &Name&).

Virtual Object Identification

The language is enhanced by two virtual objects for general use:

- "Shape" object: Rectangle containing graphical elements of the shape except for calculated fields.
- "Object" object: Rectangle containing all graphical elements of the shape including calculated fields.

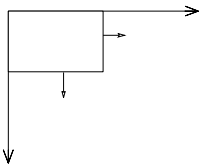


Use of the "Object" object should be limited to cases where we wish to place calculated fields outside the Shape object.

Language Syntax

The value unit is 1/360 inch (14/360 inch ≈ 1mm).

The coordinates system point of origin is the top left of the shape.



Size
width: width
height: height

shape.default.width: Original width (in the MGS) of the shape without calculated fields

shape.default.height: Original height (in the MGS) of the shape without calculated fields

Position

top: top

bottom: bottom

left: left

right: right

hcenter: horizontal median

vcenter: vertical median

Operators

= , + , - , *

max(val1,val2,...,val10) Operand number is limited to 10.

Functions

RelX(*n)* and RelY(*n*), where n is a value, enable definition of constants proportional to shape width and height.

For example, with CNST = RelX(10),

CNST is 10 if shape width is equal to its initial width.

CNST is 15 if shape width is equal to 1.5 times its initial width.

Comment

' at start of line

To access these properties, syntax is as follows:

- For calculated fields: *object name without &s* . *property* (eg. Att.width)
- For the "Shape" object: Shape.*property* (eg. Shape.hCenter)
- For the "Object" object: Object.*property* (eg. Object.top)
- For other named objects: Shape.*object name with &s*.property (eg. Shape.&Logo&.bottom)

Programming

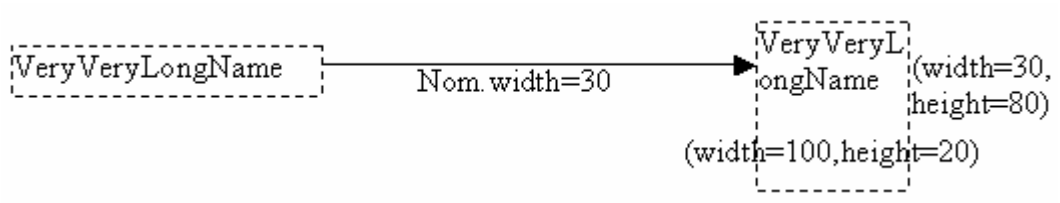
The shape deformation program is applied each time the user resizes an object based on this shape, or when one of its calculated fields changes value.

Program lines are executed singly and sequentially.

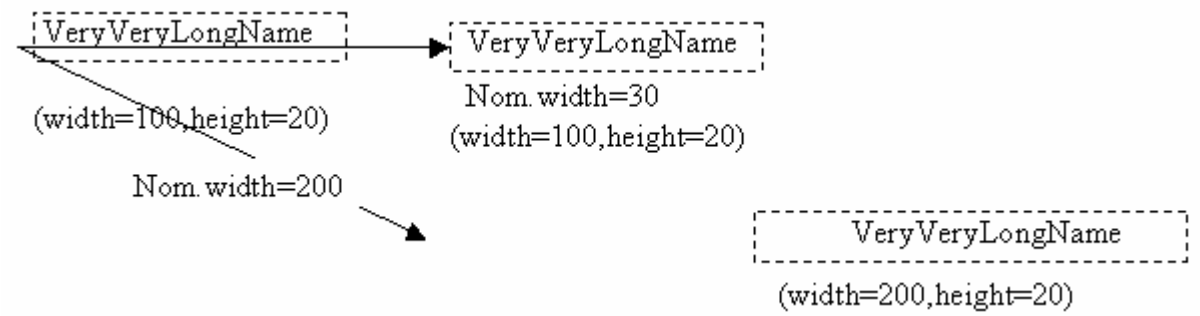
Line order is therefore of major importance, and any inversion of these lines can cause quite unexpected deformation behavior.

Calculated Field Programming

If "Word-wrap text" is active, any modification of field width by code will result in immediate recalculation of its height:



If "Word-wrap text" is inactive, the calculated field should be sufficiently wide to display the text on a single line. If in programming we attempt to assign too narrow a width, an adequate minimum width is authoritatively reassigned.

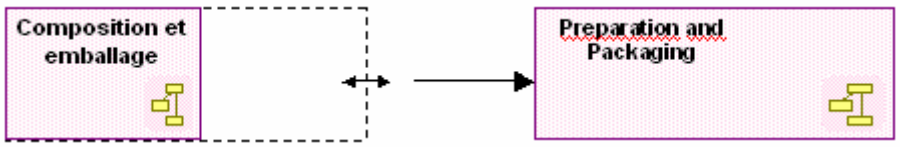


Modification of field height is never taken into account. It is always automatically recalculated as a function of field width.

General Programming Recommendations

Deformation code is called in 2 cases:

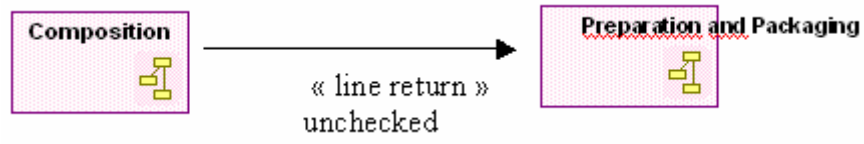
When we change size of a concept based on this shape:



Before code application

Here the code should handle text resizing.

The content of a calculated field is changed:



Before code application

Here the code should handle shape resizing.

To limit side effects due to incorrect programming, the following steps are recommended:

- Fix the size of fixed size objects.
- Fix the final width of the shape as a function of the width of fixed size objects, calculated fields and the width required for the shape (shape width before code application, see example below). *(NB1)*
- Fix the final width of the calculated fields as a function of the final width of the shape. *(NB2)*
- Fix the final height of the shape as a function of the height of fixed size objects, calculated fields and the height required for the shape. *(NB2)*
- Fix the positions of objects within the shape

NB1: Before deformation code execution, the width of calculated fields with word-wrap active is 1mm (authorized minimum), and the width of calculated fields with word-wrap inactive is equal to the effective value necessary for visibility of content. **The width of all calculated fields of the shape must therefore be defined.**

NB2: Since shape height depends on height of calculated fields, and height of calculated fields automatically depends on their width, it is essential to fix the width of calculated fields before fixing shape height.

Default Code

A shape without a code will not have the same deformation behavior as a shape with a code performing no function (no modification of shape objects properties). A default code is applied to shapes without code.

For each calculated field, the following instruction block is inserted by default in the code.

Field refers to the calculated field.

The following constants are calculated compared with the initial state of the shape (the state that can be seen in the shape editor)

FieldWidthInit: width of calculated field

FieldTopInit: vertical position of calculated field

FieldLeftInit: horizontal position of calculated field

BLOCK:

```
Shape.width = max( Shape.width , Field.width )
Field.width   = RelX ( FieldWidthInit )
Shape.width = max( Shape.width , Field.width )
Field.top      = Shape.top + RelY ( FieldTopInit )
Field.top      = Shape.top + RelY ( FieldTopInit )
```

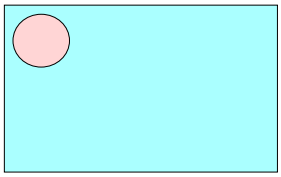
This code enables the calculated field to maintain the same relative position within the shape.

In programming a shape, position and size of all calculated fields of the shape must be managed.

EXAMPLES

Shape With Fixed Size Object

The aim is to retain the size and position of circle &Circle&, whatever the size of the shape.



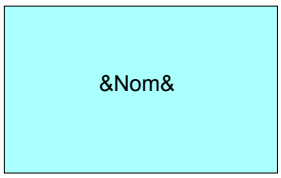
To fix dimensions of the fixed size object:

```
Shape.&Circle&.width=140

Shape.&Circle&.height=140
```

To freeze its position must be added:

```
Shape.&Circle&.top = Shape.top + 20

Shape.&Circle&.left = Shape.left + 20
```

Shape With Calculated Field




```
'---Constant---
CX_SPACE=10
CY_SPACE=10

'---Shape width--- (at minimum the width required to display text)
Shape.width=Max(Shape.width, Nom.width +2*CX_SPACE)

'---Fields width--- (field size is readapted to shape size)
Nom.width=Shape.width-2*CX_SPACE

'---Shape width--- (at minimum the height required to display text)
Shape.height=Max(Shape.height, Nom.height +2*CY_SPACE)

'---Objects Position---
Nom.hCenter = Shape.hCenter
Nom.vCenter = Shape.vCenter
```

Shape With Pictogram

The pictogram should remain at bottom right and at fixed size.

```
'---Constant---
CX_SPACE = 10
CY_SPACE = 10
```

```
'---Fixed size object---
Shape.&Pict&.width  = 2
Shape.&Pict&.height = 1

'--- Shape width ---
Shape.width = Max ( Shape.width, Max( Nom.width, Shape.&Pict&.width) +
2*CX_SPACE)

'--- Fields width ---
Nom.width = Shape.width - 2 * CX_SPACE

'--- Shape height ---
Shape.height = Max ( Shape.height, Nom.height + Shape.&Pict&.height + 3 *
CY_SPACE)

'---Objects Position---
Nom.left =  Shape.left + CX_SPACE
Nom.top =  Shape.top + CY_SPACE

Shape.&Pict&.right  = Shape.right - CX_SPACE
Shape.&Pict&.bottom = Shape.bottom - CY_SPACE
```

Shape With Calculated Field and Pictogram Juxtaposed

The pictogram should remain at bottom right and at fixed size.

The calculated field &Duration& and the pictogram should not overlap.

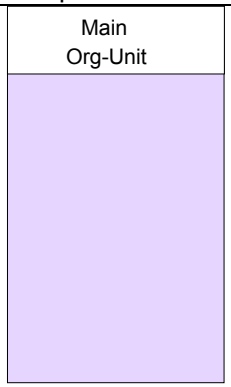
```
'---Constant---
CX_SPACE = 10
CY_SPACE = 10

'---Graphics size---
Shape.&Pict&.width  = 70
Shape.&Pict&.height = 35

'--- Shape width ---
Shape.width = max  ( shape.width, Nom.width + 2*CX_SPACE ,
                                  Duration.width + Shape.&Pict&.width +
3*CX_SPACE)

'--- Fields width ---
Nom.width   = Shape.width - 2 * CX_SPACE
Duration.width = Shape.width - Shape.&Pict&.width - 3*CX_SPACE

'--- Shape height ---
Shape.height =max(Shape.Height , Nom.Height+shape.&Pict&.height
+3*CY_SPACE, Nom.Height + Duration.height + 3*CY_SPACE )

'---Objects Position---
Nom.left    =  Shape.left + CX_SPACE
Nom.top     =  Shape.top + CY_SPACE

Shape.&Pict&.right  = Shape.right - CX_SPACE
Shape.&Pict&.bottom = Shape.bottom - CY_SPACE

Duration.left = Shape.left + CX_SPACE
Duration.bottom = Shape.bottom - CY_SPACE
```

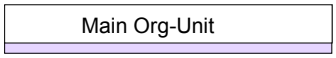
Column Shape

The white rectangle is named &TitleRect&, it should frame the name and retain its fixed size when the shape is elongated downwards. (shape type used to represent org-units in flowcharts)

```
'---Constant---
CX_SPACE=10
CY_SPACE=10

'--- Shape width ---
Shape.width=Max(Shape.width,Nom.width+2*CX_SPACE)

'--- Fields width ---
Nom.width=Shape.width-2*CX_SPACE

'--- Shape height ---
Shape.height=Max(Shape.height,Nom.height+4*CY_SPACE)

'--- Graphics Size---
```

```
Shape.&TitleRect&.height=Nom.height+2*CY_SPACE
Shape.&TitleRect&.width=Shape.width

'---Objects Position---
Shape.&TitleRect&.left=Shape.left
Shape.&TitleRect&.top=Shape.top
Nom.hCenter=Shape.&TitleRect&.hCenter
Nom.vCenter=Shape.&TitleRect&.vCenter
```





Shape With Title Below

In this example, the calculated field is outside the shape. This is a use case of the "Object" virtual object.

&Nom&

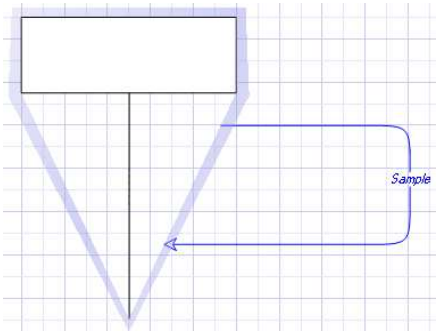
```
'---Constant---
CY_SPACE = 10

'---Shape size---
Shape.width = Shape.default.width
Shape.height = Shape.default.height

'---Object width---
Object.width = Max(Nom.width, Object.width, Shape.width)

'--- Fields width ---
Nom.width = Object.width

'---Objects Position---
Nom.left = Object.left
Nom.top = Shape.bottom + CY_SPACE
Shape.top = Object.top
Shape.hCenter = Object.hCenter
```

Here we must also place the shape object within the Object object.

Note: Object is recalculated after shape code application (smallest rectangle containing calculated fields and Shape).

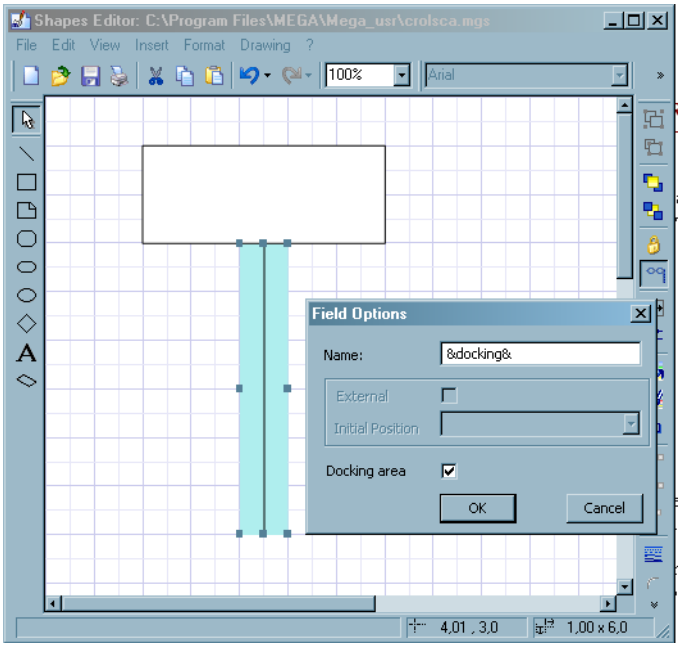
DOCKING AREA

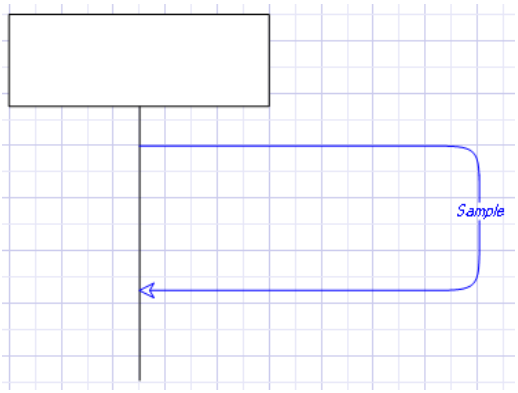
MEGA automatically computes a convex hull for each shape. This area is used to determine the intersections between the shapes and links connected to it. However, for some shapes, this area is not adequate.



To improve the behavior of the shape regarding links, it is possible to define a custom docking area. With such an area defined, links will start and end only on the edge of this area.

To define a custom docking area, draw a basic graphic object and check the appropriate option in the "field options" dialog box.





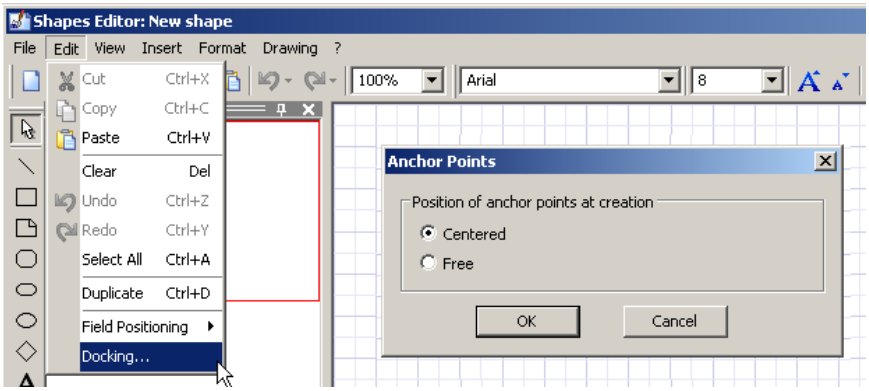
To further improve the behavior of the shape, this field can be positioned like any other field with the shape programming language. To further improve the look of the shape, this field can be hidden in the shape editor.

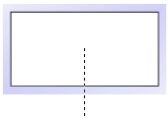
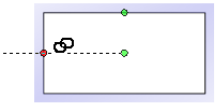
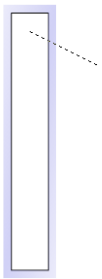
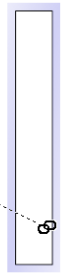
However, only one docking area is allowed, and it must be convex.

POSITION OF ANCHOR POINT AT CREATION

When connecting two objects in the diagram editor with a link, there are two different behaviors which will position the anchor point on the shape.

The behavior to be used with a shape can be set in the shape editor.



	From	To
Centered The anchor point is centered in the shape		 Targeting the green dots will change the style (orthogonal/straight) and orientation of the link
Free The anchor point is positioned where the mouse is clicked or released The default line style is used		

2

CONFIGURING DIAGRAMS

After having defined the metamodel extensions you require, you need to display these in diagrams.

MEGA Studio allows you to carry out multiple tasks such as creation or modification of diagram types, addition of MetaClasses and MetaAssociations available in the diagram type, definition of default display of MetaClasses, object shapes in a diagram type, association of a diagram type with one or several MetaClasses.

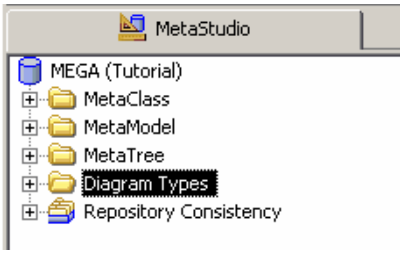
This chapter covers the following points:

DIAGRAM TYPES

Accessing Diagram Types

To access diagram types:

- From **MEGA** menu bar, select **View > Navigation Windows > MetaStudio**.
  The diagram types are grouped in the **Diagram Types** folder.



Creating a Diagram Type

To create a new diagram type:

1. In the **MetaStudio** navigation window, right-click the **Diagram Types** folder and select **New > Diagram Type**.
2. In the dialog box that opens, enter the name of the diagram type and click **OK**.

Diagram types can be grouped by category - these categories are presented as folders of diagram types.

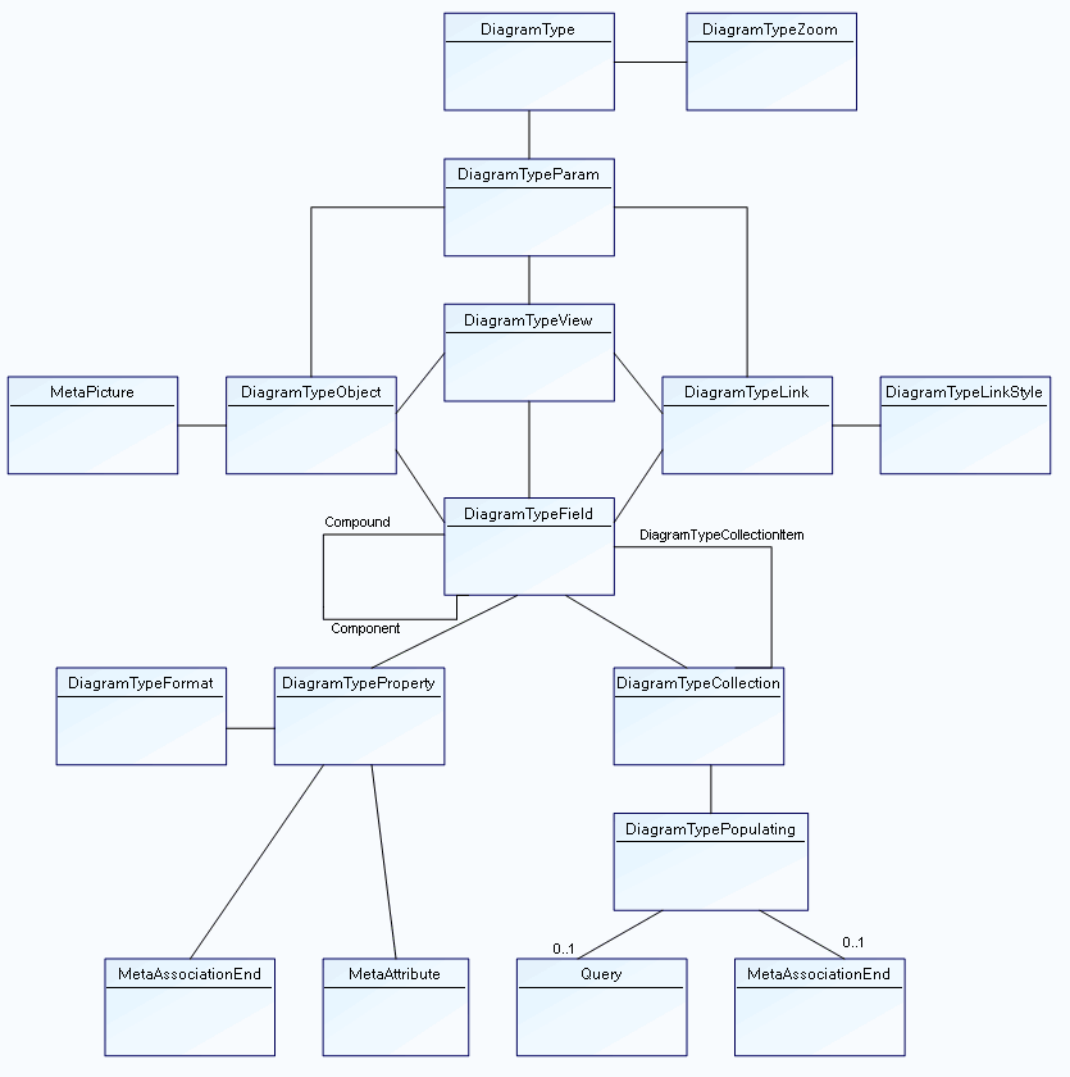
To create a new folder of diagram types:

1. In the **MetaStudio** navigation window, right-click the **Diagram Types** folder and select **New > Folder of Diagram Types**.
2. In the **Diagram Types Folder Creation** dialog box, enter the folder name and click **OK**.

CONFIGURING A DIAGRAM TYPE

Concepts

Diagram type configuration uses concepts presented in the following metamodel.



A ***DiagramType*** can describe a specific number of concepts.

A ***DiagramTypeZoom*** enables a MetaClass to be described by a diagram type (example: a procedure can be described by a flowchart).

Configurations of a diagram type are carried by a ***DiagramTypeParam***.
Diagram type configuration is based on the following concepts:

- ***DiagramTypeObject***: object types (MetaClasses) that can be used in a diagram type.
- ***DiagramTypeView***: views available in a diagram type. A view groups one or several object types that can be used in a given diagram type.
- ***DiagramTypeLink***: links that can be displayed in a diagram type.
- ***DiagramTypeField***: fields that can appear in an object shape or link.

The different concepts enabling definition of graphical representation of objects, links, views and fields (in graphical representation of objects and links).

Concept names: equivalence table

The names of concepts enabling diagram configuration management have changed in version **MEGA 2007**. The table below shows correspondence between old and new concept names.

Old Name	New Name	Definition
_ModelType	DiagramType	Diagram Type
_ModelTypeCollection	DiagramTypeCollection	Configuration of an attribute repeated n times in a diagram configuration
_ModelTypeField	DiagramTypeField	Representation of a field displaying a list of properties or a basic property
_ModelTypeFormat	DiagramTypeFormat	Defines a format or string that can change according to a specific condition
_ModelTypeLink	DiagramTypeLink:	Representation of a MetaAssociation in a diagram type
_ModelTypeLinkStyle	DiagramTypeLinkStyle	Defines a link style that can change according to a specific condition
_ModelTypeParam	DiagramTypeParam	Configuration of a diagram type
_ModelTypePath	DiagramTypePath	Representation of a path (eg. MetaAssociation) in a diagram type
_ModelTypePathPart	DiagramTypePathPart	Configuration of a DiagramTypePath element
_ModelTypePlan	DiagramTypeView	Configuration of a view and its content

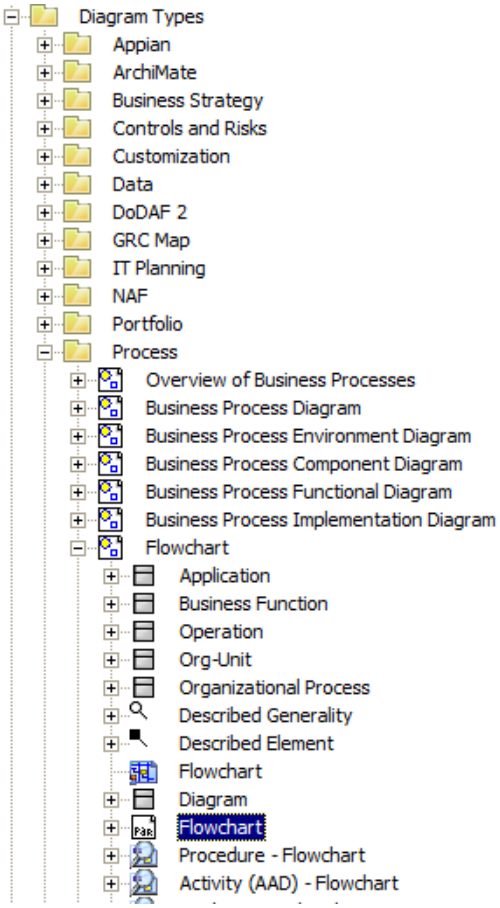
Old Name	New Name	Definition
_ModelTypeProperty	DiagramTypeProperty	Configuration of a basic attribute in a diagram configuration
_ModelTypeSegment	DiagramTypeObject	Representation of an object type in a diagram type
_ModelTypeSelection	DiagramTypePopulating	Defines the method (MetaAssociationEnd, Query) of populating a collection specified by DiagramTypeCollection
_ModelTypeZoom	DiagramTypeZoom	A DiagramTypeZoom enables a MetaClass to be described by a diagram type.

Configuration Example: flowchart

We shall illustrate diagram type configuration by considering the example of an existing configuration, that of the flowchart.

To access flowchart configuration:

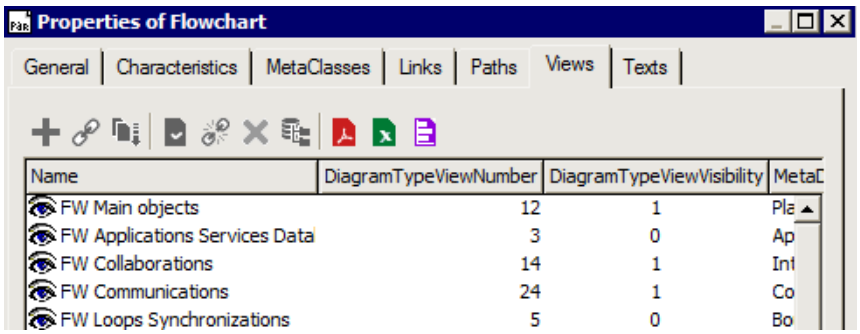
1. In the **MetaStudio** navigation window, successively expand the **Diagram Types** and **Process** folders, then the **Flowchart** folder.
   In the **Flowchart** folder he **Flowchart** DiagramTypeParam corresponds to the object of which the icon contains characters PAR.



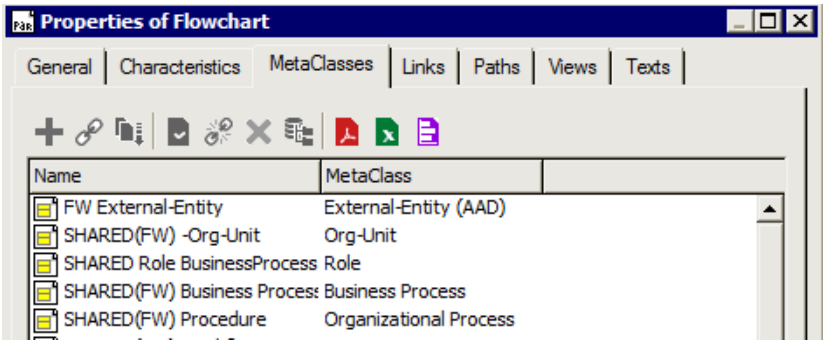
2. To view the flowchart configuration elements, expand the **Flowchart** DiagramTypeParam folder. It includes the list of:
   - ***DiagramTypeViews***: views.
   - ***DiagramTypeObjects***: MetaClasses that can be used in flowcharts. A DiagramTypeView can contain one or several DiagramTypeObjects.
   - ***DiagramTypeLinks***: Enables description of representation of a MetaAssociation (link) in a diagram.

From the DiagramTypeParam Properties window, you can access:

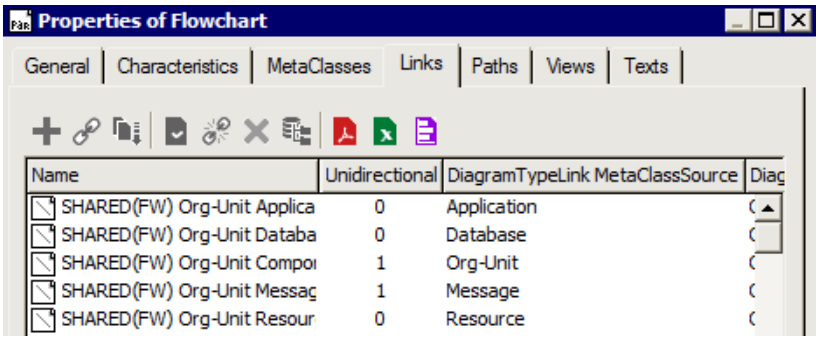
- DiagramTypeViews from the **Views** tab



- DiagramTypeObjects from the **MetaClasses** tab



- links (DiagramTypeLinks) from the **Links** tab

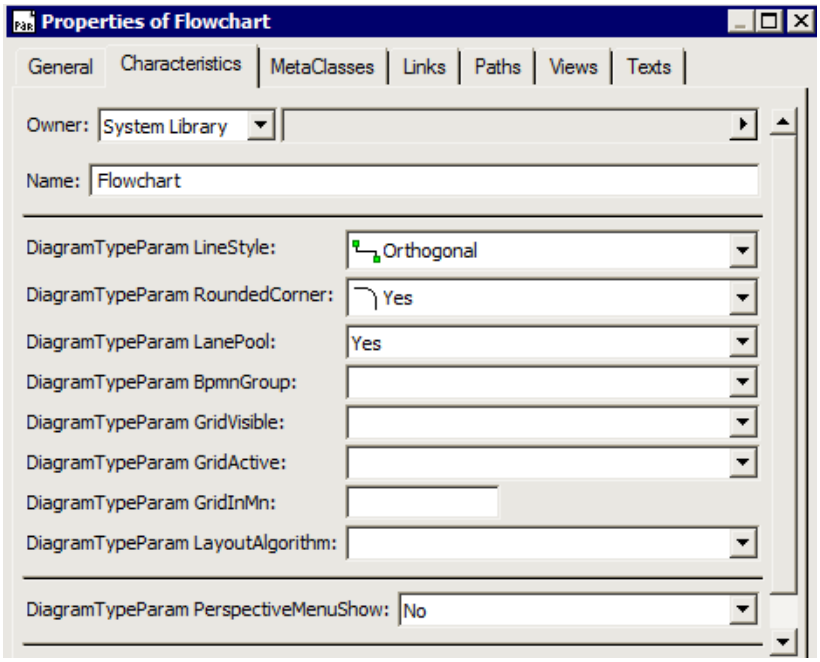


From these tabs you can create new objects and consult object properties.

In the DiagramTypeParam Properties window you can define some general properties concerning display in the flowchart.

To access these properties:

3. Access the DiagramTypeParam Properties and select the **Characteristics** tab.



4. You can configure:
   - line style
   - corner style (rounded or not)
   - use of swimlane pools
   - use of the grid

Creating and Backing Up a Diagram Configuration

Configuration of diagrams is not assured by **MEGA**, unlike standard customizations (metamodel extensions, shape customizations). It is an advanced configuration that requires:

- validation of configurations by compiling the metamodel in **MEGA Supervisor**
- configuration backup
- repeat of the previous steps after each **MEGA** version update. Diagram configurations are reinitialized at environment updates

Given the complexity of the diagram configuration procedure, configurations should be carried out in an environment other than the production environment.

The simplest method of backing up a diagram configuration is to export it.

```
Example: if you have modified organizational chart
configuration, you should export the "Organizational Chart"
DiagramTypeParam.
```

To apply your configuration to a diagram type after updating the environment:

1. In **MEGA Supervisor**, connect to the environment concerned with a user having rights to modify **MEGA** data.
2. Import the exported file containing your customized configurations.
3. Connect the DiagramTypeParam to the corresponding DiagramType.
4. Compile the metamodel.
5. Copy the image files used by your configuration in the Mega_std folder of the **MEGA** installation.

To view during a transaction the result of changes in configuration of diagrams and navigation trees:

1. In the **MEGA** Start page, click **MetaStudio Console**.
2. In the dialog box that opens, click **Refresh Context**.

To obtain the same result, you can open the script editor, enter the command ***CurrentEnvironment.refreshcontext*** and click **Execute**.

Using a New MetaClass in a Diagram

If you consider it necessary to create a new MetaClass, you will certainly need to use it in one or several diagrams.

To do this, you must first connect the new MetaClass to the Diagram MetaClass.

You must then create a new diagram type view and associate with this view the new MetaClass you have created. Alternatively, you can connect the new MetaClass to an existing view.

☛ *A view groups one or several object types that can be used in a given diagram type.*

You then have to assign an image (shape) to the new MetaClass. This shape will represent occurrences of the MetaClass in this diagram type.

The object shape contains a field used to display an attribute (normally the object name). This field can also be configured.

Finally, you can create and configure links that can be displayed in the diagram type concerned.

Creating a view in a diagram type

Before creating a new view, in the metamodel diagram connect your new MetaClass to the Diagram MetaClass.

☛ *For more details, see "The Metamodel Diagram", page 25.*

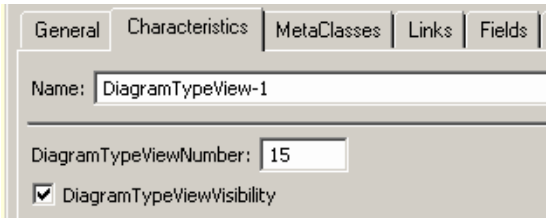
To create a view in a diagram type:

1. In the **MetaStudio** navigation window, expand the **Diagram Types** folder.
2. Expand the folder of the diagram type concerned.
3. Right-click the ***DiagramTypeParam*** concerned and select **Properties**.
4. In the **Views** tab, click **New**.
   A new view is created.

In the **Characteristics** tab of the view Properties:

1. In the **DiagramTypeViewNumber** field enter a value (between 1 and 30 inclusive).

2. Select **DiagramTypeViewVisibility** if you want this view to be visible by default in the diagram.



Adding a MetaClass to a view

A MetaClass that can be used in a diagram (DiagramTypeObject) must be associated with a view (DiagramTypeView).

To add an object type (DiagramTypeObject) to a view:

1. Right-click the view and select **Properties**.
2. In the **MetaClasses** tab, click **New**.
3. In the **MetaClass** field, click the arrow and select **Connect MetaClass**.
4. From the **Query** tool select the MetaClass that interests you and click **OK**.
   The new DiagramTypeObject is now connected to the MetaClass you selected.
   A new DiagramTypeObject is created. It appears in the list.

All DiagramTypeObjects must be connected to the Diagram MetaClass by the DiagramTypeLinkEnd MetaAssociationEnd.

Defining a shape for the new MetaClass

To define the shape that will be used for the new MetaClass:

1. In the DiagramTypeObject Properties window, select the **Characteristics** tab.
2. In the **DiagramTypeShape** box, enter the file mask to be used to graphically represent the object.
3. In the **MetaPicture** tab, click **Connect** to connect the image that interests you.

Defining a field in the new shape

The new shape you have created contains at least one field - that which enables display of the object name.

To define this field:

1. In the DiagramTypeObject Properties window, select the **Fields** tab.
2. Click **New** to create a new field (DiagramTypeField).
3. Open the DiagramTypeField Properties window and select the **Characteristics** tab.

4. In the **DiagramTypeField Name** box, specify the **&name&** value representing the name of the variable defined in the shape supplied by **MEGA**.
5. In the **Properties** tab, click **New** to create a new DiagramTypeProperty. Give it the same name as the MetaPicture.
6. In the DiagramTypeProperty Properties window, **Characteristics** tab, select the MetaAttribute to be displayed (in this case, **Name**).
7. Close all dialog boxes by clicking **OK**.

Configuring MetaAssociations

Creating a MetaAssociation

A DiagramTypeLink enables representation of a MetaAssociation (link) in a diagram. You cannot draw a link between between two objects in a diagram if the corresponding MetaAssociation does not already exist in the metamodel.

To create a DiagramtypeLink that could be displayed in the diagram type that interests you:

1. In the DiagramTypeParam Properties window, select the **Links** tab.
2. Click **New** to create a DiagramTypeLink.
3. In the DiagramTypeLink Properties window (**Characteristics** tab), click the arrow at the right of the **MetaAssociationEnd** box and select **Query MetaAssociationEnd**.
4. In the dialog box that appears, use the query wizard to find the MetaAssociationEnd that interests you.
5. In the DiagramtypeLink Properties window, select the **Unidirectional** check box if you want the MetaAssociation to have only one valid direction.

Configuring link display

Display of links in a diagram can be customized using a series of attributes accessible in the MetaAssociation Properties window (DiagramTypeLink).

To access configuration of links in a diagram:

1. In the **MetaStudio** navigation window, open the DiagramTypeParam Properties window on which the link concerned depends.
2. The DiagramTypeParam Properties window opens.
3. From the **Links** tab, right-click the desired link and select **Properties**.
4. In the link Properties window, select the **Characteristics** tab.

5. From the **Characteristics** tab you can configure the following display characteristics:
   - **DiagramTypeLink LineStyle**: link line style
   - **DiagramTypeLink LineBeginStyle**: default arrow style on first link end.
   - **DiagramTypeLink LineEndStyle**: default arrow style on second link end.
   - **DiagramTypeLink LineDouble**: specifies default link style as double line.
   - **DiagramTypeLink RoundedCorner**: specifies default line style as rounded angles.
   - **DiagramTypeLink PenColor**: default line color.
   - **DiagramTypeLink PenStyle**: default line style (solid, dotted, etc.).
   - **DiagramTypeLink PenSize**: default line width.
   - **DiagramTypeLink BrushColor**: for double lines and/or arrows with fill, specifies default fill color.
   - **DiagramTypeLink InsideConnection**: specifies if the repository link must be represented and managed in any special way.
   - **DiagramTypeLink ReverseInReorganization**: for automatic reorganization function, enables definition of hierarchical direction of link.

Adding a Zoom to a Diagram Type

A ***DiagramTypeZoom*** enables a MetaClass to be described by a diagram type.
If you wish a given diagram type to describe a new MetaClass, you must:

- connect to the diagram type a new DiagramTypeZoom
- connect the desired MetaClass to the DiagramTypeZoom
- connect to the DiagramType the MetaAssociationEnd "Described <MetaClass>" ("Description.<MetaClass>")

To add a zoom to a diagram type:

1. In the **MetaStudio** navigation window, right-click the diagram type and select **New > DiagramTypeZoom**.
2. Enter the name of the DiagramTypeZoom and click **OK**.
3. Right-click the DiagramTypeZoom and select **Properties**.
4. Display the empty collections and right-click **MetaClass > Connect**.
5. Connect the desired MetaClass using the query tool.
6. In the **MetaStudio** navigation window, right-click the diagram type and select **Connect > MetaAssociationEnd**.
7. Connect the "Described <MetaClass>" MetaAssociation. Connect the desired <MetaClass> using the query tool.

The new MetaClass can now be described by the diagram type concerned.

Customizing the UI

Sommaire

4

CONFIGURING NAVIGATION TREES

The **MEGA** workspace is organized around navigation windows, such as **Home**, **Projects** and **Main Objects**. These windows enable access via a navigation tree to information relevant in a specific context.

MEGA Studio enables creation and modification of navigation trees.

The following points are covered in this chapter:

CONCEPT AND DEFINITIONS

Navigation Window Content

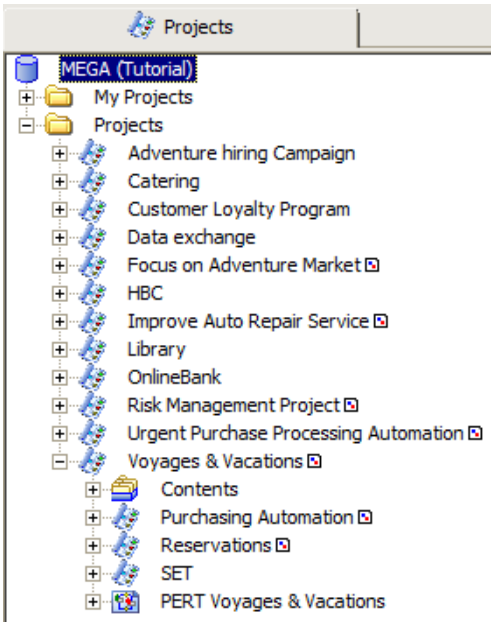
Navigation windows proposed in the **MEGA** workspace enable hierarchical access to repository objects via a navigation tree.

A navigation tree comprises navigable branches which can group:

- **MEGA** objects
- Folders
- Classification folders

To view these possibilities:

1. In the **MEGA** workspace, select **View > Navigation Windows > Projects**.
   The following window appears on the left of your workspace.



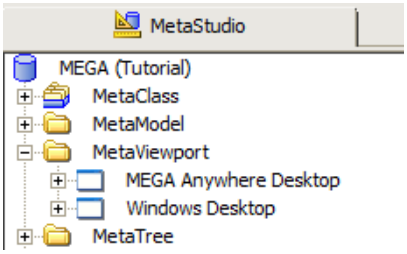
2. Expand the branch of the **Projects** folder.
   Note that sub-branches are objects of **Project** type.

   ☛ *The sub-branches of a navigation tree can contain folders or classification folders.*

Navigation Tree Structure

To access the structure of a navigation tree:

1. In the **MEGA** workspace, select **View > Navigation Windows > MetaStudio**.
   The majority of existing navigation trees are grouped in folders accessible from the **MetaViewport** folder. Trees not attached to any context are in the **MetaTree** folder.

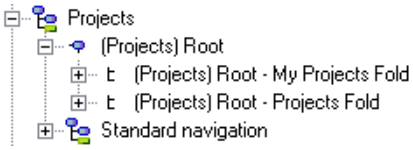


   ☛ *To access these folders, you must have opened the **MetaStudio** navigation window with **Expert** metamodel access.*

2. Expand the **MetaViewport** folder.
   A list of the different contexts in which existing navigation trees can be used appears.

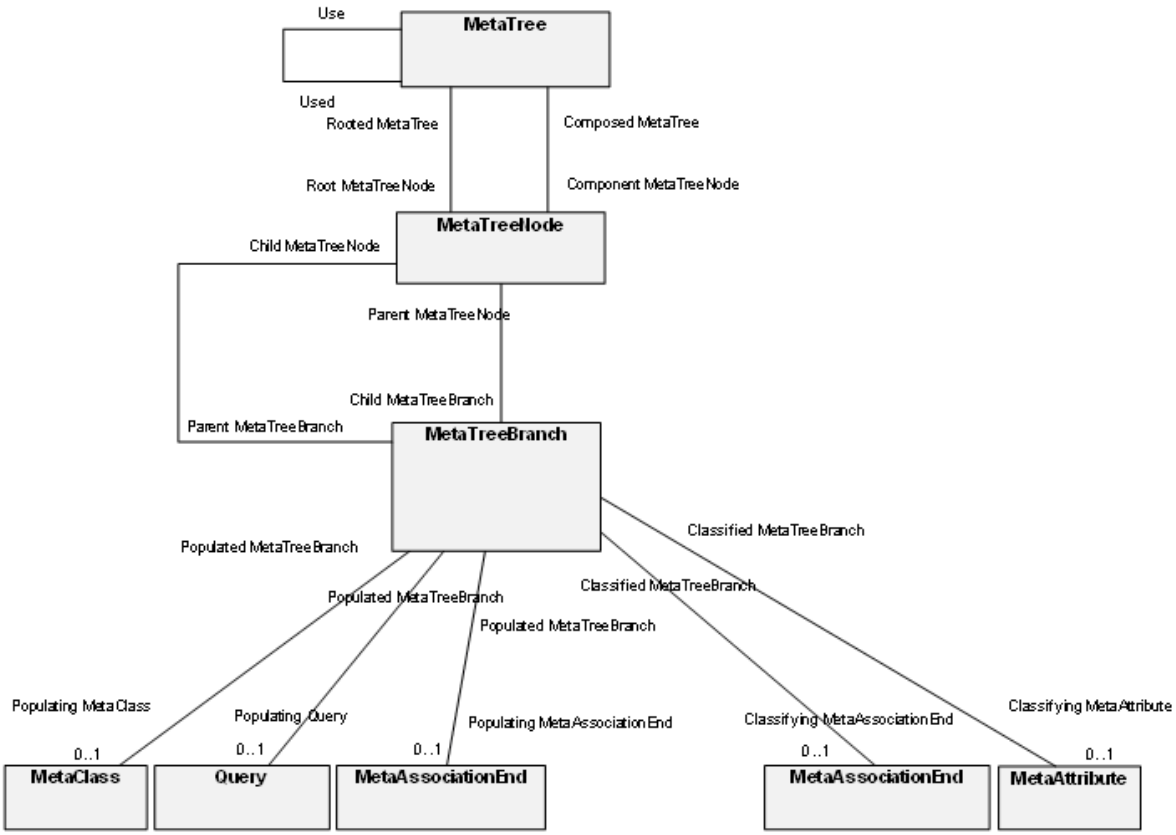
   ☛ *MetaViewport is a MetaClass. It is installed and filtered at extraction of products.*

3. Expand the **Windows Desktop** folder.
   The list of existing navigation trees in your **MEGA** workspace appears.
4. Expand for example the **Projects** tree, then the **(Projects) Root** node.
   The main branches of the **Projects** navigation window appear:
   - "(Projects) Root - My Projects Fold"
   - "(Projects) Root - Projects Fold"



Concepts Overview

Navigation tree configuration uses concepts presented in the following metamodel.



The "MetaTree" MetaClass represents the navigation tree.

The "MetaTreeNode" MetaClass represents the tree nodes. Since a node is required for creation of a branch, each MetaTree has a root node from which main branches are created.

The "MetaTreeBranch" MetaClass represents the tree branches. A branch is systematically connected to a node by the "Parent MetaTreeNode" MetaAssociationEnd. If it includes sub-branches, it is associated with a child node by the "Child MetaTreeNode" MetaAssociationEnd.

The "Populating MetaClass", "Populating Query" and "Populating MetaAssociationEnd" MetAssociationEnds enable definition of content of the branch.

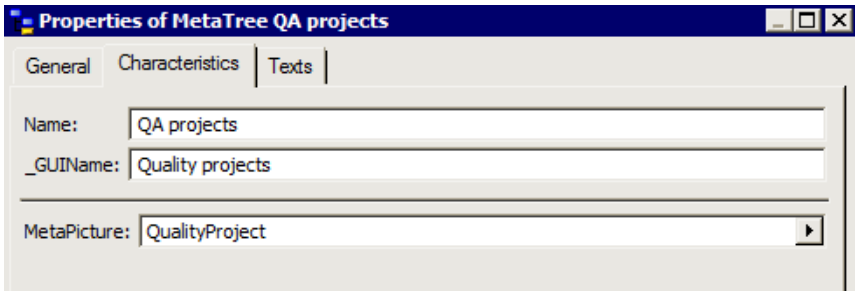
The "Classifying Attribute" and "Classifying MetaAssociationEnd" MetaAssociationEnds enable definition of classification criteria of branches associated with classification folders.

CREATING A NAVIGATION TREE

Creating the object associated with the navigation tree

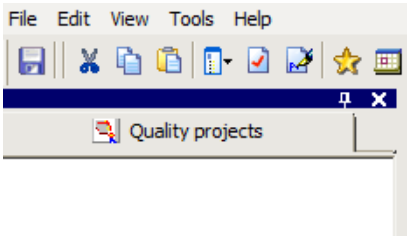
To create a navigation tree:

1. In the **MetaStudio** navigation window, expand the **MetaViewport** folder.
2. In the **MetaViewport** folder, right-click **Windows Desktop** and select **New > MetaTree**.
3. In the creation dialog box, enter the name of the new **MetaTree** and click **OK**.
4. From the Properties window of the newly-created MetaTree, specify the name and picture characteristics that will be associated with the tree in the navigation window.



To see the result:

1. Exit and restart **MEGA.**
2. In the **MEGA** workspace, select **View > Navigation Windows >** "Name of new MetaTree"**.**



Creating the root node

The root node of a navigation tree is required for creation of main branches of the tree.

To create the root node:

1. Right-click the MetaTree and select **New > MetaTreeNode**.
2. In the dialog box that opens, enter the name of the new node and click **OK**.

   ☛ *The convention adopted for naming the root node is: (<MetaTree Name>) Root.*

   Example: (Project) Root.

   You can create branches from this root node.

CREATING NAVIGATION TREE BRANCHES

MEGA Studio enables creation of branches of which content can be:

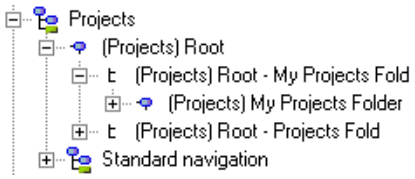
- Objects
- Folders
- Classification folders.

This paragraph covers creation of the different branch categories and introduces the filter concept.

In extended mode, you can sort objects. This functionality is described in section "Sorting Content of a Branch", page 132.

Creating a Branch Corresponding to a Folder

The **My Projects** branch of the **Projects** navigation window corresponds to the "(Project) Root - My Projects Fold" branch of the "Projects" tree. This branch itself contains other branches, of which certain are folders.



Creating a branch of folder type

To add a new branch to a navigation tree:

1. In the **MetaStudio** navigation window, expand the **MetaViewport** folder.
2. Position on the navigation tree concerned, for example "Projects".
3. Right-click the parent node of the branch, and select **New > MetaTreeBranch**.
4. In the dialog box that appears, enter the name of the new branch and click **OK**.

> ☛ *The convention adopted for naming a MetaTreeBranch of folder type is:*
>
> *(<MetaTree Name>) <Parent Node Name> - <Folder Name> Fold.*
>
> *Example: (Project) Root - My Projects Fold.*

Creating a node from a branch

A branch containing a child node carries the name of this node.

To create a child node from a branch:

1. Right-click the branch that interests you and select **New > MetaTreeNode**.

2. In the dialog box that opens, enter the name of the new node and click **OK**.

   ☛ *The convention adopted for naming a MetaTreeNode in a branch of folder type is:*

   (<MetaTree Name >) <Folder Name > Folder.

   Example: (Project) Root My Project Folder.

3. Create branches describing the content of your new branch of folder type, see "Creating a Branch Containing an Object List", page 122.

   ☛ *If no sub-branch is defined for a branch of folder type, no sub-branch will appear in the navigation window.*

Creating a Branch Containing an Object List

To create a branch containing a collection of objects of a given MetaClass:

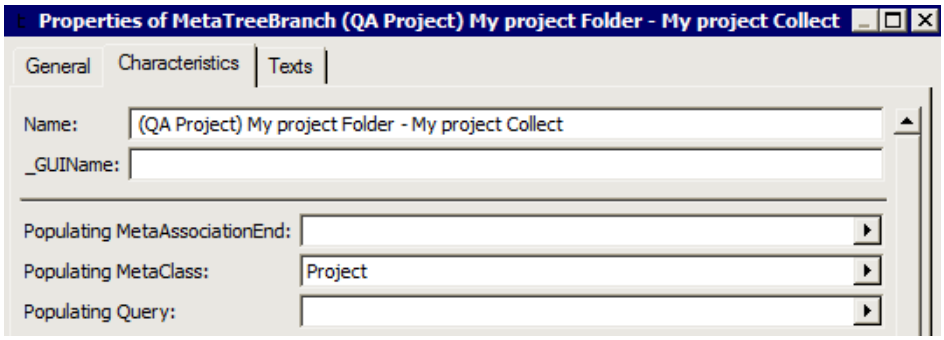
1. Right-click the parent node and select **New > MetaTreeBranch**.
2. In the dialog box that appears, enter the name of the new branch and click **OK**.

   ☛ *The convention adopted for naming a MetaTreeBranch containing an object list is:*

   (<MetaTree Name >) <Parent Node Name > - <Category Name > Collect.

   Example: (Project) My Project Folder - My Project Collect.

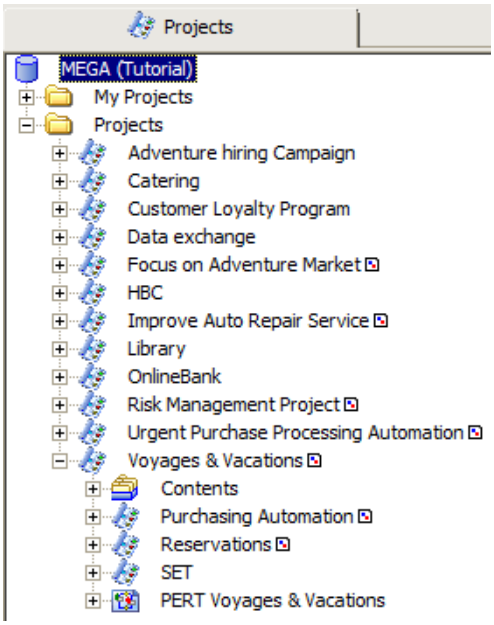
3. Expand the node corresponding to the folder and in the Properties window of the newly-created MetaTreeBranch, specify the MetaClass of the objects it contains.
   In the example below, this is the "Project" MetaClass.



To see the result:

1. From **View > Navigation Windows** , select the **MetaStudio** navigation window.

2. Expand the folders **MetaViewport**, **Windows Desktop**, then the tree you have created.
   Note that in this example, the branch created contains the list of objects of Project MetaClass.



Creating Classification Folder Branches

The **Utilities** navigation window contains a **Queries** branch.

- Expand this branch.
  Two folders appear (Internal Query/Usual Query) corresponding to query stereotype values.
  Queries can also be classified in specific folders.

Creating a classification folder

Classification criteria are specified from the value of an attribute or a MetaAssociationEnd.

For example, to classify your projects as a function of their type, you can use the "Project Type" MetaAssociationEnd.

Names assigned to the different folders correspond with the different values taken by the attribute or MetaAssociationEnd.

To classify objects of a folder:

1. Create a MetaTreeBranch from the MetaTreeNode corresponding to the parent folder, enter its name and click **OK**.

   ☛ *The convention adopted for naming a MetaTreeBranch intended for classification is:*

   (<MetaTree Name >) <Folder Name > - <Classification Criterion Name > Classify.

   Example: (Object) Root - Project Type Classify.

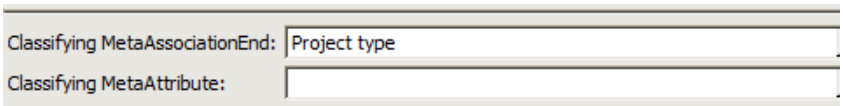
2. Create a MetaTreeNode from the new branch and click **OK**.

   ☛ *The convention adopted for naming a MetaTreeNode enabling classification of objects is:*

   (<MetaTree Name >) <Folder Name > - <Classification Criterion Name > Classify.

   Example: (Object) Root - Project Type Classify.

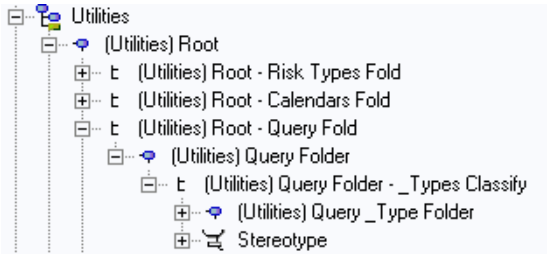
3. Open the Properties window of the branch created for the classification and specify the classification criteria.
4. In the Classifying MetaAssociationEnd box, enter for example "Project Type".



Defining content of classification folder branches

This step consists of defining branches associated with each of the classification folders as a function of values taken by the MetaAttribute or MetaAssociationEnd.

New branches are created from the "xxx Classify" node using the same principle as described in section "Creating Navigation Tree Branches", page 121.



Storing classification folders in a main folder

In the **Utilities** navigation window, queries are classified by the value of their "Stereotype" attribute value only.

If you wish to add another classification folder, for example as a function of query implementation mode, the **Queries** folder becomes "main folder" of the two classification folders:

- query by stereotype.
- query by implementation mode.

To store classification folders in a main folder:

1. Create a MetaTreeBranch from the MetaTreeNode corresponding to the parent folder, enter its name and click **OK**.

   ☛ *The convention adopted for naming a MetaTreeBranch intended for grouping classification folders is:*

   (<MetaTree Name >) <Folder Name > - <Classification Criterion Name > Classify.

   Examples:

   (Utilities) Query Folder - Stereotype Classification.

   (Utilities) Query Folder - Implementation Mode Classification.
2. Create a MetaTreeNode from the new branch and click **OK**.

   ☛ *The convention adopted for naming a MetaTreeNode enabling classification of objects is:*

   (<MetaTree Name >) <Folder Name > - <Classification Criterion Name > Classify.

   Examples:

   (Utilities) Query Folder - Stereotype Classification.

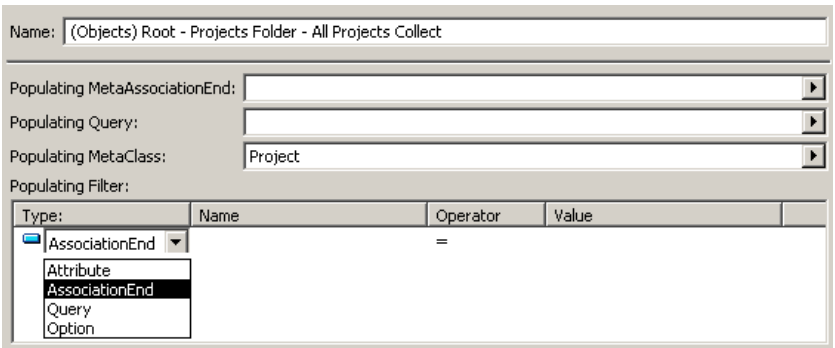
   (Utilities) Query Folder - Implementation Mode Classification.
3. Build classification sub-folders from this node.

Filtering Branch Content

A filter enables reduction of the number of objects presented in a branch using criteria defined from a MetaAttribute, a MetaAssociationEnd or a query.

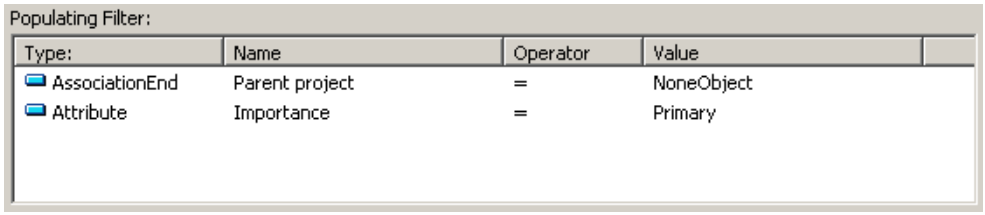
To create a filter on a MetaTreeBranch:

1. Open the Properties window of the MetaTreeBranch that interests you, for example "(Projects) My Project Folder - My Projects Collect".
2. Right-click in the **Populating Filter** space and select **Add** to create a new filter.
3. In the **Type** field of the filter created, select the type of filter you want to use.
   For example, if you want to create a filter on projects that do not have a parent project, use the "Parent Project" MetaAssociationEnd.



4. In the **Name** field, select the identifier that interests you, for example "Parent Project".

5. Select an **Operator**, then a **Value**.
   In the same way you can add a second filter criterion, for example the "Importance" attribute of the project.



To see the result:

- Close and reopen the navigation window.

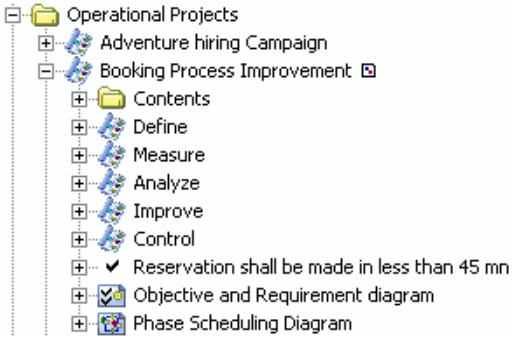
CONFIGURING NAVIGATION

Navigation in a tree depends on configuration defined at the level of:

- Navigation possibilities of the objects presented in the tree
- The nature of the tree itself
- Navigation conditions defined in advanced parameters.

Configuring Standard Navigation in a MetaClass

In the absence of specific configuration for a tree, navigation from a given MetaClass always respects the same presentation.



Standard navigation from an object is configured by creation of:

- Branches associated with object collections
- Filters defined on these object collections
- Branches associated with folders.

Configuring navigation for a MetaClass

The list of MetaClasses accessible from an object of a given type is fixed based on the MetaAssociationEnds defined for the MetaClass of the object.

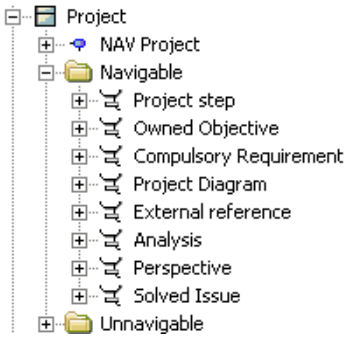
To obtain the list of navigable objects:

1. In the **MetaStudio** navigation window, expand the **MetaTree** folder
2. Under the **MetaTree** folder, expand the **MetaClass Navigation** folder.

3. Expand the folder corresponding to the MetaClass that interests you.

   Example: "Project".

   - The "Navigable" folder contains the list of navigable MetaAssociationEnds.
   - The "Unnavigable" folder contains the list of MetaAssociationEnds that are not navigable.



To specify that a MetaAssociationEnd is navigable, there are two possible solutions:

- In the "Unnavigable" folder, select the MetaAssociationEnd that interests you and drag this into the "Navigable" folder.
- From the Properties window of the MetaAssociationEnd concerned, select the **Characteristics** tab and in the **Navigable** box, select "Navigable".

Defining a branch from a MetaClass

To show branches in the navigation tree of an object:

1. In the **MetaStudio** navigation window, expand the **MetaTree** folder
2. Under the **MetaTree** folder, expand the **MetaClass Navigation** folder.
3. Right-click the MetaClass concerned, for example Project, and select **New > MetaTreeNode**.
4. In the dialog box that appears, enter the name of the new node and click **OK**.

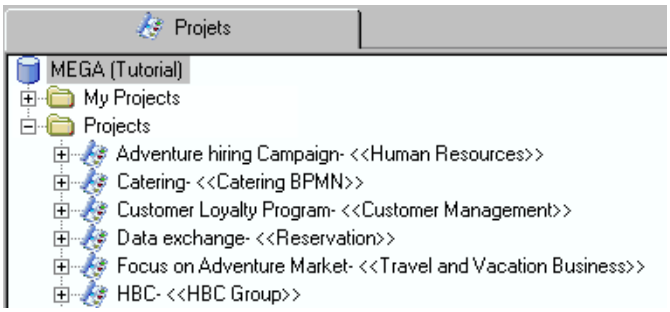
   ☛ *The convention adopted for naming this root node specific to a MetaClass is:*

   (<MetaTree Name >) <MetaClass Name > Root.
5. Create branches and filters of this new node according to the procedures described in sections "Creating Navigation Tree Branches", page 121 and "Creating Classification Folder Branches", page 123.

Managing object titles in navigation trees

Objects in navigation trees are presented by their name. This name generally corresponds to the "Short Name" attribute of the object.

If you wish project titles always to comprise the name of the project followed by the name of its library, you can modify this behavior via the MetaField concept.



Result of change of project titles in a navigation tree

The MetaField MetaClass enables redefinition of titles of objects of a MetaClass:

- from the value of an object attribute, for example its name.
- from the name of the object referenced by a MetaAssociationEnd, for example the owner library.

All components of the new title, attribute or MetaAssociationEnd, are associated with a MetaField. They are referenced by the main MetaField which is attached to the MetaTreeNode of the MetaClass.

Modifying title of a MetaClass

To modify titles of objects of a MetaClass, you must start by creating the main MetaField.

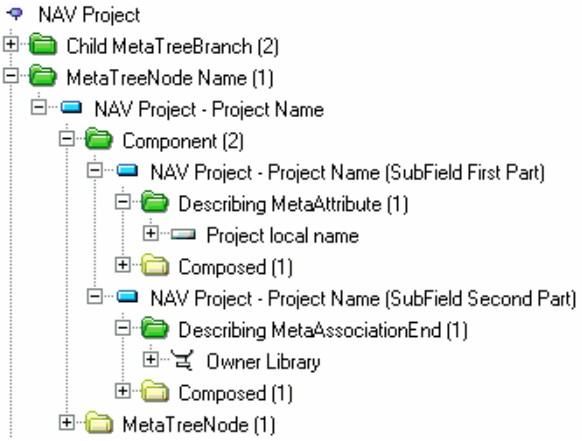
To modify title of a MetaClass:

1. In the **MetaStudio** navigation window, expand the **MetaTree** folder, then the **MetaClass Navigation** folder.
2. Expand the folder of the MetaClass that interests you (for example: Project).
3. Right-click the MetaTreeNode of the MetaClass (for example: NAV Project) and select **Explore**.
   The explorer window opens.
4. Display empty collections.
5. Right-click the "MetaTreeNodeName" folder and select **New**.
   The MetaField creation dialog box opens.
   ☛ *You must be authorized to modify MEGA data.*
6. Specify the name of the MetaField and click **OK**.
   ☛ *The convention adopted for naming the main MetaField is:*
   <MetaTree Name >) <MetaClass Name > Name.
   Example: NAV Project - Project Name

If your new title is built from a unique attribute or a single MetaAssociationEnd, you need only reference it in the MetaField you have just created. For more details, see "Defining content of a title", page 130.

Creating a composite title

If the new title is composed of several attributes or references, you must create a MetaField per component and add a reference to its content. Each component MetaField is created from the main MetaField.



Result of exploring a standard MetaTreeNode with modification of its title

To create a component MetaField:

1. Right-click the main MetaField (for example: "NAV Project - Project Name") and select **Explore**.
   An explorer window opens.
2. Display empty collections.
3. Right-click the "Components" folder and select **New**.
   The MetaField creation dialog box opens.
4. Specify the name of the sub-MetaField and click **OK**.

   ☛ *The convention adopted for naming a MetaField component is:*

   <MetaTree Name >) <MetaClass Name > Name (SubField <order> Part).

   Example: NAV Project - Project Name (SubField First Part).
5. Similarly create a MetaField per component of the new title.

Defining content of a title

The new title can be built:

- from the value of an attribute.
- from the name of an object referenced by a MetaAssociationEnd.

To specify that a MetaField is associated with an attribute:

1. Right-click the MetaField that interests you (for example: "NAV Project - Project Name (SubField First Part)"), and select **Explore**.
   An explorer window opens.
2. Open the Properties window of the MetaClass associated with the MetaTreeNode and select the **MetaAttribute** tab.

3. Select the attribute you wish to use for the title of objects of the MetaClass, and drag it to the **Describing MetaAttribute** folder of the MetaField explorer window.

> ☛ *To specify that a MetaField is associated with a MetaAssociationEnd, proceed in the same way, dragging the MetaAssociationEnd to the **Describing MetaAttribute** folder of the MetaField explorer window.*

Adding text within a MetaField

To add text within a MetaField:

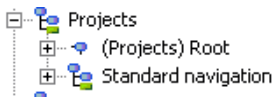
1. Open the Properties window of the MetaField.
2. Complete boxes **MetaField Prefix** and **MetaField Suffix** with character strings that you wish to add before or after referenced content.

Navigating from Tree Objects

By default, the objects presented in navigation tree branches are not navigable.

So that all objects presented in a tree will be navigable:

1. In the **MetaStudio** navigation window, expand the **MetaTree** folder.
2. Right-click the MetaTree concerned and select **Connect > MetaTree**.
3. In the dialog box that opens, enter the name of the standard MetaTree **Standard Navigation** and click **OK**.
   As an example, the result is presented below.



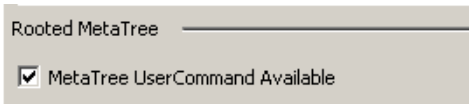
USING ADVANCED FUNCTIONS

The following configuration functions are available to a user with extended access to the metamodel.

Associating Commands with the Tree Root Node

So that commands will be available form the top of the tree in the navigation window:

- Open the Properties window of the new node and select the option **MetaTree User Command Available**.



Sorting Content of a Branch

By default, objects contained in a branch are presented in ascending alphabetical order.

MEGA Studio allows you to modify this presentation. To do this, you must specify the main attribute for sorting, as well as order of presentation (ascending or descending) of elements of the list.

For example, so that a list of "Project" type objects will be displayed in descending alphabetical order:

1. Open the Properties window of the branch concerned.
2. In the **Sorting MetaAttribute** field, select the MetaAttribute you want to use for sort.
3. In the **MetaTreeBranchSortMode** box, select the order of presentation.

Conditioning Navigation in a Tree

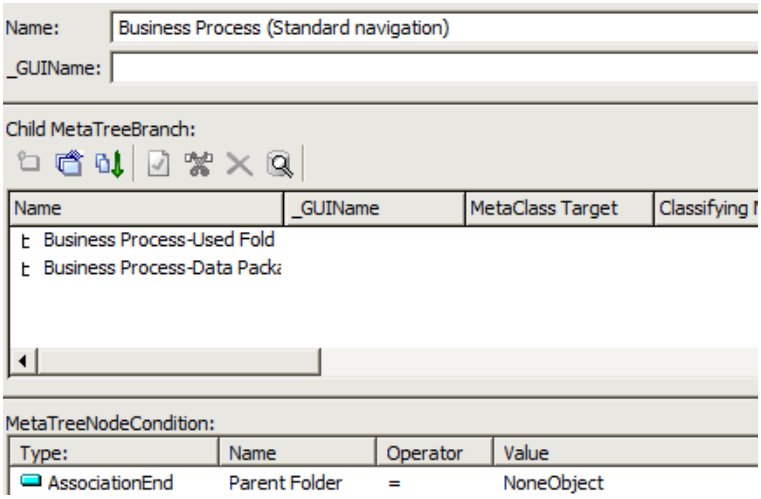
MEGA Studio offers the possibility of creating specific branches for each node of a tree. You can define navigation conditions on:

- A node
- A branch
- A classification folder

Defining a navigation condition on a node

To condition navigation on a node:

1. Open the Properties window of the MetaTreeNode that interests you, for example "Business Process (Standard Navigation)".
2. Right-click in the **MetaTreeNodeCondition** space and select **Add**.
3. In the **Type** field of the filter created, select the type of filter you want to use.
   For example, if you want to create a filter on business processes that do not have a parent folder, you use the "Parent Folder" MetaAssociationEnd.



4. In the **Name** box, select the identifier that interests you (for example: "Parent folder").
5. Select an **Operator**, then a **Value**.

   ☛ *If the node is not connected to a MetaClass, no condition is proposed.*

   *An **Option** can indicate that a condition has already been defined.*

Defining a navigation condition on a branch

Given that a branch can also correspond to classification folder, there are two types of conditioning on a branch:

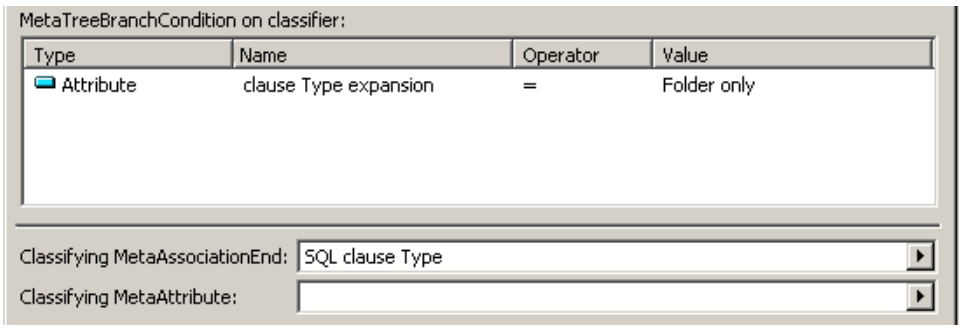
- one relating to the main branch.
- the other relating to the classification branches.

To condition navigation in a branch:

1. Open the Properties window of the MetaTreeBranch that interests you.
2. To define conditioning on the main branch, right-click in the **MetaTreeBranchCondition on Master** space and select **Add**.
3. In the **Type** field of the filter created, select the type of filter you want to use.
4. In the **Name** field, select the identifier that interests you, for example "Parent Folder".
5. Select an **Operator**, then a **Value**.

Defining a navigation condition on a classification folder

You can define a condition on a branch of the classification folder. To do this, use the **MetaTreeBranchCondition on classifier** area of the Properties window of the MetaTreeBranch that interests you, following a procedure as described in "Defining a navigation condition on a branch", page 133.

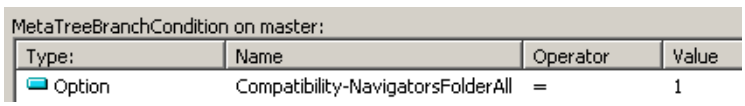


Using Options

Options can be used to define a filter, classification criterion or conditioning on a node or a MetaTreeBranch as well as on a MetaAssociationEnd or query.

You can for example specify that a branch is only navigable if the user has selected in the **Options** window, from the **Workspace** object, the option 'Display of "All..." folders in the navigator'.

This navigation limit on option appears in the Properties window of the node or branch concerned.



Forms

1 INTRODUCTION

1.1 What is FORMS?

Forms is a *framework* for design of forms used to exploit MEGA repository data. These forms are stored in the MEGA repository.

They can be used in both Windows Front-End and Web Front-End.

1.2 Accessing forms

Forms are accessed:

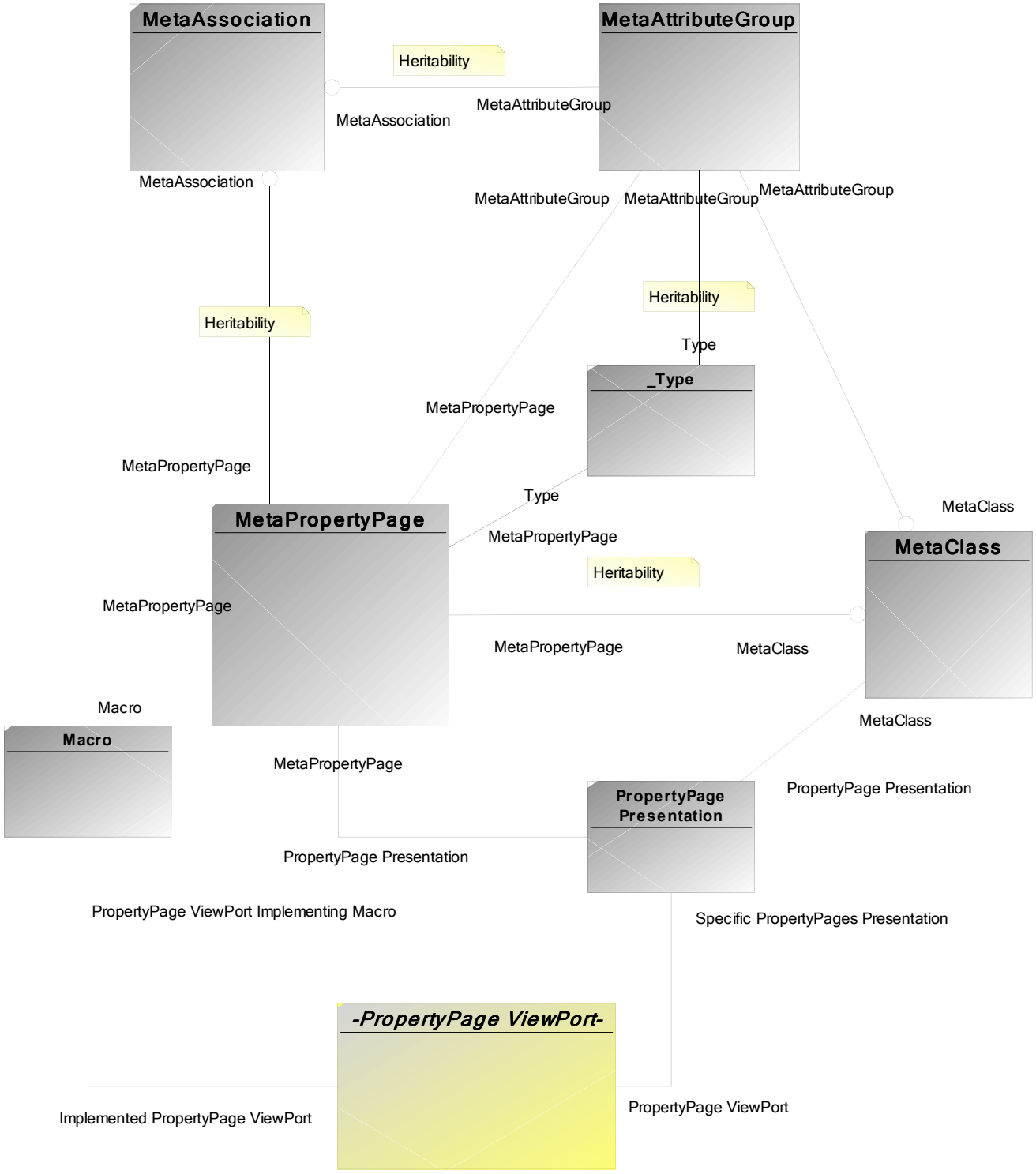
- via the "properties" menu available on MEGA objects
- in Web Front-End using a Mega Parameterized Tool
- by API, using AddPanel and PropertiesDialog functions

1.3 FORMS and wizards

MEGA wizards (modeled by the MetaWizard MetaClass) are made up of an assembly of MetaPropertyPages. Extensions specific to wizards have been integrated in forms to offer improved interaction between wizards and pages.

2 SPECIFICATION OF A FORM FOR A MEGA METACLASS

2.1 Metamodel



2.2 Standard form

2.2.1 General principles

From the time of its creation, every *MetaClass* in the MEGA repository has an implicit form without it being necessary for anything to be defined. This implicit file is built as a function of the *MetaAttributes* associated with it. As a function of their type, these attributes are broken down into several properties pages. Other more technical generic pages are added, for example a page enabling translation, and another displaying object history.

Finally, a MetaClass inherits property pages defined on the abstract MetaClasses it inherits, if the "heritability" attribute has been activated.

This standard form can be modified by parameterization. Parameterization can be carried out:

- by reorganizing presentation of the MetaAttributes of the MetaClass: the proposed breakdown may not be suitable for the desired ergonomics. This breakdown can be modified by defining specific groups of attributes on this MetaClass, modeled by the MetaAttributeGroup concept.
- by modifying the visual aspect of the main properties page (the Characteristics page), or of a pages resulting from one of the MetaAttributeGroups mentioned above.
- by inserting specific pages not concerning editing of a MetaAttribute.

2.2.2 Attribute groups and MetaAttributeGroups

2.2.2.1 Implicit groups of a Metaclass

Attributes of a MetaClass are broken down generically into the following implicit groups:

<Administration>
This group contains all administration attributes. These MetaAttributes are characterized by activation of the "*In administration tab*" flag of their *extended property*, but certain conventional *MetaAttributes* appear automatically. It is associated with the General type, has order number 1 and is therefore the page that appears first.

<Translation>
This group contains all translated attributes and texts of the *MetaClass*. It is associated with the General type and has order number 70.

<Texts>
This group contains all texts of the *MetaClass* not connected to an explicit *MetaAttributeGroup* . It is associated with the Text type and has order number 32000.

<Characteristics>
This group contains all attributes not assigned to a generic group or a specific group. It is associated with the Characteristics type and has order number 2.

<Extensions>
When the <Characteristics> group has been replaced by a specific group (see below), the <Extensions> group contains all attributes not assigned to a generic group or a specific group. It is not associated with a type and has order number 31990.

2.2.2.2 Explicit groups: MetaAttributeGroups

Generic organization of attributes and the form that results from this may not be suitable for the desired ergonomics. This is particularly the case when:

- there are a large number of MetaAttributes and/or you want to organize these as a function of the subject they cover.
- you want to see texts appear with the attributes and not in the Texts page.
- you want to define a properties page visually, or extend its content to something other than attributes.

To define a group specific to the MetaClass:

1. Create an occurrence of the MetaAttributeGroup.
2. Connect this to the MetaClass.
3. Associate with it the desired MetaAttributes.

In associating a MetaAttributeGroup to a _Type, you specify the tab in which it will appear.

A generic page is created for each MetaAttributeGroup, displaying its content. This page can be redefined by creating a MetaPropertyPage and connecting it to the MetaAttributeGroup.

It is also possible to associate with a MetaAttributeGroup TaggedValues (taken into account if they are also connected to the MetaClass) or MetaAssociationEnds (taken into account if they are legAttributes)

The MetaAttributeGroups defined on abstract MetaClasses are inherited by the sub-MetaClasses, depending on the value of the Heritability attribute.

2.2.2.3 Modification of <Characteristics> group - <Extensions> group

You may want to parameterize the Characteristics page of a *MetaClass*:

- to include attributes not originally included (eg: texts),
- to add elements not directly related to a MetaAttribute of the MetaClass (eg: a list of objects connected to a tree), or
- to specify a specific user interface for each page.

To do this:

1. Create a *MetaAttributeGroup.*
2. Connect the *MetaAttributeGroup* to the *MetaClass.*
3. Connect to it the 'Characteristics' generic *_Type*.

   This association with the *_Type* is sufficient to identify the *MetaAttributeGroup* as substitute for the generic group.

If the list of attributes to be included in the page is not modified, it is not necessary to connect the *MetaAttribute* to this *MetaAttributeGroup*: in this case, all the *MetaAttributes* that the corresponding generic group would have included are connected "virtually".

However, if at least one *MetaAttribute* is associated with this *MetaAttributeGroup*, the list is considered as complete: any *MetaAttribute* not included in a generic or specific *MetaAttributeGroup* will be added in the <Extensions> generic group. If you do not want it to appear and if implementation of the Characteristics page allows, it is sufficient to connect the *MetaAttributes* included in the Extensions page to the Characteristics *MetaAttributeGroup*.

2.2.3 MetaClass properties pages

The list of pages displayed in a properties dialog box is determined specifically for each MetaClass.

Added to deduced pages of implicit and explicit groups defined for the MetaClass are:

- Standard pages of links or menu trees.
- Specific pages directly associated with the MetaClass.

2.2.3.1 Properties page tab and order

At the time of their collection, pages are classified as a function of their type, which enables creation of the list of tabs to be displayed in the form. The order of pages serves not only to sort pages within a tab, but also to sort the tabs themselves. The order number of a tab is defined by the order number of its first page. The General tab has order number 1, the lowest number possible, and which is the number of the "Administration" page. When a tab contains only one page, this page appears directly in the position of the tab, unless explicitly specified otherwise in the corresponding *_Type*.

When the page is deduced from a group, it inherits the type and order number defined for this group as well as its name: the *_GUIName MetaAttribute* of the *MetaAttributeGroup* is used for this, or by default its name.

A type and order number are defined for each non-dependent standard page of a group. These cannot be modified.

With each page explicitly defined via an occurrence of _PropertyPage, you can associate a specific type that will enable definition of its tab. Generic implementations of pages enable redefinition of the order number.

2.2.3.2 Standard pages relating to MetaAssociations

Parameterization of certain operators related to the MetaClass enable control of display of the following pages, which display the associated objects in tree form using the MetaAssociations concerned:

- Correspondences page: displays a tree parameterized by the _Operator "Correlate". This page is added when a MetaAssociation with a behavior relating to this operator other than 'Abort' has been found on the MetaClass. It is General type and has order number 50.
- Impacts page: displays a tree parameterized by the _Operator "Impact". This page is added when a MetaAssociation with a behavior relating to this operator other than 'Abort' has been found on the MetaClass. It is General type and has order number 48.
- Complements page: displays a tree parameterized by the _Operator "Complement". This page is added when a MetaAssociation with a behavior relating to this operator other than 'Abort' has been found on the MetaClass. This page has no type and therefore appears as the main tab. It has order number 30000. Like all pages relating to an *_Operator*, the page name is by default that of the operator. It is however possible to define another name for this tab, by connecting the operator to the *MetaClass* and by specifying in the desired language the *_GUIName* attribute present on this link.

2.2.3.3 Customizing a page relating to a *MetaAttributeGroup*

The mere presence of a *MetaAttributeGroup* is sufficient to cause appearance of a specific properties page for a *MetaClass*. This page may not however be suitable:

- You want to be able to hide the properties page according to criteria specific to the occurrence.
- Presentation of attributes may not be satisfactory; you want to modify look, presentation order, title, appearance conditions, etc.
- You want to add to this page elements not derived from an element of the *MetaAttributeGroup* (for example a tree, schedule, list, attribute from another connected occurrence, etc.)

Page specific implementation has been provided by MEGA to satisfactorily manage this *MetaAttributeGroup*.

The page corresponding to this MetaAttributeGroup can be customized. This customization will be provided by an occurrence of *_PropertyPage* connected to the *MetaAttributeGroup*.

2.2.3.4 Adding a page not dependent on a *MetaAttributeGroup*

One or several pages displaying no element relating to a *MetaAttributeGroup* can be added to the properties dialog box of a *MetaClass*. This customization will be provided by an occurrence of *_PropertyPage* directly connected to the MetaClass. Customization possibilities offered by MEGA for such pages are covered later.

2.2.4 Property page filtering according to product

The list of properties pages, as well as their content depends on products installed and the view the current user has of the metamodel.

Implicit property pages cannot be specifically filtered. However, they do not display attributes hidden to the user; and an empty generic page is not inserted in the dialog box.

Depending on the case, appearance of a page is also conditioned by filtering of its corresponding *_PropertyPage* as well as by that of its *MetaAttributeGroup*: if one of these two concepts is filtered, the page does not appear.

2.2.5 Pages relating to MetaAttributes of MetaAssociations

MetaAttributes connected to *MetaAssociations* are visible either:

- from the properties dialog box of an object seen from its link, or
- in a form specific to the link: this dialog box is presented in Diagram when you want to read properties of a link, and it can also be accessed by APIs.

2.2.5.1 Form of a link

Standard behavior classifies attributes in generic groups following the same principle as for *MetaClasses*:

- <Administration> group for administration MetaAttributes.
- <Location> group for translated MetaAttributes of the MetaAssociation.
- <Text> group for MetaAttributes of Text type.

- <Characteristics> group for other *MetaAttributes*.

You can define MetaAttributeGroups on MetaAssociations.

You can define MetaPropertyPages on MetaAssociations.

2.2.5.2 Form of an object seen from a link

In the absence of parameterization, the *MetaAttributes* of the *MetaAssociation* by which the object is seen are presented in the properties dialog box. Standard processing distributes *MetaAttributes* of the *MetaAssociation* conforming to their MetaAttributeGroup. They are presented at the end of the page, preceded by a bar containing the name of the opposite *MetaAssociationEnd*.

Administration attributes, as well as the Order attribute, are displayed in the Administration page, while text type attributes are inserted in the Texts page. Other attributes are displayed in the Characteristics page.

This processing is carried out by page generic implementations only, or specific implementations processing the problem explicitly.

Specific MetaPropertyPages defined on the MetaAssociation will also be seen in this form if the heritability attribute has been activated.

2.3 Form defined by a ViewPort

The form associated with a MetaClass can be redefined in a precise use context. This use context corresponds to the *PropertyPage ViewPort* context, implemented by the *Desktop* and *Web Site Template* MetaClasses.

This redefinition is made by creating a PropertyPagePresentation associated with the ViewPort and the MetaClass. You can then associate with the PropertyPagePresentation the MetaPropertyPages you want to see in the form of the MetaClass for this ViewPort. In addition, the PropertyPage Definition attribute enables specification of whether the PropertyPagePresentation replaces the standard form – in which case the form will comprise only those pages associated with the presentation – or whether it is an addition – in which case the pages associated with the presentation will be added to the standard form.

A Macro can be associated with the ViewPort. In this macro you can implement page filtering or page addition.

You can explicitly invoke the form of an object defined by a PropertyPageViewPort using the PropertiesDialog function:

```
MegaObject.PropertiesDialog "ViewPort=<ViewportID>]"
```

where **ViewPortID** is a field containing the absolute identifier of the ViewPort.

2.3.1 Implementing page filtering

This implementation hides standard form pages that you do not want to see in the specific form. It has the following prototype:

```
Function FilterPage(Context as MegaPageLoader,PageID,TypeID) As Boolean
```

PageID is the identifier of the Page to be filtered (typically a MetaPropertyPage identifier or MetaAttributeGroup identifier, if the page is implicit)

TypeID is the identifier of the page Type, which defines the folder in which it will appear.

Context carries information on the object being loaded and the handling tools.

This function returns True if the page is visible, False if it should be hidden.

The **MegaPageLoader** component has the following methods and properties:

.Object as MegaObject: object to which the form being loaded relates (read-only)

.Title as String: title of page to be filtered (read/write)

.ViewPortID: identifier of viewport being loaded (read)

.Holder as MegaPageHolder obtains the Form component being loaded (see below)

.MacroCLSID: macro implementing the page to be filtered This enables discrimination of conventional pages if necessary

.SameID(id1 [, id2]) as Boolean: with one parameter, compare the supplied identifier with that of the properties page, with two parameters, compare the identifiers

2.3.2 Implementing a Page Loader

When you want to add specific or dynamically defined pages to a form, you must implement a method using the **InvokeOnObject** function.

This component also enables redefinition of certain form elements and should implement:

```
Sub InvokeOnObject(Source As MegaObject,Holder As MegaPageHolder,InitString as String)
```

Source: contains the object to which the form relates

Holder: component that manages the form, to which pages can be added

InitString: initialization string, empty in the case of a call from a ViewPort

The **MegaPageHolder** component has the following functions and methods:

.Caption as String (read/write) form title

.Object as MegaObject: object to which the form relates (can be empty if you load an associative object)

.Relationship as MegaObject: associative object to which the form relates, exclusive with .Object

.Description as MegaObject: description of the object to be displayed (can be empty)

.Insert(PageID[,PageName as String, PageType as Object, Obj as MegaObject, Parameterization As String, Props as MegaCollection, Options As Integer])

PageID: identifier of the MetaPropertyPage to be displayed. If the page is calculated, this can be any identifier, but different from other identifiers of the form page.

The following elements are optional:

PageName: name of the page (mandatory if the page is not a MetaPropertyPage)

PageType: _type of the page, enabling definition of the tab in which it will appear

Object: MegaObject to which the page relates if it is not the same as that of the form

Parameterization: when the page is calculated, contains the parameterization text enabling its supply. This text corresponds exactly to content of a _parameterization text of a MetaPropertyPage

Props as MegaCollection: when the page is calculated and displays properties in a standard way, this collection can replace Parameterization. It must contain a list of properties conforming to a description.

Options: bit field containing the following details:

```
#define PAGEOPTION_PROPPAGESTANDARD 0x1000
#define PAGEOPTION_PROPPAGEADMIN    0x2000
#define PAGEOPTION_PROPPAGETEXT     0x4000
#define PAGEOPTION_METAPROPPAGE     0x8000
```

PAGEOPTION_METAPROPPAGE is implicit if the Caption is not defined. Otherwise this flag indicates that the PageID corresponds to a MetaPropertyPage.

.TabStyle as String: (read/write) tab display mode (Windows Front-End only). Possible values are:

"SideList": tabs are displayed vertically on the left of the form

"ComboBox": tabs are displayed at the top in a drop-down list

"BottomTabs": tabs are displayed at the bottom

"ScrollTabs": tabs are displayed at the top in a single line

"Hidden": When there is only one page in the form

.IsStatusBarHidden as Boolean: (read/write) display of status bar

.IsCaptionHidden as Boolean: (read/write) display of caption

2.3.3 Overloading standard tabs

Standard tabs are presented with a name, a certain order and with a predefined appearance that can be modified in the context of a ViewPort. To do this, use the (ViewPort/_Type) link, and more specifically the "Parameterization Link" parameterization text in which you can redefine elements of this tab. To do this, use the Folder section of this parameterization text:

```
[Folder]
Permanent =  { YES | NO }
Selector = { TabStyle }
Order = nnn
Name = { name | <Field> }
```

Permanent YES indicates that the tab is always displayed, even if it contains only one page. NO indicates that when the tab contains only one page, we display this directly

Selector (Windows Front-End) enables redefinition of the appearance of the tab (see **TabStyle** function described above)

Order enables redefinition of the tab order number. In this case the order number of its component pages is ignored.

Name enables modification of tab name. If you want to translate this name, you must use a <Field> pointing to a system repository object of which you display the short name.

Note: You can modify standard appearance of a tab by applying this parameterization to the *_Parameterization* text of the *_Type* itself.

2.3.4 Overloading form behavior

To define generic behaviors of elements of forms:

➔ Parameterize the MetaViewPort by means of its *_Parameterization text*. You can particularly parameterize the default behavior of listViews (cf 3.5.3.1.2.10).

This enables reusing forms in different contexts.

```
[ListViewDefault]
    MultiSelection=1
    ModelessPropPage=1
    PropPageAffinity=<AffinityID>
[ListViewExport]
    <ExportName>=Item(<ButtonID>),Param(<extraParameter>),Name(<Name>),Picture=<P
ictureID>,Method=<MethodID>
```

MultiSelection indicates that listViews are by default in multiselection mode.

ModelessPropPage indicates that « Properties » command invoked from the listViews s the corresponding form in docked mode (default behavior is Popup mode). In that context, you can define the affinity to be used for these forms – that means the container in which they will be displayed – by means of **PropPageAffinity.**

ListViewExport section enables defining default export buttons in the listViews (see 3.5.3.1.2.5)

2.4 Generated form

A form can be loaded from a Macro; this Macro dynamically defines the list of pages to be added in a form, in the same way as for the macro of a ViewPort described above, and implements the method:

```
InvokeOnObject(Source As MegaObject,Holder As MegaPageHolder,InitString as
String)
```

In this case InitString can contain an initialization string enabling transmission of a context to the macro.

This form is invoked using the PropertiesDialog function:

```
MegaObject.PropertiesDialog "Loader=<macroID>[initstring]"
```

MacroID being a MegaField corresponding to the loading macro.

This field can be followed by initString characters that will be transmitted to the macro.

If the form to be invoked contains only one page, and this page is a MetaPropertyPage, it is not necessary to implement a Macro, and you can simply call:

```
MegaObject.PropertiesDialog "PropPage=<pageID>"
```

2.5 Page general parameterization

Components supplied by MEGA that can be used in customization are the following Macros:

_PropertyPageStandard: standard characteristic page. This implementation should be used to customize the Characteristics page

_PropertyPageExtension: standard page derived from a *MetaAttributeGroup*. This implementation should be used to customize the *MetaAttributeGroup* page.

_PropertyPageComment: standard text page. This implementation should be used to customize the Texts page.

_PropertyPageLink: page displaying a list of objects associated by a *MetaAssociation* specific to the dialog box object. This implementation is depreciated, the functions offered by _PropertyPageExtension to manage lists - Control(ListView) - being considerably more comprehensive; in addition, certain generic behaviors (such as export buttons) have not been implemented on this type of list.

_PropertyPageTree: page displaying a tree of which the dialog box object is the root. Standard pages displaying a tree. This implementation is depreciated, the functions offered by _PropertyPageExtension – Control(TreeView) - to manage lists being considerably more comprehensive.

_PropertyPageTreeViewOption: page displaying an options box. Used in particular in the Procedure properties dialog box to manage ISOxxx tabs. **This implementation is not compatible with HOPEX**.

The following parameterizations are to be inserted in the *_Parameterization* text of the *_PropertyPage*

2.5.1 Page name and order number

The name of the tab representing the page is the _GUIName of the _PropertyPage. If this attribute is not specified, the _GUIName of the MetaAttributeGroup that it displays is used, if this exists. If not, the Name of the _PropertyPage is used. Text pages implemented by **_PropertyPageComment** have a specific name logic: if only one text is displayed (for example Comment), this text name replaces the page name.

The page order number is defined by the option:

```
[Page]
  Order = <Nombre>
```

2.5.2 Page active by default

It is possible to specify a particular page as active by default in the dialog box.

```
[Page]
  IsDefActive = { 0 | 1 }
```

Note: This parameterization applies only to initial display of the dialog box, subsequent displays taking account of the last active tab selected by the user.

If several pages are specified as being active by default, the page effectively active will be "one of these… ".

2.5.3 Filtering a page as a function of the object

It is possible to control appearance of a page as a function of a condition relating to the form object. To do this, the following option should be specified:

```
[Filter]
  Condition = <Bool-Expression>
```

The condition relates to the object. Texts relating to MetaAttributes, TaggedValues or MetaAssociations of this object can be included. Comparison operators =, <>, <=, >=, < and >, or/and boolean operations and brackets are supported.

Attributes and occurrences can be included in hexaidabs form prefixed by # or in the form of fields.

At a binary comparison, the first element (if it is a field) is interpreted as the value of a *MetaAttribute* or a *TaggedValue* of the current object.

Comparison can be numerical or alphanumerical as a function of the variables used. If the first element is an attribute, the test type is determined by the attribute type. It should be remembered that 1000 is larger than 2 (numerical comparison) while '1000' is smaller than '2' (alphanumeric comparison).

Condition syntax description:

- Bold non-italic (and red) elements are keywords or separators of the language
- Elements between square brackets are optional
- Elements between brackets are optional and can be repeated
- Elements between curly brackets are alternative choices

```
<Bool-Expression> :: <And-Expression> ( OR <And-Expression> )
<And-Expression> :: <Condition> ( AND <Condition> )
<Condition> :: [ NOT ] {
( <Bool-Expression> ) | < Comparison> |
ItemExist( <Field> , <Field> ) |
Available( <Field> ) |
ContainString( <Field> , <Field> ) |
SeenFrom( <Field> ) |
```

```
TrueForOne( <Field> , <Bool-Expression> ) |
TrueForEach( <Field> , <Bool-Expression> ) |
IsAdminMode | IsConfidentialityCustomized | True | False | IsWindowsMode |
IsPermission( { Update | Delete | Change | Unlink | Lock } ) |
IsMetaPermission( <Field> , { Update | Delete | Change | Unlink | Create | Read }
) |
IsClass( <Field> ) |
IsType( <Field> ) |
IsChildAvailable( <Field> ) |
IsToolAvailable( <Field> ) |
IsConcreteType( <Field> ) |
ApplyTest( <Field> ) |
AccessLevel( <Constant> )
}
<Comparison> ::
{ <Field> | <Fonction> | <Constant }
    { = | <> | <= | >= | < | > }
{ <Field> | <Fonction> | <Constante }
<Function> :: ItemCount( <Field> ) |
Number( <Numerical-Expression> ) |
SessionValue( { CurrentLanguage | DataLanguage | SystemLanguage | User } )
<Numerical-Expression> :: numerical expression

<Constant> :: ' ( <Letter> ) > ' | "( <Letter> ) " |
[ - ] <Number> ( <Number> )
<Letter> :: any ANSI character other than the separation character
<Number> :: { 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 }
```

2.5.3.1 Boolean functions

ItemExist(Collection,Occurrence): true if the occurrence cited belongs to the collection.

Available(**Object**): true if the object is visible depending on products available on the key and filters defined for the user.

TrueForEach(**AssociationEnd,Condition**): true if the condition is true for all objects accessed by the collection (the condition applies to the accessed object)

TrueForOne(**AssociationEnd,Condition**): true if the condition is true for at least one of the objects accessed by the collection (the condition applies to the accessed object)

ContainString(*Champ*,Test): true if the test string is contained in the value represented by the field.

SeenFrom(Collection): true if the form object is seen from the collection

IsAdminMode: true if you are in the administration module

IsConfidentialityCustomized: true if confidentiality of the current environment has been customized

True: always true

False: always false

IsWindowsMode: true if you are in Windows Front-End

IsPermission({**Update**|**Delete**|**Change**|**Unlink**|**Lock**}**)**: tests permission required on the object

IsMetaPermission(Collection,{**Update**|**Delete**|**Change**|**Unlink**|**Create**|**Read**}**)**: tests permission required on the collection seen from the object

IsClass(MetaClass): true if the object is of the class

IsType(MetaClass): true if the object inherits the class

IsChildAvailable(Collection): true if the collection is visible from the object for the connected user

IsToolAvailable(Tool): true if the tool is visible from the object for the connected user

IsConcreteType(MetaClass): true if the object is compatible with the class

ApplyTest(MetaTest): executes the metatest on the object and returns the results

AccessLevel(Constant): tests visibility level of the current user. The first letter of the constant corresponds to the level tested: **B**eginner, **S**tandard, **E**xpert …

2.5.3.2 Other functions:

ItemCount(Collection): returns collection object number

Number(Expression): converts expression to number. Useful if you want to force numerical comparison between two values

SessionValue(value): returns absolute identifier of the requested session value, which can be:

CurrentLanguage	: session current language
DataLanguage	: session repository language
SystemLanguage	: session system repository language
User	: user connected to the session

3 FORMS PROPERTIES PAGE DESIGN

3.1 Basic principles and initialization

A Forms MetaPropertyPage is implemented by one of the two following macros:

_PropertyPageStandard: this implementation is used in "Characteristics" pages.

_PropertyPageExtension: this implementation is used in other cases.

These implementations use the _parameterization text defined on the MetaPropertyPage, and more specifically the [Template] paragraph.

They also use the MetaAttributeGroup associated with the MetaPropertyPage: the page is automatically populated by the properties defined in this MetaAttributeGroup

In addition, the _MetaPropertyPageStandard will automatically add to the form:

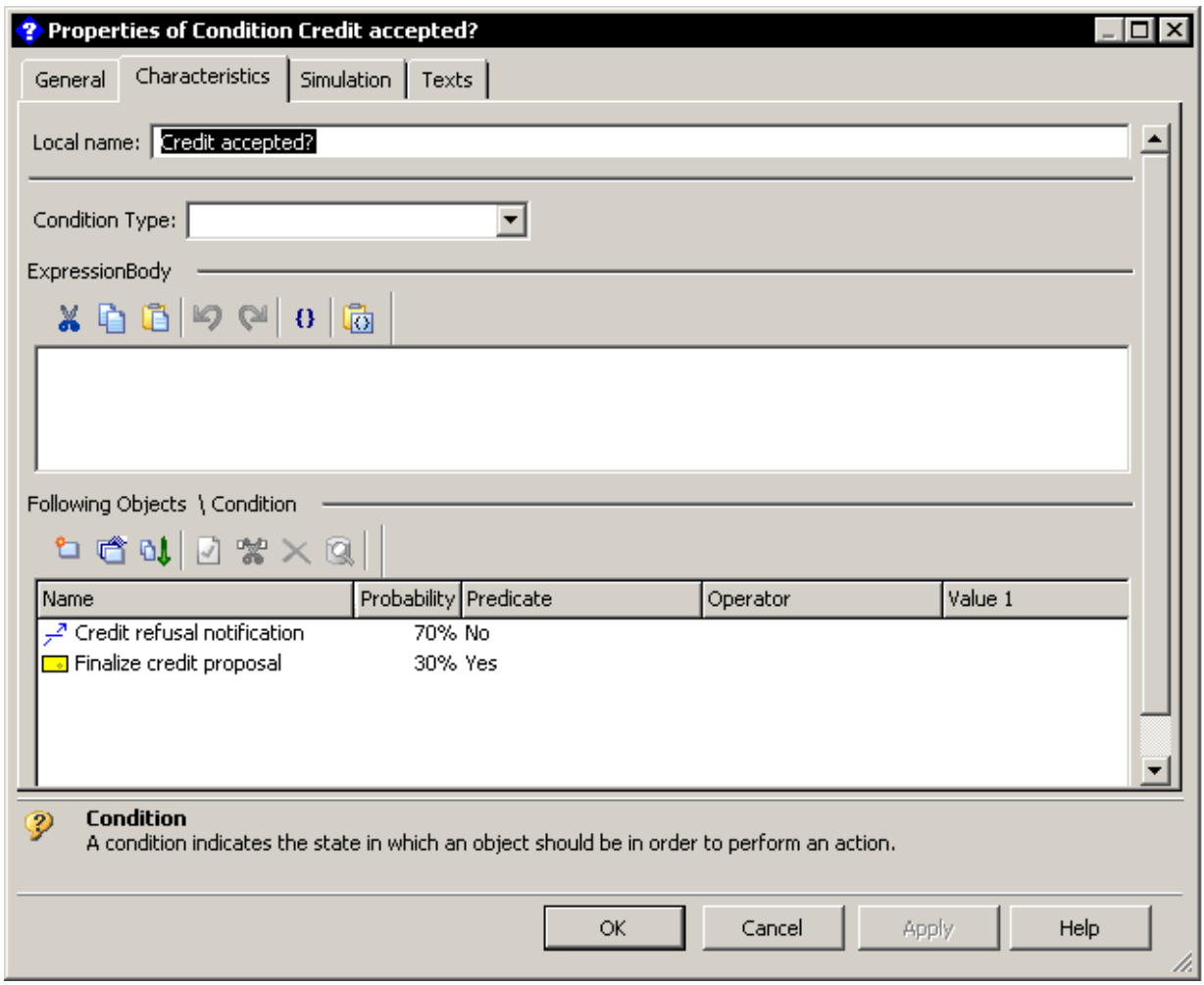
- the object name
- its owner
- the MetaAttributes defined in the accessed MetaAssociation (if the object is seen from a MetaAssociation) and not associated with a specific MetaAttributeGroup of the MetaAssociation.

These implementations also use other sections of the _parameterization text, enabling definition of page behavior.

The name of the properties page is the *GUIName* of the *_PropertyPage*, except for the Characteristics page of which the name is predefined. If not specified , the *GUIName* of *MetaAttributeGroup* is used.

The following example indicates programming of the condition page shown below.

```
[Template]
AttGroup=Group(Bar),Pos(Top)
FObjGroup=Group(Bar),Pos(Bottom),Name(Following     objects)
ExprGroup=Group(Bar),Pos(Middle),Name(ExpressionBody)
FObj=Map(Following     objects)
TypeAtt=Item(Condition     Type),In(AttGroup)
FObjList=Item(Following
objects),Contains(FObj),In(FObjGroup),Control(ListView),Title(No)
ExpressionAtt=Item(ExpressionBody),In(ExprGroup),Control(Text),Title(No)
```



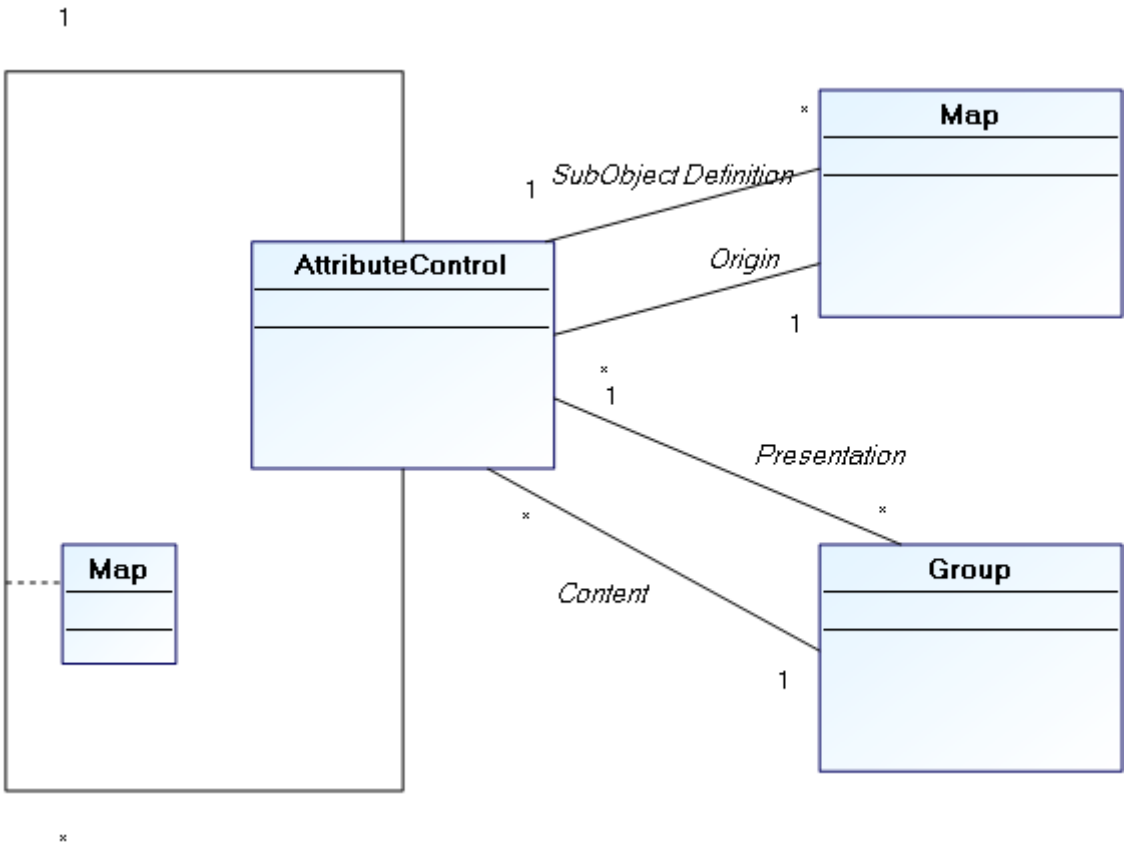
Page minimum size can be defined in another section of the parameterization text:

```
[Page]
MinWidth = <width>
MinHeight = <height>
<width> and <height> are specified in DialogUnits.
```

This parameterization can be essential if you want to define specific deformations of elements at page resizing: in this case, the minimum page size is used as the basic size in the parameterization.

For information, DialogUnits is a unit independent of font size. In this unit, a medium character of the font used occupies a rectangle 4(width) x 8(height). To obtain the effective size of the page, you need to know the medium character size of the font used to display the form.

3.2 Properties page logic model



3.2.1 AttributeControl

A page is defined as being an AttributeControl hierarchy. In the model of the above object, the page is itself an AttributeControl.

An AttributeControl is therefore a page element. It relates to a MEGA object and is associated with a metamodel concept representing information on this object.

There are three main AttributeControl types:

- **Implicit AttributeControls**

  These are automatically associated with MetaAttributes, TaggedValue (or more generally AbstractProperties) and MetaAssociationEnd _LegAttributes, depending on their type and format. They therefore reference properties directly defined on the object.

  Implicit AttributeControls are suitable in the vast majority of cases to adequately display object properties, as well as most extensions (AbstractProperties or MetaAssociationEnd). These extensions are not directly associated with the object, but are defined in the System repository and must have a value for the object. The keyword XRef(True) enables indication of an element as extension, and directs the template analyzer to the correct AttributeControl type to be displayed.

- **Explicit AttributeControls**

  These propose specific data entries on object properties, or display of calculated data obtained from the object and defined by a metamodel concept (for example a

matrix, an HTML formatter or a MetaTree), or display of presentation elements (titles, comments) referencing system repository elements without direct reference to the displayed occurrence (for example codeTemplates or resources).

- **Composite AttributeControls**

  These are explicit AttributeControls of which definition is complex and included in the template. This definition can reference system repository objects, which will be associated with the composite element by a composition map. For example, to define a ListView, you may explicitly define its columns, which are mainly MetaAttributes: The latter will be cited as the composition map element associated with the ListView.

3.2.2 Maps

Maps are used to logically group AttributeControls. They are associated with Page type AttributeControls. There are two types of map:

- object maps
- composition maps

3.2.2.1 Object maps

These enable redefinition of the object associated with an AttributeControl. In this way it is possible to display in the same form properties from several distinct objects. These maps are explicit elements of page type AttributeControls, indicating the list of objects to which the page relates.

A map of this type points to a specific object, and does this in several ways:

Pilot maps: these are commanded by a composite AttributeControl (for example a ListView or a TreevVew). In this case the Pilot(AttCtlName) keyword associated with the map indicates the AttributeControl that will pilot it. The object of the map therefore corresponds with the object selected in the piloting control. When the selection changes, all AttributeControls of the page are reinitialized with the new object of the selection. If no object is selected, map elements are deactivated. In a map of this type, the identifier used in the Map(ident) keyword is not used.

Unique collection maps: these maps have a collection as identifier. In this case the first element of the collection is considered as the object of the map. If the collection is empty, map elements are deactivated.

Maps on object: these maps are for calculated pages, since they reference an explicit object of the repository, by keyword Child(childID). The object pointed by the map must be connected to the object of the page by the collection identified in the Map, and correspond to the absolute identifier supplied.

Maps on root: identified by the keyword Root(Yes) , these maps point to the repository root, and are therefore independent of the displayed object.

Context maps: these maps enable use of connections to tools displaying pages. For wizards, the "Context" name implicit map points to the collaborative object of the wizard.

3.2.2.2 Composition maps

Composition maps enable definition of content of a composite AttributeControl: for example, columns of a ListView or branches of a TreeView are defined asAttributeControls, defined in a composition map. This map will then be associated with the MetaAttributeComposite. These

maps do not have a real existence in the object model of the form, since they merely participate in definition of the hierarchy of the AttributeControls.

3.2.3 Groups

Groups are used to define appearance of a form. They are associated with Page type AttributeControls.
Groups defined for the page are assembled vertically in a defined order. A certain number of groups are implicit and are created automatically when Attributes are included.
The page analyzer first lists the composition of the *MetaAttributeGroup* (if this exists). We then search in the configuration for any possible elements that could overload it. If there is no overload, or if the overload does not specify a group, these attributes are arranged in standard groups as follows:
Naming groups (short name, long name) are associated by default with the **NameGroup** implicit group as follows:

- Extensions are associated with the **ExtensionGroup** implicit group
- Texts are arranged in a control associated with the text group
- Attributes derived from the association are associated with the **LinkGroup** implicit group
- Other attributes are associated with the standard group.

NameGroup, **ExtensionsGroup** and **LinkGroup** groups can be used in the parameterization: you can then associate specific elements.
Groups are positioned in the page in the following way, **Pos(xxx)** being a syntactic element enabling definition of the general position of a group. It should be noted that Groups of the same position are ordered alphabetically according to name.

Properties page group positioning

```
NameGroup
PosGroups (Top)
StandardGroup
PosGroups (Middle)
ExtensionsGroup
PosGroups(Extension)
LinkGroup
PosGroups (Bottom)
TextGroups
```

The text group is always positioned at the end; only the last zone displayed can be resized vertically, and texts are the most likely elements for resizing. However, if you want to display a ListView that can be resized, it can be positioned alone in a Group in 'Bottom' position. For example, this is what is proposed by the condition page above. It explicitly positions the *ExpressionBody* text in another group so the ListView will effectively be the last zone displayed.

3.3 Specifying templates

3.3.1 Syntax

Page programming is carried out in the *_Parameterization* text, which is a configuration text: it must therefore conform to configuration text compatible syntax. The `[Template]` paragraph of this text is used.

All elements included in this paragraph should be defined in a single line.

Each element is named to conform to configuration text syntax (name = definition). The name is never included in the properties page, but is used to order groups.

It is essential that the order of parameters of each element be respected.

Syntax of elements is detailed below. Elements in bold (red) are keywords; optional parameters are between curly brackets. Alternative elements are between curly brackets separated by vertical bars

For certain elements it is possible to specify positions and sizes. Characteristics should always be expressed in *Dialog Units*.

Elements recognized by this syntax are:

3.3.1.1 Conditions

Conditions enable conditioning of the appearance of groups or of elements. The condition relates to the properties dialog box object occurrence.

Condition relating to the form object:

```
<Name> = Condition(<Bool-Expression>)
```

Condition related to the object pointed by a map:

```
<Name> = ConditionFrom(<MapName>[:{True|False}],<Bool-Expression>)
```

If the map does not point to any object, the Condition is evaluated as False, unless otherwise specified after the colon.

Syntax of conditions is described in section 2.5.3.

3.3.1.2 Groups

These enable positioning of elements with respect to each other. In addition, it is possible to condition appearance or deactivation of the elements included.

Elements appearing in the same group are canonically presented in their order of declaration. It is however possible to include them in the order defined by the metamodel using the MetaClass/MetaAttribute, MetaClass/TaggedValues and MetaClass/MetaOppositeAssociationEnd links using the following option:

```
[Page]
NoGroupOrder = 1
```

The groups are declared in the template using the following syntax:

```
<Name> = Group(<Aspect>)[,Pos(<Position>)][,Name(<Field>)][,At(<column>[,<line>])]
[,Size(<width>,<height>)][{,HiddenOn(<ConditionName>)|,DisabledOn(<ConditionName>
)}]
[,Initially(<ConditionName>)]
```

<Aspect> :: {Bar|Frame|VisibleBar|Hidden }: Characterizes physical representation of the group - single bar at top, frame around attributes or nothing. **VisibleBar** indicates that the bar is visible, even if the group is the first on the page. Groups of **Frame** can be collapsed in Web Front-End.

<Position> :: {Top|Bottom|Extensions|Middle}: Characterizes overall positioning of group in page. Default is **Middle**.

<Field>: Field relating to any MEGA occurrence of which the name, preferably translatable, will be displayed in the page as the group name. You can also include a simple character chain, which will be non-translatable.

<column>: Numerical value in Dialog Units indicating horizontal shift of elements included in the group. This data can be included alone.

<line>: Numerical value in Dialog Units indicating vertical position of the top of the group.

<width> <height>: Group height and width. As for <line>, these parameters are optional and need only be used when the specifier wishes to precisely control the position of a group. If not specified, group size is calculated as a function of the cumulative size of groups included, and group position as a function of other groups present in the page.

<ConditionName>: Name of a previously defined condition. If the condition is not fulfilled, the group and its entire content are either masked (**HiddenOn**), or deactivated (**DisabledOn**). The keyword **Always** can be used instead of a condition. Applied to the keyword **Initially**, the condition indicates the initial appearance of a collapsible group (this is only effective in Web Front-End).

3.3.1.3 Maps

```
<Name> = Map(<Field>){,Root(Yes)|,Cookie(<Field>)|,Child(<Field>)}
[,Visibility(<Visibility>)][,Advisor(Yes)][,Pilot(<ItemName>)]
<Field> : field referencing a MEGA object
```

<Visibility> : {Always|IfExist|IfFrom}: Enables conditioning of display of Map elements. Default is **Always**, and Map elements are then systematically displayed (they are disabled if the connected object is not found). If not specified, the elements obtained defined from this Map are masked if the connected object does not exist (**IfExist** case) or if the object cannot be seen from the MetaAssociationEnd opposite the specified MetaAssociationEnd (**IfFrom** case). In this case the Map points to the parent object.

For composition maps, no parameterization (including <Field> of the map) is taken into account. These maps serve only to group AttributeControls defined for a composite AttributeControl, and it is the latter that uses the elements associated with it as required.

For the other maps, <Field> specified after the map keyword represents the Collection enabling access to the object pointed by the map from the form object. This collection is therefore

generally a *MetaAssociationEnd*, a *Query* or any form of *Abstract Collection* accessible from this object. The object pointed is determined as for the following API function:

```
obj.GetCollection("<Field>").Item(1)
```

When the Collection contains several elements, the result is therefore unspecified, except if the keyword **Child** indicates the occurrence to which the Map should point.

In this case, the object pointed is determined as for the following API function:

```
obj.GetCollection("<MapField>").Item("<ChildField>")
```

This use is however generally reserved for pages generated dynamically, since such an identifier is rarely available at specification.

The keyword **Advisor(Yes)** enables definition of a Map that is not directly use to display elements but that causes the page refresh when a modification is detected in the collection with which it is associated.

The keyword **Refresh(Yes)** indicates that the collection must be reselected at page refresh. It must be used when the collection associated with the Map is a request or a calculated association, and when its content might change while the form is displayed.

The keyword **Pilot(<ItemName>)** enables to indicate the form element, which will define the object pointed by the map.
When an object is selected in the element, this notification is processed by the map that modifies, as appropriate, its pointed object. Only elements that enable selection of an element can be pilots, either listViews, treeViews, or Metatrees.

Maps also enable access to objects not directly associated with the form object:

The keyword **Root(Yes)** indicates that the map points to the repository root:

```
obj.GetCollection("<Field>").getRoot
```

The keyword Cookie enables referencing of a context object. By compatibility, this keyword enables access to the collaborative object of a wizard (**`Context.cookieObject.`**
In this case the Cookie corresponds to the identifier of the *MetaClass* MetaWizard (~DutIllKV5X40[MetaWizard]), and the identifier of the Map corresponds to that of the collaboration of the cookie.

This access mode has however been replaced by the **Context** implicit map, which enables access to this object without needing to know the identifier of the collaboration.

The keyword **Visibility(**<Field>**)** enables definition of a Map pointing to the parent object of the form object, when the latter is obtained from a collection. In this case <Field> must correspond to the identifier of the accessed MetaAssociationEnd.

The **FromLeg** implicit map indicates the associative object of the form object; when the latter is obtained from a Collection (**`obj.getRelationship`**). By using it you can include MetaAssociation generic attributes in the page.

By compatibility, the keyword **LegObject** is synonymous with **Map**.

3.3.1.4 AttributeControls

AttributeControls enable definition of page content, or composite elements of this page. Within the same Map (keyword **From**), all AttributeControls must have distinct <Field> identifiers, since it is not desirable to display the same element in the same form several times (at update in the two elements, we cannot determine which value will finally be used). If necessary, the keyword **Duplicate(Yes)** enables explicit specification of a duplicate.

When the AttributeControl is not defined in a map, either a composite or calculated element will appear, but more generally a property of the form object. Elements defined in the MetaAttributeGroup associated with the page are implicitly added; if an element is redefined here (its <Field> then corresponds to the identifier of a property listed in the MetaAttributeGroup), it replaces the implicit element: In this way you can therefore modify implicit display of an intrinsic property of the form object. If an implicit element appears in the page and is not connected to a MetaAttributeGroup, this may mean that it appears in a page other than that actually defined: You should therefore check that these two pages are not present simultaneously in the form, or that redundancy of the elements is not problematic (which is the case for example when one of the two elements is read-only).

When the element concerns a property of the object concerned, it is not generally necessary to include the control type to be displayed, this latter being deduced from the metamodel.

The keyword **XRef(True)** offers this implicit processing of properties not directly associated with the object (for example *TaggedValues*). This keyword indicates that the interpreter of the page can consider that the definition of the <Item> is accessible in the repository (for example via the **GetPropertyDescription** API function) and that it can be used to determine the implicit control type to be used.

```
<Name> = Item(<Field>)[,From(<MapName>)][,Duplicate(Yes)]
  [,{Contains|Remoting}(<MapName>)]
  [,In(<GroupName>)]
  [,Control(<ControlName>)]
  [,At(<left>[,<top>])][,Size(<width>,<height>)][,MaxSize(<width>,<height>)]
  [,VClip(<VClipMode>)][,HClip(<HClipMode>)]
  [,Visibility(<Visibility>)][,Name(<NameField>)]
  [,Title(<TitlePos>)][,At(<left>,<top>)][,Size(<width>,<height>)]
  [,Param(<Param>)]
  [,{HiddenOn|DisabledOn}(<ConditionName>)]
  [,Mandatory(<ConditionName>)]
  [,XRef(True)]
```

<Field>: References the MEGA element mapping the object. This can generically be a MetaAttribute, a TaggedValue, a MetaAssociationEnd. For implicit AttributeControls, the corresponding value must be accessible by the API function **obj.getProp("<Field>")**. For explicit or composite AttributeControls, other object types are possible, and use of <Field> depends on the control type.

<MapName>: Name of a Map relating to the element. It must be explicitly defined, but can be implicit (map **Context** map or **FromLeg** map)

The **From** Map indicates that the AttributeControl does not relate to the object, but to the object pointed in the map

The **Contains** Map is used on composite AttributeControls and enables definition of the list of its component AttributeControls. Implicit maps cannot be used in this case.

(compatibility): The **Remoting** Map indicates that the AttributeControl pilots a map; in this case the Control should not be specified, since the keyword only operates for **Control(DropDownSelection).** This specification has been replaces by keyword **Pilot** defined on the Map.

<GroupName>: Name of the Group in which the element will be physically located. The group must be defined before the element if it is explicit.

<ControlName>: Identifies the name of the type of graphic element to be used. The list of controls and their specificities are define in chapter 3.5. This parameter is not necessary in the case of an implicit control (property defined on the object, or external reference indicated by keyword **XRef(True)**.

<left>, **<top>**: Horizontal and vertical coordinates, relative to the group, expressed in DialogUnits (except exceptions). The vertical coordinate can be omitted; in this case the horizontal coordinate expresses shift relative to the group. These coordinates are optional, the element being positioned below the previous element by default.

<width>, **<height>**: Element height and width, expressed in DialogUnits. If not specified, a default size appropriate to control type and property characteristics is used.

<VClipMode>, **<HClipMode>**: These parameters enable management of deformation of a control at page resizing.

VClipMode enables definition of vertical deformation. It can have the following values:

No: No vertical deformation – distance from top of element to top of page is fixed.

Yes: Deformation proportional to that of page: distance from top of element to top of page is fixed, as is distance from bottom of page to bottom of element.

TopToBottom: Equivalent to **Yes**.

Bottom: No vertical deformation of element, distance from bottom of element to bottom of page is fixed.

Center: No vertical deformation of element, distance from bottom of element to middle of page is fixed.

TopToCenter: Distance from top of element to top of page is fixed, as is distance from bottom of element to middle of page. Vertical deformation of element in proportion half of page deformation.

CenterToBottom: Distance from top of element to middle of page is fixed, as is distance from bottom of element to bottom of page. Vertical deformation of element in proportion half of page deformation.

HClipMode enables definition of horizontal deformation. It can have the following values:

No: No horizontal deformation. Distance from left limit of element to left of page is fixed.

Yes: Distance from left limit of element to left of page is fixed, as is distance from right limit of element to right of page. Stretch of element is therefore proportional to that of page.

LeftToRight: Equivalent to **Yes**.

Right: No horizontal deformation. Distance from right limit of element to right of page is fixed (element remains aligned on right).

Center: No horizontal deformation. Distance from left limit of element to middle of page is fixed (element remains centered horizontally).

LeftToCenter: Distance from left limit of element to left of page is fixed, as is distance from left limit of element to center of page. Stretch of element is therefore in proportion half of page deformation.

CenterToRight: Distance from left limit of element to center of page is fixed, as is distance from right limit of element to right of page. Stretch of element is therefore in proportion half of page deformation.

Clipmodes must be defined with great care. An incorrect specification can cause overlap or covering of elements.

If you use clipmodes on several group elements, you must explicitly define minimum page size: this is the size on which elements are based to be correctly positioned. In Web Front-End, the size of the group in which the element appears is taken into account for element docking, if it is specified.

If clipmodes are not specified, standard behavior depends on control type. Most controls are of fixed maximum size and are resized horizontally when this maximum size is greater than effective page size. Composite controls and texts are resized by default to the maximum allowed by page size when they are located at the end of the page.

<Visibility>: Determines presence or absence of the element as a function of the user view of the metamodel. It can have the following values:

Standard: Applies the mapping visibility level (default value)

Always: Overrides systematic display

Admin: Displays only in extended view

Hidden: Systematically hides the zone and should therefore be used as overload of an unwanted attribute.

<NameField>: Enables replacement of the title of the element. Can be a field, and in this case the name of the referenced MEGA object will be displayed as zone title. A 'hard' non-translatable character chain can also be included. By default, the title of the zone is the name of the concept used as <Field>.

<TitlePos>: Indicates position of the zone title related to the zone. It can have the following values:

Up: The title is positioned above the element

Left: The title is positioned to the left of the element

No: The title is not displayed.

Positioning is ignored if title coordinates are explicitly given by the **At** and **Size** parameters defined immediately after. If these parameters are not defined, titles are aligned and sized to the maximum title size of the group concerned so that group elements can themselves be aligned.

<Param>: Optional configuration specific to control type. Can be a character string or a reference to a **Parameter** (in this case the parameter name is preceded by '@').

<ConditionName>: Name of a previously defined condition. If the condition is not fulfilled, the zone is either masked (use with **HiddenOn**), or deactivated (use with **DisabledOn**). However, unlike masking of groups, zone location remains reserved in the case of **HiddenOn**. If the page is not Synchronized (in this case, application of the condition does not cause page redesign). Keywords **Always** or **Never** can be included instead of a condition.

Applied to keyword **Mandatory** and to a modifiable element, the condition enables definition of whether the obligation constraint should be applied to the element.

Conditions are reassessed when displayed objects are modified in the repository, or when you modify their value (in the case of synchronized pages). If the value of a condition changes at these updates, the page can be redesigned.

By compatibility, the keyword **Attribute** is synonymous with **Item**.

3.3.1.5 Parameters

Can be used in the Param() field of an element when this field is long, complex, or includes brackets.

The chain supplied begins at the first bracket (after Param) and ends at the final bracket of the line.

In this case, the Parameter name should be included, preceded by @.

Example:

```
MyParam = Parameter(Long Parameter, with (Parenthesis) and « specials chars[>)
MyAtt = Attribute(…) …,Param(@myParam)
```

3.3.1.6 Inclusions

In page parameterization, reference can be made to another parameterization. This enables:

- factorization of elements common to several parameterizations.
- making page content dependent on elements specific to the form object.
- calculation of part of page content using a Macro.

Origin of inclusion is defined in parameterization by the keyword Origin.

Static inclusions use a parameterization included in a system repository object explicitly cited in the inclusion. In this elementary case there is no origin.

In dynamic inclusions, the system repository object in which we read the parameterization is deduced from the displayed object.

Inclusions can call a macro that will generate the parameterization to be included.

The lines of definition obtained by inclusion are inserted in the template of the page in the position where the keyword is located.

This device is recursive.

```
<Inclusion> = IncludeProfile(<ObjField>)[,DefaultOf(<InclusionName>)]
   [,From(<MapName>)][,In(<GroupName>)]
   [,Origin({FromLeg|Service|Object|ServiceList|Value|Macro})]
   [,Text(<TxtField>)][,Paragraph(<PrgName>)]
```

<ObjField>:

If the inclusion does not depend on an Origin, identify the system on which the parameterization will be queried. This object is not necessarily a _**PropertyPage**. If not, use of this identifier depends on the origin.

<InclusionName>:
Indicates that this inclusion is only effective if the <InclusionName> dynamic inclusion has found no system object on which to read a parameter. It therefore enables definition of a default content.

<From>:
When the inclusion depends on an origin, enables calculation of the origin from the object pointed by the map rather than the form object.

<GroupName>:
Includes elements of the inclusion for which no group was defined in the specific group.

<TxtField>:
Identifies the text in which configuration is included as appropriate. By default, this is the **_Parameterization**.

<PrgName>:
Name of the paragraph in which configuration will be read. By default, this is the **Template** paragraph.

The keyword **Origin** enables determination of the calculation mode allowing us to obtain the parameterization. It can have the following values:

- **FromLeg**: in this case we read the parameterization text on the MEGA object corresponding to the description of the form object. (in API: obj.gettypeobject.getID). In particular, this allows us to include elements relating to the MetaAssociationEnd from which we observe the form object.
- **Service**: in this case, <ObjField> is a MetaAssociationEnd or a property of object type, and we read the parameterization from the object corresponding to this property, (in API: obj.getProp("<ObjField>") if it is specified and belongs to the system repository.
- **Object**: in this case parameterization is read directly on the form object (in API: obj.getID). This is only possible of the observed object is in the system repository.
- **ServiceList**: parameterization is similar to **Service**, except <objField> here is a Collection identifier allowing us to obtain system repository objects from the form object. Parameterizations of accessed objects are concatenated when there are several of these (in API: for each srv in obj.getCollection("<ObjField>") …)
- **Value**: in this case <*ObjField*> is a listed property; when the value of this property for the form object corresponds to an listed value (*MetaAttributeValue* or more generally *Abstract PropertyLiteralValue*), we read parameterization on this occurrence of the literal value.
- **Macro**: in this case <*objField*> is a macro. The macro should implement the following function:

```
Function InvokeOnObject(
obj As MegaObject,
idText,
Page as MegaPropertyPageStandard) As String
```

obj being the form object, **idText** the identifier of the requested text (*_Parameterization* by default, or <*TxtField*> if specified, and **Page** the page in preparation.

This macro should send a character string which will be analyzed as a parameterization text. The paragraph [Template] (or [<PrgName>] if specified) of this text will be included in the parameterization.

3.3.2 Other options of standard pages

On a standard page you can define options that enable definition of its general behavior. These parameterizations can be declarative in the *_parameterization* text of the Page, or programmatic (see 3.4.1 and 3.4.2.4)

[Dialog]
CommandHandler=<MacroID>: Enables definition of a Macro that will be connected to the page and can listen to notifications from AttributeControls or the form (see 3.4.3)

[Page]
CheckVisibility = 1 (default) or 0: This option enables conditioning of page display due to the fact that its content is not empty at creation of the form. It should be deactivated if content of the page is calculated dynamically, but also for performance reasons; calculation of page content can be costly, and if this option is activated, should be carried out at creation of content (therefore at initialization) and not at display of the page. This option should also be deactivated if presence of the page is determined by the product or the ViewPort, and not by its content.

ReloadOnActivate = 0 (default) or 1: If activated, indicates that the page should be recalculated each time it is redisplayed. This can be useful when page content is determined dynamically and can vary depending on updates made from other pages.

ImmediateUpdate = 0 (default) or 1: When this option is inactive, the page is validated transactionally, that is the object is not updated with values modified in AttributeControls by clicking a button of the form (OK or Apply) or the wizard (Previous, Next or Finish. Otherwise the object is updated as soon as the AttributeControl is modified; in certain cases however this modification is effective only after a certain delay (Web Front-End) or when the modified AttributeControl loses focus.

IsSynchronized = 0 (default) or 1: If activated, indicates that the page is synchronized; in this case the conditions in its parameterization are evaluated with values in the form rather than with the effective values of the object attributes. This option is pointless if ImmediateUpdate = 1, since in this case these values coincide.

ActiveControl = 0 (default) or 1: When this option is activated, the validity control of an AttributeControl is executed at its modification; otherwise it is not executed until validation. This option is pointless if ImmediateUpdate = 1.

ContinueApply = 0 (default) or 1: If this option is activated, the page can be validated even if certain of its AttributeControls have detected errors. This mode can be indispensable in the case of a docked form without buttons, since in this case it can be impossible to close the form.
LabelTemplate = string: Enables redefinition of a page name. This string can contain a reference to an object of the system repository (in the form of a field), and in this case the name of this object will be used as the page name.

3.3.3 Synchronized pages - Immediate pages

3.4 Programmatic access to forms

This section describes possibilities of accessing page content by API code, so as to be able to specify a specific style on the form.

3.4.1 Executing a form by API

A form can be directly executed from APIs by the PropertiesDialog method. This method applies to the object that is the form object:

```
myObject.PropertiesDialog [option]{[,option]}
```

The following options are exclusive and enable determination of form content:

"PropPage=<Field>" displays the <Field> page in the form. By default, the ShowContextMenu option is deactivated.

"ViewPort=<Field>" displays the form of the object, conforming to PropertyPageViewPort <Field>. By default, the ShowContextMenu option is activated.

"Loader=<Field>initString" displays the form defined in the <Field> macro (see 2.3.2). By default, the ShowContextMenu option is deactivated.

The following options enable modification of form appearance or behavior:

"ApplyMode=Continue": enables continuous mode activation. In this mode, page modifications can be applied and the form closed, even if there are errors. This can be indispensable in docked mode.

"Immediate={0|1}": activates immediate mode; in this mode, application of the modification is instantaneous.

"Title=*caption*": enables modification of the form title.

"CheckAction={0|1}": enables display (1) or not (0) of the button for actions related to workflow of the form object (as appropriate) in the button/status bar.

"TabStyle={styles}": enables modification of tab display style (Windows Front-End) (see 2.3.2).

"ShowContextMenu{0|1}": enables activation (1) or deactivation (0) of display of object menu from the form title bar.

"HideStatus={0|1}": hides (0) or shows (1) the form button/status bar. If hidden, the form automatically passes to mode Immediate=1 – in absence of button it is not possible to apply modifications consistently outside this mode.

"ActivePage=*pageid*": enables definition of the initial page of the form.

"DefaultSize=*width,height*": enables definition of default size of the form (in pop-up mode).

"Position=*left,top*": enables definition of the position of the form in pop-up mode (Windows Front-End only).

As many arguments can be transmitted to the method as there are options. It is nevertheless possible to include several options in the same argument, separating these by a semicolon. The following two commands are therefore equivalent;

```
myObject.PropertiesDialog "ApplyMode=Continue;Title=my caption"
myObject.PropertiesDialog "ApplyMode=Continue" , "Title=my caption"
```

This is not however possible for the *Loader*= option, the *initString* string being totally free and can therefore contain semicolons.

The above options concerning form pages can be specifically redefined on each of the pages (see 3.3.2).

3.4.2 Access interface

3.4.2.1 Access to form content

You usually access interfaces relating to properties pages via a handler, that is via functions called at processing during execution of a form or wizard. It is however possible to directly access the component of a form using the PropertiesDialog function.

```
Set oList = oObject.PropertiesDialog([option]{[,option]}[,] "Description")
```

The string **"Description"** must be the last parameter.

The **oList** object returned is an enumerator enabling listing of form pages. The enumerated objects are **MegaPropertyPage** components.

3.4.2.2 MegaPropertyPage

The **MegaPropertyPage** enumerated above have the following properties:

.getId: GUID of the page (corresponds to the GUID MetaPropertyPage property of the page when it corresponds to a MetaPropertyPage).

.GUIName: page name that can be displayed in the user interface language.

.name: page name in the user language.

.Default: True if the page is the default page (active tab of form).

.IsTab: if True, the page in question is not a page with content, but a tab containing sub-pages.

.level: level of the page in hierarchy of tabs. If higher than 1, the page appears in a tab.

.ParentId: when the page is in a tab (level > 1), contains the identifier of the page containing the tab.

.sourceID: identifier of the MetaPropertyPage corresponding to the page. When the page is calculated, it does not correspond to a MetaPropertyPage, and in this case the identifier (if it exists) is the identifier of the system repository object that motivated appearance of the page; this is generally a MetaAttributeGroup.

.TypeID: identifier of _Type associated with the page.

.Component: when the page is a described page, returns the corresponding **MegaPropertyPageComponent** component. Otherwise returns *Nothing*.

3.4.2.3 MegaPropertyPageComponent

The **MegaPropertyPageComponent** object enables access to page content via an informal MegaObject (ie. it does not correspond to an occurrence of the MEGA repository) describing the page as an *AttributeControl*.

.Content enables access to this MegaObject.

.QueryVerb indicates if this **MegaPropertyPageComponent** implements the requested method. By this means, you can determine if it is a **MegaPropertyPageStandard**: in this case the **connect** function (for example) is implemented, and therefore QueryVerb(**"Connect"**) returns True.

3.4.2.4 MegaPropertyPageStandard

When the page is a standard page, the **MegaPropertyPageComponent** is therefore also a **MegaPropertyPageStandard**. It is using this component that wizard triggers access content of pages. It implements the following functions and properties:

Functions:

.Refresh(Optional reset as Boolean) requests the page to refresh. By default, this consists of requesting refresh of the AttributeControls it contains. If **reset** is present and is *True*, the page is totally recalculated. It is preferable to avoid using this option unless absolutely necessary, since the calculation can be costly and cause loss of contextual data.

At refresh of an AttributeControl:

- either the AttributeControl value was modified from the last application of updates: in this case the value is not modified, so as not to lose the entry made by the user.
- or the AttributeControl was not modified, in which case the displayed value is recalculated.

.Connect(connection As Object) enables association with the page of a component which can listen to notifications sent by AttributeControls.

.Disconnect(connection As Object) enables dissociation of a component associated by **Connect**.

.SetModify(val As Boolean) enables forcing of status of the page to be "modified" or "not modified" depending on the value *val*. In Windows Front-End in particular, the effect of this can be to "gray" or "ungray the "Apply" button: if after a SetModify(False) no form page is in "modified" state, the button will be grayed. It will be ungrayed at a SetModify(True).

.SetButton(button As String,Action As String) enables modification of appearance of the button of a form or wizard displaying the page. This function should be used with care, since buttons of wizards and forms are not identical:

For a wizard, buttons are named CANCEL, PROCEED, NEXT, PREV.

For a form, buttons are named CANCEL, OK, APPLY, HELP.

Possible actions are:

SHOW: shows button
HIDE: hides button
ENABLE: activates button

DISABLE: deactivates button
FOCUS: gives focus to button (if possible, and Windows Front-End only)
DEFAULT: defines the button as default button (Windows Front-End only)
TOCLOSE: applied to CANCEL button in a form, and in Windows Front-End only, renames the "Cancel" button to "Close" and deletes the "OK" and "Apply" buttons. To be used when an irreversible action has occurred (in this case it is no longer possible to cancel).

.OnCommand(Cmd as String) as String: enables simulation of sending a notification to the page. This function should only be used in specific contexts – for example to interact an updateTool with a dialog Handler (see 3.4.3 and 3.4.4). The command can contain a string in JSON format, and in this case the notification is considered as a DesktopCommand. It can also simulate a specific notification sent by an element of the page, in which case it is of form "ntfname:itemname". The value returned depends on the notification.

The following properties are accessible in read-only:

.getId As String: returns CLSID of the page. Equivalent to **MegaPropertyPage.getId**.

.getPageId: returns identifier of the MEGA object modeling the page. This is generally a MetaPropertyPage identifier, but not necessarily so.

.getRoot As MegaObject: returns repository root.

.template As MegaObject: returns the object on which the page was activated.

.isConnected As Boolean: indicates that the page has connections that can listen to notifications.

.labelTemplate As String: page title.

The following properties can be modified at form initialization (for example in a wizard), with the exception of statusMessage and isContinue properties, which can be modified at any time. If not, they can be consulted at any time. In most cases you can define these properties statically in page parameterization (see 3.3.2), but they can be overloaded in this way.

.transactional As Boolean: if True, the page is transactional and updates only at an explicit command (for example a click on "Apply" or "OK" buttons, or in a wizard "Previous", "Next" or "Finish" buttons. Otherwise, update is carried out at each modification of an AttributeControl.

.activeValidation As Boolean: if True, the validity control of the page is carried out at each modification of an AttributeControl. Otherwise, it is only carried out at validation. Pointless in the case of a non-transactional page.

.statusMessage As String: indicates to the page container an error message or status to be displayed, as appropriate.

.isSynchronized As Boolean: Indicates that the page must be synchronized, that is that conditions defined in its template apply to data assigned in AttributeControls, and not to effective value of the object associated with the form. Pointless in the case of a non-transactional page.

.isContinue As Boolean: indicates that the page accepts validation, even if certain of its AttributeControls detect an error.

The following properties are not accessible at processing of a notification (see 3.4.3)

.currentControl as MegaObject: returns the AttributeControl sender of the notification.

.currentSelectedItem as MegaObject: when the sender AttributeControl is a list, a tree, or more generally an element handling a list of objects, and when the notification in progress references an element selected in this list, then this element is accessible using this property.

.currentNotification as String: returns the name of the current notification.

3.4.2.5 AttributeControl MegaObject and associated model object

The object model of a page (and more generally of a form) can be accessed via a hierarchy of MegaObjects. These objects do not correspond to occurrences of the database. The list of properties, methods and collections defined for these MegaObjects is not defined in the repository, so they can be accessed by their name without risk of incompatibility.

3.4.2.5.1 Properties defined for this MegaObject

Nature (String): name of AttributeControl type corresponding to the element (see 3.5).

Kind (Integer): completes nature of AttributeControl. This is a bit field containing in particular the following values:

ReadOnly: 0x2000 (8192): indicates that the element is read-only.

Numerical: 0x0100 (256): indicates that the element corresponds to a numerical value.

WidthDefault: 0x0100 (2048): Indicates that the element manages display of default value.

Mandatory: 0x4000 (16384): indicates that the element is mandatory.

NoEdit: 0x0400: indicates for a composite element with a data entry zone (ComboBox, EditButton…), that this zone is read-only.

ResetOnRefresh: 0x0200 (512): This flag is for drop-down lists and elements displaying a menu, and indicates that the menu or content of the list can change if the element has undergone modifications, and therefore needs to be recalculated.

ValidateInput: 0x8000 (32768): indicates that the updated value in the element must be specifically checked when the page is in active validation mode.

ItemName (String): element internal name. This name is stable and unique and corresponds to the name of the element declared in the template. In the case of an implicit element, this name is based on the absolute identifier of the implicitly listed property. The name can be used when you are querying an element in a collection using the Search function. It is also used to build the name of notification functions relating to this element.

Style (Integer): bit field defining element style.

Styles relating to element docking:

```
CLIPLEFT      0x00000002   dock at left (default)
CLIPRIGHT     0x00000004   dock at right
CLIPTOP       0x00000008   dock at top (default)
CLIPBOTTOM    0x00000010   dock at bottom
CLIPMLEFT     0x00000020   dock at left and at middle
CLIPMRIGHT    0x00000040   dock at right and middle
CLIPMBOTTOM   0x00000080   dock at bottom and middle
CLIPMTOP      0x00000100   dock at top and middle
NOVCLIP       0x00001000   ignore vertical docking
NOHCLIP       0x00002000   ignore horizontal docking
BOTTOMALL     0x00080000   dock at bottom if element is last on page
```

Styles relating to element title:

```
NOTITLE       0x00010000   element without title
TITLEUP       0x00020000   element with title above
SETTITLE      0x08000000   reserved for RadioButtons, to calculate their size
```

Styles relating to positioning and size:

```
AUTOPOS       0x00000800   element position has been calculated
AUTOSIZE      0x00004000   element size can change dynamically and is calculated by
the element itself
SCREENUNITS   0x00008000   element size expressed in pixels and not in DialogUnits
BORDERED      0x00100000   element has border
```

Styles relating to element availability

```
DISABLED      0x00400000   element is deactivated
HIDDEN        0x00800000   element is hidden
MANDATORY     0x04000000   element entry is mandatory
```

Other styles

```
DEFAULTED     0x00040000   indicates that element manages default value
COMPUTEDHELP  0x20000000   indicates that element help must be calculated
ALWAYSUPD     0x10000000   indicates that element must be systematically validated,
even if it has not been explicitly modified

DYNAMIC       0x01000000   indicates that element visibility can change during data
entry
REMOTING      0x02000000   indicates that element can pilot other elements
```

ClipMode (String): property calculated from **Style** and describing docking mode. This string is empty if no docking is defined for the object, otherwise it takes the form:

"vertical":"<Vclip>","horizontal":"<Hclip>"

If vertical or horizontal docking is not defined, the corresponding element is not present in the string.

Vclip can be "TopToBottom", "Bottom", "Center", "TopToCenter", "CenterToBottom"

Hclip can be "LeftToRight","Right","Center","LeftToCenter","CenterToRight"

ID: element identifier. This identifier is based on the dynamic identifier and cannot be used to make information relating to this element persistent.

MapID: identifier of the Map in which the element is defined. This identifier allows you to find the Map in the collection associated with the parent AttributeControl of the element.

GrpID: identifier of the Group in which the element is positioned. This identifier allows you to find the Group in the collection associated with the parent AttributeControl of the element.

Name (String): element title in user language.

GUIName (String): element title in interface language.

HTMLTitle (String): element title in HTML format in interface language. This title can be "rich".

Order (Integer): element order umber, corresponding to the element dynamic identifier. It determines order of access to elements at browsing by tab key.

Width, Height, Left, Top (Integer): indicate position of the element relative to its group, as well as its size. This data cannot be specified if positioning is automatic (ie. not defined

explicitly) or if its size corresponds to default size. This data is supplied in DialogUnits (see 3.1) except in special cases.

TitleRect (String): when title position has been explicitly defined, this property contains a JSON indicating this position. The JSON takes the form:
{ "x" : x, "y" : y, "width" : w, "height" : h }
If one of these properties is not defined, it does not appear in the JSON.
If this property is not defined, the space occupied by the title is included in the size of the element.

SourceID: contains the absolute identifier defined for the element. Generally corresponds to the absolute identifier of the displayed property, and more specifically to the identifier associated with the element in the template by Item(<SourceID>).

ObjectID: identifier of the object with which the element is associated. This is the form object if the element is not in a Map; otherwise it is the object pointed by the Map.

Options (String): contains options defined for the element. This value generally corresponds to parameterization contained in the Param() field in the element definition.

This property can be modified: certain control types can adapt their behavior at modification of this parameterization.

Value (String): value displayed by the element. If the element does not explicitly a value - this is for example the case for a CheckBox or RadioButtons – it contains the value of the property associated with the element for the source object. For composite elements not proposing a consistent value, this property cannot be used for specific purposes.

This property can be modified.

TargetID: contains an absolute identifier relating to the value displayed in the element, when this has a sense, and in particular:

When the element is designed to display a MEGA object, or more generally an attribute of object type, TargetID contains the identifier of this object.

When the element displayed has an enumerated value attribute, TargetID contains the identifier of the enumerated value.

When an element is updated by value, TargetID corresponds to identifier null if this value has not been identified as an enumerated object or value. This comparison is only made at validation of the element.

This property can be modified.

Data (String): character string containing information relating to the current value of the element. Its content depends on the control type and is generally presented in JSON format.

This property can be modified.

Format (Integer): bit field specifying nature of the element value, and information enabling use or specification of this value.

FORMAT_IDABS 0x0001 value corresponds to an object identifier and must be synchronized with **TargetID**
FORMAT_TRANSLATION 0x0010 value is not in current language
FORMAT_NEUTRAL 0x0020 value is in "Neutral" language
FORMAT_USERLANGUAGE 0x0040 value (enumerated) is in user language
FORMAT_HASDEFAULT 0x0080 for an enumerated attribute, the value (empty) indicating an unvaluated property is replaced by the value (default)
FORMAT_NUM_FLOAT 0x0100 value is a floating comma number
FORMAT_NUM_SIGNED 0x1000 value is a signed number
FORMAT_NUM_DURATION 0x2000 value is a number representing a duration
FORMAT_NUM_PERCENT 0x2000 value is a number representing a percentage

FORMAT_NUM_EFFECTIVE 0x8000 value is an unformatted floating comma number

Read Only (Boolean) or short form **RO**: indicates that the element is read-only or deactivated.

Length (Integer): when the element displays a character string, indicates maximum string size. This property can be used to limit data entry or determine element size if this is not provided.

HelpKey (String): character string enabling determination of content of help associated with the element.

ErrorMessage (String): when the value of the element is invalid, or more generally an error has been detected related to this element, this string contains the corresponding error message.

IsDirty (Boolean): indicates that the element has been modifies and requires validation.

IsDefaultValue (Boolean): when the element manages default value, indicates that the current value corresponds to the default value.

IsInitialized (Boolean): indicates that the element has been initialized. Otherwise, the page has not yet been displayed and the values of properties that depend on the form object (for example Value, TargetID, Data) cannot be used.

Tick (Integer): numerical value incremented each time the element is modified.

Group, GroupGUIName (String): name of the group in which the element is located.

GroupOrder (Integer): order number of the group in which the element is located. Enables ordering of page elements.

ControlID (String): contains CLSID of the update Tool attached to the element.

3.4.2.5.2 Methods defined for this MegaObject

GetObject as MegaObject: object with which the element is associated. This is the form object if the element is not in a Map; otherwise it is the object pointed by the Map.

myCtl.GetObject.GetID corresponds to *myCtl.ObjectID*. We pass via this function when the element object is informal, since in this case we cannot use *GetObjectFromID* to access this object with its identifier only.

Search(itemIdent{,optional mapoption1, mapoption2 }) as MegaObject: When the element is a page or composite element, this function finds a sub-element according to its name (itemName).

It is also possible to find an element using its SourceID with an identifier (internal format) or a field as itemIdent argument. In this case several page elements can have the same SourceID, in particular if they are in different Maps. In this case you can specify the Map in which to search.

```
Search(sourceID, "MapID", mapID)
```

Otherwise, or more explicitly if you write:

```
Search(sourceID, "AnyMap")
```

the function returns the first element of the page with this sourceID.

You can recursively search for a sub-element from a composite sub-element by using an itemName path made up follows:

```
ItemNameMain>SubItemName
```

Refresh(optional Reset as Boolean) refreshes the element; if Reset, the element is recalculated.

Page as MegaPropertyPageStandard: when the element is defined in a Standard Page, this function returns the component corresponding to this page.

Component as Object: certain elements have an evolved access level, implemented by a specific component accessible via this function. For example, SubPages enable access via this function to the MetaPropertyPage component associated with the SubPage (when it is instanced).

3.4.2.5.3 Collections defined for this MegaObject

GetCollection("AttributeControl"): returns the list of sub-elements of the element.

GetCollection("Map"): when the element is a page (or SubPage), returns the list of maps defined on the page. These maps are described by an AttributeControlMap MegaObject described below. You can search for a map in this collection using the MapID property of an AttributeControl (**`mapCollection.Item(attcontrol.MapID)`**)

GetCollection("Group"): when the element is a page, returns the list of groups used in this page. These groups are described by an AttributeControlGroup MegaObject described below. You can search for a group in this collection using the GrpID property of an AttributeControl (**`mapCollection.Item(attcontrol.MapID)`**)

3.4.2.5.4 Main properties of AttributeControlMap MegaObject

ID: map identifier. This identifier is dynamic and cannot be used to make information relating to this element persistent. It corresponds to the MapID property of AttributeControls defined in this map.

Name as String: name of the map as defined and used in the template.

TargetID: absolute identifier associated with the map.

PilotID as String: when the map is piloted, contains the itemName of its pilot.

3.4.2.5.5 Properties of AttributeControlGroup MegaObject

ID: group identifier. This identifier is dynamic and cannot be used to make information relating to this element persistent. It corresponds to the GrpID property of AttributeControls defined in this map.

Name: name of the group as defined and used in the template.

Order as Integer: group order number, enabling definition of order of groups in the page.

GUIName: group title.

HTMLTitle: group title in HTML format.

SourceID: when the group title has been defined from a repository object, contains the identifier of this object.

Style as Integer: bit field indicating group style.

```
STYLE_BAR     0x10000       : separator not collapsible
STYLE_FRAME   0x20000       : frame, collapsible
STYLE_ALWAYS  0x40000       : separator present, even if in first position
STYLE_CLOSED  0x80000       : group is initially closed
```

Size as String: contains initial size of this group when defined, otherwise returns an empty string. The returned string corresponds to the following format:

```
"width" : w, "height" : h
```

If one of these two values is not defined, it does not appear in the string.

3.4.3 Dialog Handler

The dialog Handler system enables improvement of specific interaction possibilities on a properties page by allowing you to add interactions at events occurring when editing the form. These interactions are made using an interaction component implemented by a macro, and connected to the page.

Connection can be:

- either declarative in parameterization of the PropertyPage (see 3.3.2)

```
[Dialog] CommandHandler=<MacroID>
```

- or by code - in wizards – using the Connect function of the MegaPropertyPageStandard component (see 3.4.2.4)

When the macro is connected, it can carry out processing on notifications from AttributeControls of the page, or on events relating to the form or page.

3.4.3.1 Notifications

```
Sub On_ctlName_ntfName(Page as MegaPropertyPageStandard)
```

When it is implemented, this method is called when the AttributeControl of which the itemName property is ctlName sends the notification ntfName.
At this call:

Page.currentControl corresponds to the notification sender element. Therefore *Page.CurrentControl.ItemName* corresponds to ctlName

If the sender manages an object list and if the notification concerns the element selected in this list, *Page.currentSelectedItem* corresponds to the selected object.

Page.currentNotification corresponds to ntfName

If this method is not implemented, the OnCommand method (see below) is then called.

```
Sub OnCommand(Page as MegaPropertyPageStandard,ntf as Integer,ctl as Integer)
```

This method is called each time a notification sent by an AttributeControl of the page is not specifically processed by a method.

ntf is a notification identifier number. Standard notifications are named and their name is accessible via Page.currentNotification.

ctl corresponds to the Order property of the sender AttributeControl. It can change if the page is built dynamically, or if elements can be hidden and the page is synchronized or in immediate mode. For these reasons, it is preferable to use Page.currentControl.itemName, corresponding to the name of the element as defined in the template, to determine the notification sender.

At this call:
Page.currentControl corresponds to the notification sender element.
If the sender manages an object list and if the notification concerns the element selected in this list, *Page.currentSelectedItem* corresponds to the selected object.

3.4.3.2 Other events

```
Sub PropertyPage_Initialization(Page as MegaPropertyPageStandard)
```

This method is called at page initialization.

```
Function CheckPropertyPage(Page as MegaPropertyPageStandard) As String
```

This function is called before each page validation. It allows you to indicate an error preventing page validation. In case of an error, this function returns an explanatory error message. If there is no error, it returns nothing or an empty string.

In case of error, validation is cancelled and no update is carried out.

```
Function OnPropertyPageApply(Page as MegaPropertyPageStandard) As String
```

This function is called after each page validation. In particular, it allows you to carry out supplementary updates. It is possible to indicate an error preventing page validation. In case of an error, this function returns an explanatory error message. If there is no error, it returns nothing or an empty string.

In case of error, if the page is not in continuous mode, if we are in:

- a form, it is not closed,
- a wizard, the transition is stopped and we remain on the page.

```
Sub PropertyPage_OnDesktopCommand(Page as MegaPropertyPageStandard, JSONCommand
As String)
```

This function is specific to Web Front-End and enables sending of a notification from Desktop to the form or wizard, if scopes of tools will allow this. The notification is only sent on the active page of the tool.

3.4.4 UpdateTools

An UpdateTool is a component designed to manage update of a property (MetaAttribute, _AbstractProperty or MetaAssociationEnd seen as an attribute).

This component is only invoked through an AttributeControl, whether in a form, a wizard or an in-place data entry; it is not designed to parameterize display of an attribute in a regeneration.

An UpdateTool enables:

- definition of the AttributeControl to be used to update the property.
- definition of options and complements of this AtrributeControl, as could be done in page template configuration.
- definition of content of the drop-down list, when the attribute type includes one.
- if appropriate, definition or completion of the menu and processing of commands, or implementation of button processing.
- definition of the value effectively displayed by the element.
- definition or completion of validation of the element, in particular update of the repository induced by this validation, and application of specific controls.

An updateTool cannot be associated with a varchar type property (text). Updates of these properties are defined by their associated _type.

3.4.4.1 Implementing an updateTool for a property

An updateTool is implemented by a Macro. To be recognized as such, this component must contain the function.

```
Function AttCtl_GetDefaultKind() As String
```

This macro should be associated with the property according to the property type. If it is:

- a MetaAttribute, the MetaAssociationEnd "*MetaAttributeUpdateTool*" should be used.
- a MetaAssociationEnd *_LegAttribute*, the MetaAssociationEnd "*_LegUpdateTool*" should be used.
- an *Abstract Property* (for example a *TaggedValue*), the MetaAssociationEnd "*Update Tool*" should be used.

3.4.4.2 Defining AttributeControl type

The first role – the only mandatory role – of an updateTool is to define the type of AttributeControl that will be used to update the property.

- If this type does not depend on the edited occurrence, and if it is not necessary to define options, you can simply implement the function **`AttCtl_GetDefaultKind()`**mentioned above.
- Otherwise, you must add the function to it:

  ```
  Function AttCtl_GetKind(Context As MegaUpdateToolContext) As String
  ```

  Unlike the previous case, this function enables different behavior according to the edited object, and specification of options forAttributeControl, using the context component **MegaUpdateToolContext**.

These two functions should return a character string of form:

```
<ControlName>{:<option>}{,<option>}
```

ControlName is the name of the AttributeControl type (see **Error! Reference source not found.**)

Possible options are:

- type complements such as lists in the **Kind** property of the AttributeControl MegaObject (see **Error! Reference source not found.**),
- options that specify the AttributeControl use context:

ManageReadOnlyMenu: in the case of an EditMenu, indicates that the updateTool takes into account the fact that the object or property is read-only and adapts the menu or its behavior to this fact. By default, the framework considers that read-only is not managed and specific commands of the updateTool are not inserted in the menu.

ManageValueID: in the case of an attribute with enumerated values, indicates that the TargetID property, systematically corresponding to the enumerated value absolute identifier, is perhaps used to update the attribute when it is of the corresponding type.

SingleOnly: indicates that the AttributeControl does not operate in multiselection mode and that attribute entry should be deactivated in this case.

In the AttCtl_GetKind function it is also possible to modify AttributeControl style and options.

3.4.4.3 MegaUpdateToolContext component

MegaUpdateToolContext enables access to updateTool context. It includes:

- the following objects:

  .MegaObject As MegaObject: occurrence to be modified. In the case of an entry derived from MultiSelection, this is a calculated MegaObject aggregating the collection of selected objects.

  .AttributeID As Variant: attribute identifier (resp. Abstract Property or MetaAssociationEnd) to which the updateTool relates.

  .GetRoot As MegaRoot: access to properties page MegaRoot.

  .AttributeControl As MegaObject: returns the component corresponding to the element controlled by the updateTool. This component is AttributeControl type (see **Error! Reference source not found.**) and provides direct access to the page element, as well as the page itself (where appropriate).

  .ComboListCollection As MegaCollection: when the element manages a drop-down list, this collection corresponds to the list of elements appearing in the list. It is designed to be populated by the updateTool in the *AttCtl_FillCombo* function (see below).

- the following properties:

  .EditText as String: value displayed when the element contains a text area (in editing or read-only). This property can be modified and corresponds to *.AttributeControl.Value*.

  .IsInPlaceMode As Boolean: is True if the element is invoked in the context of in-place entry. This property can be accessed after object initialization: The type of element displayed can differ if you are in in-place mode.

  .IsMultiSelection As Boolean: is True if the element is invoked in the context of multiselection entry. This property can be accessed after object initialization, for example when the updateTool does not know how to manage multiselection and in this case wants to force mode to "read-only".

  .ValueID: where appropriate, contains an identifier corresponding to displayed value, corresponding to *.AttributeControl.TargetID*; this identifier can correspond to the identifier of an object (when the property is object type) or to that of a MetaAttributeValue (when the object is enumerated). The updateTool may have to explicitly manage the displayed value (*EditText*) and the identifier pointed (*ValueID*), in particular when implementing a command modifying the value of the element. This is the case for elements of object type, for which it is highly desirable to have the identifier if known, as well as the name to be displayed. It should be noted that modification of *EditText* results in invalidation of *ValueID*: The latter should therefore be updated after *EditText*.

  In the case of Web Front-End, update from a combined list does not update the value of this variable if the property concerned is not of Object type.

  .ValueTypeID: When updateTool implements a generic MetaAssociationEnd, attribute or object type abstract property, you can use this property to consult or update the MetaClass type of the target object. Where appropriate, this property can be used to create the standard menu of the element.

- the following functions:

.Invalidate: notifies that the element has been modified. Among other things, this enables ungraying of the "apply" button in properties pages.

.SetDirty: should be used at menu command or button press processing, when you want to indicate to the element that the current value should be considered as having been modified and therefore that the element should be updated.

.Recompute(Item as String): invalidate the menu (if Item contains "Menu") or drop-down list (if Item contains "DropList") of the element and results in menu recalculation at the next invocation (ie. click on menu or drop-down list button. It is possible to transmit ("Menu,DropList") as parameter.

.AddAccelerator(key as String,cmd as Integer{,altkey as String}): (Windows Front-End only) can be used in the *AttCtl_GetAccelerators* function of the updateTool, to define accelerators (key combinations triggering actions when the element has focus).

key corresponds to the keyboard key,

altkey indicates the combined keys and can contain values "ALT", "SHIFT" and "CONTROL" (value "ALT,CONTROL" - for example – is also included) ;

cmd is the number identifying the command and transmitted in the *AttCtl_AcceleratorInvoke* function when the corresponding accelerator is activated.

3.4.4.4 Generic functions

These functions concern initialization, display and update of an element. If they are present in the updateTool, they are invoked whatever the element type.

```
Function AttCtl_SetText(Context,editText) As Boolean
```

This function is called at supply of the element, and can be used in particular when you want to display on the element a value not corresponding to the effective value of the property.
The editText argument corresponds to the value that will be displayed if the updateTool does not carry out any processing. This argument can be modified, and if the function returns True, then the text display area is updated with the new value.

```
Function AttCtl_SetDefaultText(Context,editText) As Boolean
```

This function is called when you want to display the element default value (for elements with default value view system). Its operation is identical to that of AttCtl_SetText.

```
Function AttCtl_UpdateText(Context,editText) As Boolean
```

This function is called after element validation, when we are preparing to display the new value after element update. Its operation is identical to that of AttCtl_SetText.

```
Functon AttCtl_UpdateCheck(Context,Status,ErrorMessage) as Boolean
```

This function is called at element validity check – when the page is in "*ActiveValidation*" mode. It should not execute any update.
In the case where the check detects an error, this error should be documented by updating the *ErrorMessage* argument with a character string explaining the error; this string will then be

displayed to the user. If there is no error, an empty string should be returned, or the argument not modified.
The Status variable contains the return code, which in this case can be:

STATUS_ERROR (1): an error has been detected. If ErrorMessage is not empty, then STATUS_ERROR is automatically activated.

STATUS_NOCHANGE (2): indicates that the element will not result in property update.

If this function does not return value FALSE, standard validity checks of the property are called.

```
Function AttCtl_Update(Context,Status,ErrorMessage) as Boolean
```

This function is called at element validation; it should execute validity checks, then update the property – except if element default update is suitable.
In the case where the check detects an error, this error should be documented by updating the *ErrorMessage* argument with a character string explaining the error; this string will then be displayed to the user. If there is no error, an empty string should be returned, or the argument not modified.
The Status variable contains the return code, which in this case can be:

STATUS_ERROR (1): an error has been detected. If ErrorMessage is not empty, then STATUS_ERROR is automatically activated.

STATUS_NOCHANGE (2): indicates that the element will not result in property update.

STATUS_OK (4): the property has been correctly updated (in this case False should be returned so that standard update is not executed).

If this function does not return value False and status is not STATUS_ERROR, property standard update is called.

3.4.4.5 Specific functions implemented by UpdateTool

```
Function AttCtl_FillCombo(Context,ComboListColl) as Integer
```

If present, this function is called to populate the drop-down list of an AttributeControl.
To do this:

➔ Add in ComboListColl collection the values you want to include in the drop-down list.

This collection is a collection of enumerated Values, corresponding to the type used in MegaObjects accessing the compiled Metamodel, as used in particular by the GetPropertyDescription(<propID>).Values collection.
These MegaObjects have in particular the properties:

Rpbid: identifier in base 64. For properties of object type, this identifier is assigned at selection in the drop-down list to **ValueID**, and is therefore used to update the property at validation (except of course if validation is specifically handled by the updateTool). Otherwise, where appropriate, this is the identifier of a MEGA enumerated value.

InternalName: internal value, used as standard to execute update for attributes not of object type. For an object type attribute, this can be a field referencing the listed object.

GUIName: name displayed in the drop-down list.

So that elements will be effectively integrated in the list, they can be:

- MegaObjects representing enumerated values obtained from a description. It is only in this case that you can display bitmaps, derived from modeling of enumerated values (note: there are no bitmaps in ComboEditMenus)

- MegaObjects explicitly created in the collection, not directly corresponding to repository objects, and not outliving this collection.

Return value can take the following values:

0: call default processing: typically, the default enumerated list corresponds – for the enumerated properties – to the following code

Context.getRoot.getPropertyDescription(Context.AttributeID).Values

For _legAttributes, the default code supplies the drop-down list by means of favorite candidate query if it has been defined.

1: display list from collection - case of a standard enumerated property

2: display list from collection, taking account of absolute identifier of the enumerated value - applicable to properties of object type

-1: as 1, if you want to display (Default) rather than (Empty) to indicate empty value in the combo

-2: as 2, if you want to display (Default) rather than (Empty) to indicate empty value in the combo

Note: if there is inconsistency between the return value and the property type, behavior risks being not as expected. In Windows Front-End, an error will be activated if the properties pages debugging option has been activated.

The following example enables management of an object type property of which the target is compatible with the 'Org-Unit' MetaClass. The list is supplied with all repository org-units. Note that this code is similar to standard code supplying a drop-down list from a favorite candidate query (in this case the query is used in GetCollection in place of the "Org-Unit" MetaClass).

```
Function AttCtl_FillCombo(oContext As MegaUpdateToolContext,oFillCollection As MegaCollection) As Integer
  Dim oOrgUnit
  for each oOrgUnit in oContext.MegaObject.GetRoot.GetCollection("OrgUnit")
    Dim oAdded
    Set oAdded = oFillCollection.Create(oOrgUnit.GetID)  ' assignment of absolute identifier of org-unit to value created.
    oAdded.GUIName = oOrgUnit.ShortName
    oAdded.InternalName = oActeur.MegaField()  ' warning, internal value should not exceed 20 characters… We can put simple counter in this case since in the case of an object the internal value is not used
  Next
  AttCtl_FillCombo = 2
End Function


Function AttCtl_ImplementsMetaCommand(Context) As String(or Integer)
```

Presence of this function indicates that the updateTool will handle all or part of the AttributeControl pop-up menu – and therefore applies only to elements that have such a menu.

It is first possible to redefine the *capability* applicable for the element (see **Error! Reference source not found.**). To do this, the function can return either a numerical value corresponding to the required *capability*, or a character string containing keywords **LINK**, **LIST**, **SEARCH**, **NEW, EMPTY** to activate the corresponding capabilities **Integrate link specification** (0x20), **List** (0x1), **Search** (0x2), **Create** (0x8), **NoEdit** (0x1000).
It is then possible to add specific commands in this Popup menu by implementing in the updateTool the functions allotted to MetaCommandManager, in particular:

```
Sub CmdCount(obj,count)
Sub CmdInit(obj,num,name,category)
Sub CmdInvoke(obj,num)
```

On calling these methods, the updateTool context is made available to the macro implementer by the *AttCtlContext* property of *PropertyBag* of the global MegaMacroData

This system enables insertion of specific menu commands. It does not however enable insertion of pop-up sub-menus corresponding to object menus as the standard ChildMenu does; this possibility is offered by a specific system. You should:

1. Define capability in the form of a character string and insert in it the keyword **SPECIFICCHILDMENU.**
2. Implement in the updateTool the following function:

   ```
   Function AttCtl_GetMenuObject(Context,Indice,Name As String) As MegaObject
   ```

   This function should return the MegaObject of which you want to display the menu.

   It is possible to specify *Name* with the pop-up menu label; if this is not done, the pop-up will have as label the name of the MegaObject description.

   Several object sub-menus can be inserted; the function is called as many times as it returns a MegaObject different from the previous one; the Index argument is incremented at each call (initial value is 1).

The following is an example of updateTool displaying a menu, applicable to a MegaIdentifier type property.

```
'MegaContext(Fields,Types)
'Uses(Components)
Option Explicit

Function AttCtl_GetDefaultKind()
AttCtl_GetDefaultKind = "ComboBoxMenu:ValidateInput"
End Function

Function AttCtl_ImplementsMetaCommand(AttCtlContext As MegaUpdateToolContext)
AttCtlContext.ValueTypeID = "~BEy8SnY(yKk0[City Planning Area])"
AttCtl_ImplementsMetaCommand = 7
End Function

Sub CmdCount(obj,count)
count = 3
End Sub

Sub CmdInit(obj,num,name,category)
name = "Command " & num
category = 4
End Sub

Sub CmdInvoke(obj,num)
Dim AttCtlContext as MegaUpdateToolContext
Set AttCtlContext = MegaMacroData.GetBag.AttCtlContext
  Dim oR
  Set oR = AttCtlContext.MegaObject.GetRoot.GetCollection("Acteur").SelectQuery( -
           "Invoke Command #"  & num & " on Attribute " &  -
           AttCtlContext.AttributeControl.Page.GetID,True)
if oResult.Count = 1 Then
AttCtlContext.ValueID = oResult.Item(1).GetID
AttCtlContext.EditText = oResult.Item(1).ShortName
```

```
        end                    if
End Sub
```

Also applicable in the case where an AttributeControl can give access to a PopupMenu, it is possible to define and manage accelerators, that is keys able to execute commands when pressed while AttributeControl has focus; these accelerators are only currently available in Windows Front-End:

```
Function AttCtl_GetAccelerators(Context)
```

This function enables definition of accelerators. To do this use the **AddAccelerator** function of *Context*. The return value of this function is not significant, and in VbScript a Sub can be used. When an accelerator has been defined, the following function is called to activate it:

```
Function AttCtl_AcceleratorInvoke(Context,cmd) as Boolean
```

Return value of this function is optional and, if specified, indicate the the AttributeControl has been modified (value True) at accelerator call.

```
Function AttCtl_OnPush(Context) as Integer
```

This function is called when a button is clicked (if this button does not open a pop-up or drop-down). For controls of "Button" type, this function is only called if the "NoCall" option is defined. It is also called for an Image type control, if the action has not been deactivated (by option Click=Inactive) or handled by a menu (Click=PopupMenu). In this case the data field of the AttributeControl can contain click coordinates in form "cursorPos" :"line,col" .

If return value is not null, this indicates that the updateTool has processed the command. If it is positive, this indicates that the element has been updated by the command.

3.4.4.6 Processing notifications sent by the element

When an AttributeControl is modified or must handle events, it can notify the updateTool if the latter implemented the function

```
Function AttCtl_OnCommand(Context,ntfCode,item) as Boolean
```

item is a numerical value indicating notification context, and *ntfCode* the notification code, when context is an interface element. In Windows Front-End, notifications relating to elements are retransmitted to the updateTool, and Windows documentation contains the list. In Web Front-End, only certain notifications are sent, interactions being less detailed. Only notifications valid in Web Front-End are listed below. Note here that there can be significant differences between Web Front-End and Windows Front-End:

- in Web Front-End, you do not send detailed interactions specific to internal operation of Windows.
- successive notifications can be sent in a different order.
- in Web Front-End, it is not always possible to interact with the rest of the form at an internal notification – in particular Page.Refresh will only operate the notification comes from an effective client command.

For this, it is recommended to act only on clearly identified notifications and to be aware – if appropriate – that the code may not operate in Web Front-End.

3.4.4.6.1 Internal notifications

These notifications are not necessarily sent at an interaction. They can also be sent at page creation. This is why during their processing, execution of actions using the rest of page content

should be avoided (for example refresh operations), the rest of page content not necessarily being operational at the time of notification.

Item values for these notifications are:

- **ITEM_INIT** (0x8000 – 32768) sent at element initialization.
- **ITEM_SET** (0x8007 - 32775) sent when the element value has been modified, either at a direct interaction or by code.
  ntfCode is 0 if the element is reinitialized (not specified)
- **ITEM_READONLY** (0x8006 - 32778) notification sent when element read-only property has been changed – This notification is only currently effective for EditMenus.
  ntfCode is 1 if the element is deactivated.

3.4.4.6.2 Notifications resulting from user action

Item values for these notifications are:

ITEM_DROPDOWN (1) notification concerning element drop-down list. In this case, notification code to be used is:

CBN_SELENDOK (9) This notification is sent when a new element has been selected from the drop-down list

ITEM_MENU (2) notification concerning element pop-up menu. In this case, notifications to be used are:

MNU_DROPDOWN (7): sent before pop-up menu display
MNU_ENDOK (9): sent after menu display, when a command has been executed.
MNU_ENDCANCEL (0xA – 10): sent after menu display, if no command has been executed.

ITEM_EDIT (3) notification concerning edit area. In this case, notification code to be used is:

EN_CHANGE (0x300 – 768) – sent when content of the area has changed.

ITEM_BUTTON (4) notification concerning button. In this case, notification code to be used is:

BN_CLICKED (0) – button has been pressed.

ITEM_TITLE (5) should not normally be used.

3.4.4.7 Multiselection MegaObject

When an updateTool is instanced in a multiselection framework, it has (context.MegaObject) a virtual MegaObject calculated from the list of selected objects.

In addition, the Context.IsMultiSelection property is True.

On this object, the GetProp() function calculates the property value according to that of the listed objects: if a value is identical on all objects, then GetProp()returns this value. Otherwise,

the value is considered as unspecified. Unique or indexed properties (such as the absolute identifier for example) are not therefore significant for the multiselection object.

Regarding the SetProp() function, it applies the update on all objects of the selection; it is therefore not possible to update a unique index.

If the updateTool implementer needs to explicitly know the list of selected objects, this can be obtained from the multiSelection MegaObject using the ad hoc collection megaObject.GetCollection("~p1qjEch)GnxA[SelectedObjects]")
However, other collections accessible from this MegaObject are empty (in particular they are not created from collections of selected objects, as properties are).

3.5 AttributeControl types

This section covers the different control types that can be displayed in a page. For each of these controls, the following is explained:

- Parameterization options, which can be specified in the Param() keyword.
- Use of properties and, as appropriate, collections and methods of the MegaObject corresponding to this type of element.
- Explicit notifications sent by this type of element.

3.5.1 Implicit AttributeControls

These types are automatically associated with properties depending on their type and format. Under certain constraints, it is possible to redefine the control displaying a property.

Unless otherwise indicated, all the elements listed below are presented preceded by their title.

3.5.1.1 Edit

This type is the default type used for a modifiable property. Its default size is adjusted to the declared length of the property and the number of characters that can be entered.

The value displayed is the external value of the attribute.

Properties:

Value corresponds to the displayed value.

Notifications sent:

Change: sent when edit zone content has been modified.

3.5.1.2 Static

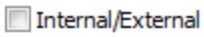
This type is the default type used for a read-only property. Its default size is adjusted to the declared length of the property.

The value displayed is the ''display' value of the attribute.

Properties:

Value corresponds to the displayed value.

3.5.1.3 CheckBox



Default type for Boolean properties. By default, it has the NOTITLE style, and the title value is displayed in the static zone located after the button; default size of the element is calculated depending on the size of this title.

By default, this control uses the internal value of the property to execute update; to do this, it considers that this internal value is the number 1 for check value and 0. This control therefore operates natively with boolean type properties, short et long. In other cases, correct operation can only be assured with a *CheckOn* option.

This control has only two values, and does not distinguish unspecified properties from 'False' properties (which correspond to value 0). If you wish to distinguish these values, use 3StateCheckBox type.

Option:

CheckOn=<UnCheckValue>/<CheckValue>: This option enables display of a property with only two valid values in the form of CheckBox, and this whatever its format. It is particularly adapted to simplify entry of attributes with two enumerated values. To do this, specify (in ASCII format) the value playing the "uncheck" role and the value playing the "check" role.

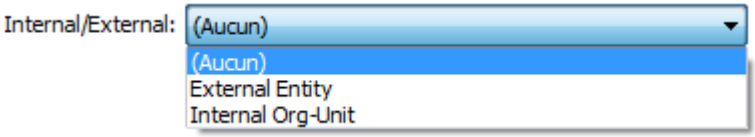
Properties:

Value: corresponds to ASCII value "0" or "1".

Notifications sent:

Click: sent when we click the box.

3.5.1.4 DropDownList



Drop-down list, used as standard for closed list enumerated properties. This element does not therefore allow entry of a value, which can only be modified from a drop-down list.

When the value of the property does not correspond to an available enumerated value (this can also occur if the enumerated values are filtered), the effective value of the property is nevertheless added to the drop-down list content. In principle, it should be possible to activate an entry without modifying the previous value, and this in any circumstances.

When the property is mandatory, the value (None) is not proposed.

The displayed value corresponds to the external value of the property; this value is normally presented in the user interface language, but it can be parameterized so that this display is in the current language by activating the "Keeps user language" indicator (0x40) in its 'Extended properties' MetaAttribute.'

Default size of this zone is calculated according to the size of the external values of the list of enumerated values.

Properties:

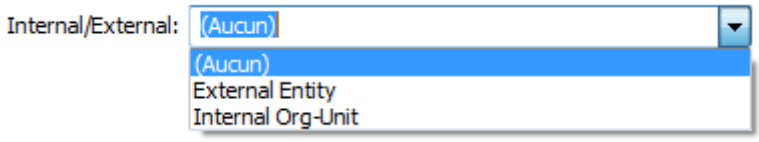
Value: displayed value.

TargetID: absolute identifier of the enumerated value corresponding to the displayed value.

Notifications sent:

SelEndOK: sent when a new value has been selected in the list.

3.5.1.5 ComboBox



Drop-down list combined with edit zone, used as standard for open list enumerated properties. The value presented in the edit zone does not necessarily correspond to an enumeration value.

It is possible to directly enter the value to be modified in the edit zone; when it corresponds to an external or internal value of the enumerated values declared on the property, the physical update is executed with the internal value of the latter. The language used for external enumerated values can be parameterized by the "Keeps user language" indicator in their 'Extended properties' MetaAttribute (see 3.5.1.4).

Default size of this zone is calculated according to the size of the external values of the list of enumerated values, and the size of the attribute itself.

Properties:

Value: displayed value.

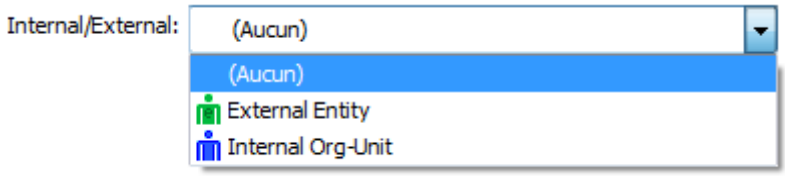
TargetID: absolute identifier of the enumerated value corresponding to the displayed value, if applicable. When the displayed value has been directly modified in the edit zone, and as long as the zone has not been validated, this value is null. It is at validation, most often at update, that comparison is made between an enumerated value and this entered value.

Notifications sent:

SelEndOK: sent when a new value has been selected in the list.

Change: sent when the edit zone has been modified.

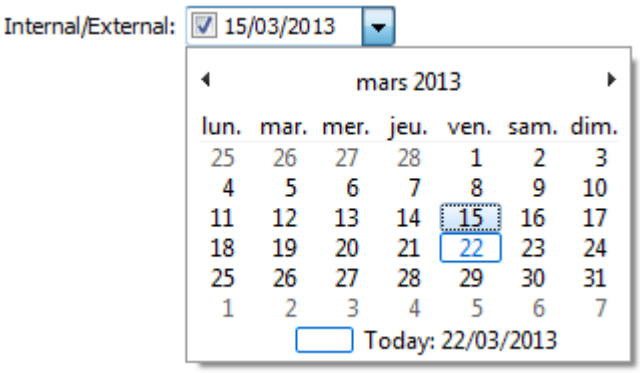
3.5.1.6 ComboBitmaps



Drop-down list associated with an image. This type of element is associated with enumerated properties when the "Activate images" indicator (0x4000) has been activated in their 'Extended properties' MetaAttribute.

Operation is similar to a DropDownList (3.5.1.4). The image displayed is defined in the enumerated value.

3.5.1.7 DatePicker



Edit zone associated with date entry via a calendar. This type of element is associated with properties of date type.

The date is displayed in external format.

This element can be used to edit times using the *TimeFormat* option.

This element operates correctly only with Date type properties.

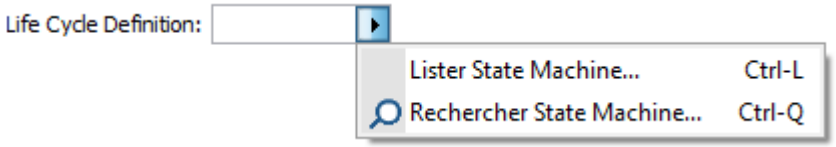
Option:

TimeFormat: entry of a time.

Properties:

Value: value of date entered in ASCII format (GMT yyyy/mm/dd hh:mm:ss)

3.5.1.8 EditMenu (and other controls designed for object type properties)



Edit zone associated with a button enabling opening of a pop-up menu. This type of element is associated with properties of object type (including in particular _legAttributes). These properties can also be associated with StaticMenus, ComboBoxMenus, DropDownListMenus, even ComboBoxes or DropDownLists, depending on parameterization of their capacity, as well as the definition of candidate queries. This parameterization also enables definition of menu content.

The edit zone displays as standard the short name in *display* format of the object pointed by property when it exists; otherwise the ASCII value of the absolute identifier pointed is displayed.

3.5.1.8.1 Specification of update capacity and possibilities

Capacity (Capacity) is a MetaAttribute defined on the _AbstractProperty MetaClass and the _LegAttribute MetaAssociationEnd enabling piloting at metamodel level of behavior of the modification tool concerned. This attribute is a bit field for which the following values are defined:

- **List (1)**: presence of "list" menu command. If the entry zone has been modified, List presents only those objects with names beginning with the value entered in the edit zone.
- **Search (2)**: presence of the "search" menu command, which enables object selection using the MEGA standard query tool.
- **ChildMenu (4)**: presence of the menu of the associated object as a sub-menu.
- **Create (8)**: presence of "create" menu command.
- **NoEdit (4096 - 0x1000)**: the edit zone is read-only. If the element is not required, the "delete" menu command is added.
- **Integrate link specification(default) (32 - 0x20)**: in the case of a _LegAttribute, specifications defined on the MetaAssociationEnd relating to an entry, and in particular candidate definition, are taken into account for presentation of the element.
- **Ignore link specification (16 – 0x10)**: in the case of a _LegAttribute, specifications defined on the MetaAssociationEnd relating to the entry are ignored.
- **AutoCreate (256 – 0x100):** when the user has modified the edit zone of the element, and therefore the element does not make explicit reference to an existing object, validation of the element results in creation of an object whose name corresponds to the entered value.
- **AutoList (128 – 0x80):** when the user has modified the edit zone of the element, and therefore the element does not make explicit reference to an existing object, but the name entered in this zone can correspond to repository objects, the list of these objects is proposed at validation.

When taking into account of MetaAssociationEnd entry specification has not been deactivated, we search on this latter:

- value of MetaAttribute LinkOptions. If this value is odd (bit 1 activated), the Search menu is hidden.
- the list of Candidate queries which have been associated with it, as well as the value of the "Favorite" MetaAttribute defined on this link. If we find a favorite Candidate query (that is with value "yes" in the "Favorite" MetaAttribute), the control presented displays a drop-down list button enabling listing of occurrences defined by the favorite candidate query. The other candidates will be the subject of specific menu elements: if we select these menu elements, the corresponding query will be executed, enabling object selection.

When the property cannot be modified, or when user rights on MetaAssociationEnds are restricted, the menu and the capacities used are correspondingly adapted; if the zone cannot be modified, only the ChildMenu will be presented as appropriate.

The type of element presented therefore depends on these specifications, and in particular on:

- the presence of a menu enabling selection or consultation of the object presented in the element.
- the definition of a favorite candidate query, which will display drop-down list behavior (and therefore possibly a second button).
- the fact that the edit zone is read-only or not.

Menu	*favorite query*	*read-only*	*control type*
Yes	No	no	**EditMenu**
Yes	No	yes	**StaticMenu**
Yes	Yes	yes	**ComboBoxMenu**
Yes	Yes	no	**DropDownListMenu**

No	Yes	yes	**ComboBox**
No	Yes	no	**DropDownList**

3.5.1.8.2 Target MetaClass

The behavior and appearance of the element menu depends on the MetaClass of the property target object. In the case of a MetaAssociationEnd, this MetaClass corresponds to the opposite MetaClass. In the case of a MetaAttribute of Object type, you can define this MetaClass using the "*Referenced MetaClass*" MetaAssociation. In the case of an *Abstract Property* of Object type, it is the MetaAssociation (*Property Object / MetaClass*) that enables definition of this MetaClass.

When the target MetaClass is defined, the search, listing and query tools will be confined to this MetaClass.

If the target MetaClass has not been defined in the metamodel of the displayed property, it can be specified in the parameterization of the element using the Target option. This option also enables specialization of an element to a specific concrete MetaClass, in this case a generic MetaAssociation.

If the target MetaClass is not defined, we consider that the object can be of all MetaClasses of the repository. In this case, the AutoCreate and AutoLink capacities cannot be effectively used.

3.5.1.8.3 Editing name and automatic behavior

When the editMenu zone is not read-only, the user can enter a name in this zone. This enables:

- specification of the search key of the "list" command.
- initialization of the object name in the "create" command.
- update execution by finding the object whose name corresponds to the entered value.

This name can however be ambiguous. On the one hand it is assimilated in the short name of the object, and can therefore correspond to several objects when the target MetaClass has a namespace. on the other, when the target MetaClass is abstract, objects of different MetaClasses can have the same names.

Validation of an entry zone in this way has therefore been globally confined to concrete target MetaClasses without namespace; when there is risk of ambiguity, zone validation is refused and the element is considered as invalid.

It is however possible to alleviate this behavior using AutoLink and AutoCreate capacities. AutoLink enables removal of ambiguity by proposing at zone validation a dialog box listing all objects corresponding to the name entered in the element; update is executed with the object explicitly selected in this list. AutoCreate enables automatic object creation if there is no object corresponding to the selected name. It is preferable to combine AutoLink and AutoCreate so as not to create the object until possible ambiguities have been resolved.

3.5.1.8.4 Options

It is possible to redefine all or part of the capacities defined in the metamodel using options on the element:

Target=<MetaClassId> enables redefinition or specification of the target MetaClass.

Capabilities=<HexaNum> enables redefinition of capacity of the property. HexaNum is understood as a hexadecimal number.

DefaultList=<QueryId> enables definition or redefinition of the query to be used to supply the drop-down list, as appropriate. This query is not necessarily a candidate.

3.5.1.8.5 Properties

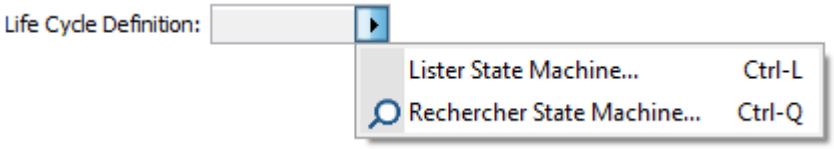
Value: displayed value. It corresponds, if its content has not been modified by the user, to the short name of the object pointed by the element in *Display* format. If the pointed absolute identifier does not correspond to an existing object of the repository, **Value** contains the ASCII value of this identifier.

TargetID: absolute identifier of the object pointed by the element. When the user modifies content of the edit zone, this property is deleted. When a menu command allowed explicit selection of a new object, this property contains the identifier of this new object. When capacity allows, the sub-menu relating to the object menu (ChildMenu) corresponds to the repository object pointed by this property. When specified, this property is used to validate the element. Otherwise, and in the absence of AutoLink or AutoCreate capacity, element validation uses the value defined in **Value**, considering it in external format. This value can be used consistently if it corresponds to the ASCII value of an absolute identifier.

3.5.1.8.6 Notifications

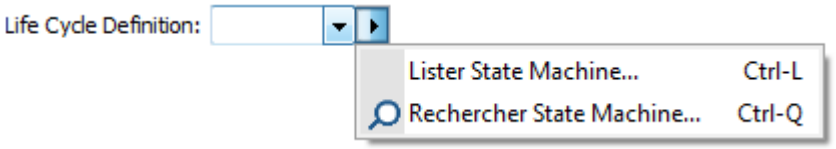
Change: sent when the edit zone has been modified.

3.5.1.9 StaticMenu



Element associated with properties of object type, with specific capacities. See 3.5.1.8

3.5.1.10 ComboBoxMenu

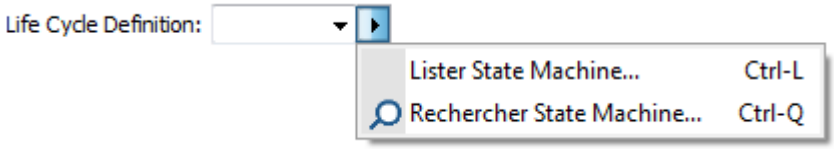


Element associated with properties of object type, with specific capacities. See 3.5.1.8

The first button enables expand of a drop-down list.

The second enables menu opening.

3.5.1.11 DropDownListMenu



Element associated with properties of object type, with specific capacities. See 3.5.1.8

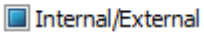
The first button enables expand of a drop-down list.

The second enables menu opening.

3.5.2 Explicit AttributeControls

These types are never proposed as standard, and are therefore always defined in the framework of a template or an updateTool. They enable either proposal of alternative entries for certain property types, or definition of form presentation elements (titles, comments, images).

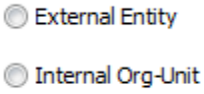
3.5.2.1 3StateCheckBox



CheckBox with third state including 'not specified' state. Can be used if you are concerned by being able to de-specify the attribute, or if you want to explicitly manage a required boolean attribute.

Options, properties and notifications: see 3.5.1.3.

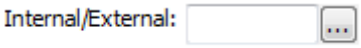
3.5.2.2 RadioButtons



The RadioButtons type can replace the DropDownList type (see 3.5.1.4). The main difference is the fact that the list of enumerated values, which serves to build the list of buttons displayed, is requested just one time at element creation, and cannot subsequently be modified.
This element is presented by default without title.
When element size is not specified:

- Element height is deduced from the number of buttons to be displayed.
- Width is calculated according to the number of text characters associated with each button, as well as the number of characters of the title. Since this calculation is made in DialogUnits and cannot therefore take account of characteristics of the displayed font, the effective width may not be appropriate.

3.5.2.3 EditButton



This type is a simplified version of an EditMenu, which enables execution of only one command.

A parameterization of this element enables use of this button for file or directory selection, using FileSelection and DirectorySelection options.

In others, use of an updateTool or dialogHandler is necessary to implement the command corresponding to button press.

Options

- **Icon=<idPicture>**: enables replacement of <…> of button by a medium-sized image extracted from the indicated MetaPicture.
- **Bitmap=<idPicture>**: enables replacement of <…> of button by a small image extracted from the indicated MetaPicture.

- **FileSelection**: button press enables file selection. In this case you can use the following options to specify required operation:

  Open indicates that we want to open an existing file; if not, we want to indicate a file to be created or modified (**Save** mode by default).

  FileExists=YES|NO (NO by default) indicates if the file should exist. This option is superfluous in **Open** mode.

  DefaultExtension<xxx> enables definition of file default extension.

  Filter<xxx{;yyy ...}> enables definition of filtering on file extensions listed in defining acceptable file extensions.

  PathExist=YES|NO (default YES): when in Save mode and we enter the file name in the edit zone, enables check that directories specified in the path should exist (case YES). In the case of validation, detects an error if the path references non-existent directories.

  In Windows Front-End, button press displays the file selection Windows user interface, in Save mode or in Open mode depending on the selected option.

  In Web Front-End, this check enables piloting of a file upload to the server (Open mode).- in this case the value of the property (available on the server) will contain the name of the downloaded file - or a download in the case of Save - in this case the value of the property will contain the name of the file to be downloaded; if this element is used in a wizard and has not been explicitly downloaded during execution of the wizard, the download starts automatically when the wizard finishes.

  A wizard using this type of element could therefore operate in the same way, whether in Web Front-End or Windows Front-End, in:

  - Open mode, we could consider that the value corresponding to the property is the name of an existing file which we could then open.
  - Save mode, we could use this property as being a file name to be created (or modified) by the system process; at termination of the wizard, the file will be available to the end user (who has specified the directory in Windows Front-End, or it will be downloaded in Web Front-End).

  By code, it is possible to indicate a root directory using the "**root**" field of the **Data** property. This directory will be used in Windows Front-End to initialize the user interface directory.

  DirectorySelection: button press enables directory selection. In this case you can specify using the **PathExist=YES|NO** option if we must check the existence of the specified path or not. This parameterization does not function in Web Front-End.

Properties

- **Value**: value displayed, except if **FileSelection** option is activated and we are in Web Front-End: in this case the value will correspond to the uploaded or downloaded file (this file name not known from the user interface).
- **Data**: in this case **FileSelection** or **DirectorySelection**, this property can contain a string in JSON format containing a "root" value corresponding to the initial directory.

  Example: **{ "root" : "D:\MyDirectory"** }

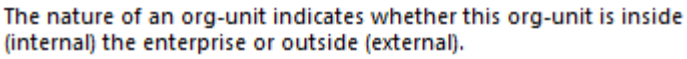
  If this variable is not in JSON format, we consider that it contains the directory name.

  This value is consulted at Windows user interface opening, enabling file or directory selection.

Notifications:

- **Change**: sent when the edit zone has been modified.
- **Click**: sent when the button is clicked.

3.5.2.4 HelpComment



This element is used to display a help or explanation in a form. It is designed to display a text on several lines. It can display rich text. It does not enable data entry and is read-only.

This text can be the comment of a system repository object; in this case the element identifier corresponds to the identifier of this object, and can therefore point an object of any type.

It can also, by option, display a text. In this case the element identifier corresponds to that of the text to be displayed.

In Windows Front-End, it can enable navigation to other system repository comments if the system objects are referenced in the form of a field in the initial text.

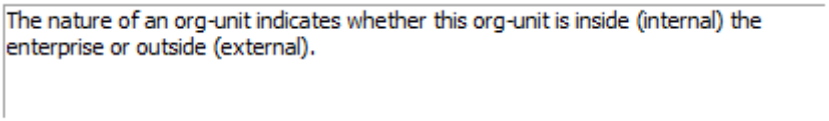
This element is presented by default without title.

Options:

TextMode: indicates that the element must display a text of the object pointed by the element; this text corresponds to the element identifier. Otherwise, the comment of the system object corresponding to the element identifier is displayed.

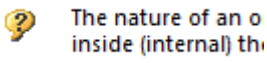
The following options are not taken into account in Web Front-End:

Edge: displays the comment in the form of a simple framed text. In this case the following options are ignored, and navigation by field is deactivated.

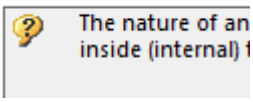


Border: the comment is framed.

Icon or **Info**: the help icon is displayed alongside the comment.



Icon and Border can be associated.

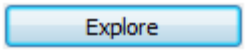


Properties:

Data: text content in ASCII format.

Value: text content in HTML format.

3.5.2.5 Button



Display of a clickable button enabling execution of an action.

If the element identifier points to an _Operator of method type, or a macro implementing a method, the method is executed when the button is pressed.

This element is presented by default without title. The name displayed in the button corresponds to the value of the title, except if the button displays an image.

Options:

NoCall: the method is not called when the button is pressed. Used when the action triggered by the button is processed by the **Click** notification, and not by the method.

Icon=<idPicture>: enables replacement of the name displayed in the button by a medium-sized image extracted from the indicated MetaPicture.

Bitmap=<idPicture>: enables replacement of the name displayed in the button by a small image extracted from the indicated MetaPicture.

When the button displays an image, the size of the button should correspond to that of the image, and is therefore expressed in pixels. The button is therefore SCREENUNITS style. If the size of the button is specified in the template, it will be considered as a size in pixels.

Units=Dialog: in the case where the button displays an image, enables sizing of the button in DialogUnits by deactivating the SCREENUNITS style; this can be used if the button must be aligned with other elements. In this case its size can conform to that of the other elements.

Notifications:

Click: sent when the button is clicked.

3.5.2.6 MegaEditCheck

A MegaEditCheck includes 3 areas:

- a value, which could be editable
- a check box, which could be clickable
- a label.

It is without exception necessary to implement an UpdateTool to be able to use a MegaEditCheck. Without this, a MegaEditCheck behaves like a simple edit area: label value is not specified and check box is not used.

To consult or modify label and check box values, the Data property of the AttributeControl should be used.

This property describes the value and status of these areas in the form of a character string.

It is of form

"label":"<Label>","check":"<checkValue>","show":"<showValue>"

- <Label> corresponds to the string displayed in the label static area.

- <checkValue> corresponds to check box value and can be C (checked), U (unchecked) or I (indeterminate).
- <showValue> enables configuration of check box display and can be E (enabled), D (Disabled) or H (Hidden). In this last case, the check box is not visible.

As indicated in paragraph **Error! Reference source not found.**, it is possible to modify only one of these elements. For example AttCtl.Data = """check"":""C""" forces check mark display without affecting the values of "label" or "show"

If you want to access specific information of the Data string, you can use a MegaToolkit function: JSONStringSearch

```
Set myToolkit = myRoot.CurrentEnvironment.Site.Toolkit
CheckValue = myToolkit.JSONStringSearch("{" & AttCtl.Data & "}", "check")
```

CheckValue contains C, U or I.

Properties:

Value: edit area content

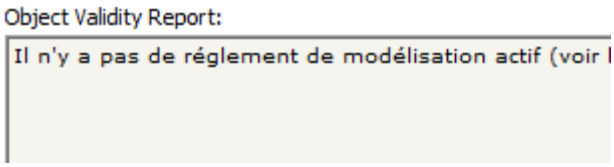
Data: check box and label content and value (see above)

Notifications:

Change: sent when the edit zone has been modified.

Click: sent when the check box is clicked.

3.5.2.7 Viewer



A viewer enables display of an HTML Formatter in a page; it is presented in the form of an HTML browser inserted in the page.

By default, the identifier associated with the element corresponds to the identifier of the formatter; this formatter is launched on the object pointed by the element.

You can also directly launch a macro to generate this HTML content by means of **DirectMacro option**. In that case the identifier associated with the element corresponds to the identifier of the macro to be executed. This macro must implement the **Generate** or **GenerateStream** method which will be call to generate the HTML content of the viewer.

```
Sub Generate(obj As MegaObject, ctx As MegaObject,data As String,strout As String)
Sub GenerateStream(obj As MegaObject,ctx As GenerationContext,data As String,stream)
```

Data corresponds to the Data value of the AttributeControl.

This system enables to specify viewers that run on informal objects, associative objects, or whose content can depend on the association browsed from the object pointed by the element, as appropriate.

Formatter execution is asynchronous.

The element is displayed by default with its title above.

It has style BOTTOMALL and therefore is deformed to use all the page when it is the last element.

Being an HTML component, this element offers great freedom in what can be displayed. However, it is advisable to consult HTML formatter documentation if you want to trigger interactions that can operate in Web Front-End and in Windows Front-End.

However, if you want to use a viewer to display systems designed to execute updates, it is advisable to avoid direct update execution, which is not easy in Web Front-End. By configuration it is possible to synchronize updates from the viewer with form validation. The principle used is to designate a Macro to handle updates from the viewer at form validation. To do this, an interaction mode between the viewer and its container is defined.

Interactions are by means of a callback function, which should be presented in the header of the HTML document. This function has (javascript) as prototype

```
function onContainerCmd(prm,attctl)
```

prm is a character string indicating the interaction type

attctl is a parameter referencing the viewer container in Web Front-End. In Windows Front-End, this parameter is empty. In particular it enables identification of whether the use context of the HTML document is Web Front-End or Windows Front-End.

Calls to this function are carried out if the following options are present:

Initialize: the callback function is called at initialization of the form, with value prm = "initialize".

Refresh: indicates that the viewer should not be recalculated at refresh; instead of recalculation, the onContainerCmd function is called, with value prm = "refresh".

Appliable: indicates that callback should be called at form validation; the onContainerCmd function is called, with value prm = "apply". By default, this function is only called if the element is ‘‘dirty’, that is considered as modified. It is however systematically called at each form validation if the **alwaysCall** option is specified. In the current version, this option is essential for Web Front-End operation.

The onContainerCmd("initialize" function is systematically called when any of the options enabling callback call is specified.

When calling onContainerCmd("apply", it is possible to specify the **Data** property of the element to control updates to be executed.
To be usable, this property must contain a JSON format character string including the macro to be called on the server:

```
"applyMacro":"<MacroID>"
```

At validation on the server, if this field is found in the **Data** property of the element, the <MacroID> macro is instanced and we call on this macro the function

```
Function InvokeOnObject(object,idText,Page)
```

Where **object** is the object of the AttributeControl, **context** the component corresponding to the AttributeControl, and **param** the JSON string corresponding to the **Data** value containing data allowing the macro to know updates to be executed.

To specify the Data property from the onContainerCmd("apply",attctl) function, proceed differently according to whether you are in Windows Front-End (attctl = "") or Web Front-End context.

- In Windows Front-End, you can directly update the AttributeControl using the context object:

```
external.ContextObject("PageContext").SetData(result.data)
```

- In Web Front-End, the value returned by the onContainerCmd function should be a JSON format string. Present in this string can be the fields:
  - data: is assigned to the **Data** property.
  - reload: causes reloading of the formatter if assigned to True.
  - errorMessage: is assigned to the **ErrorMessage** property and cause – if not null – an error on the page.

The **context** component, accessible in Windows Front-End from the **external** object and in the InvokeOnObject function of the validation macro, has the following properties:

.setData(string): updates the **data** property of the AttributeControl.

.setDirty(bool): generates an "Update" notification if true and positions the dirty value.

.setError(string): updates the **ErrorMessage** property of the AttributeControl. If this string is not empty, indicates that the element is in error and the form cannot be validated.

.refresh(): refreshes the viewer.

.notify(action): generates a notification on the page.

The following example proposes an implementation enabling generic update of object properties via a viewer.
To do this, consider that the validation macro uses an ApplyParameter field of JSON Data. This property should be an object table, each object containing the following fields:

id: <identifier of property to be updated>

value: <property value>

format: (optional) update format

The validation macro is implemented in javascript to use a JSON format parameter as simply as possible It browses the table defined above and executes the corresponding update.

```
//Language:javascript
function InvokeOnObject(obj,device,param){
  obj.getRoot().print(param);
  var vparam = eval("(" +param + ")");
  obj.getRoot().print(vparam.applyMacro);
  for(var i = 0; i < vparam.applyParameter.length; i++) {
    var itemObj = vparam.applyParameter[i];
     obj.getRoot().print("ID=" + itemObj.id + ", VALUE=" + itemObj.value + ",
FORMAT = "+ itemObj.format);
    obj.SetProp(itemObj.id, itemObj.value , itemObj.format);
  }
}
```

This macro having been defined, you must now create an HTML document able to manage this interaction. This document is displayed and will enable update of ShortName and Cardinal MetaAttributes of a MEGA object (for example a Site, an Operation, an Org-Unit or any other MetaClass using the Cardinal MetaAttribute).

The HTML document of this example is of form:

```
<hmtl>
  <head>
    <script language='javascript'>
      function onContainerCmd(prm,attctl) {
        var fieldVal1 = document.forms[0]['input1'].value;
        var fieldVal2 = document.forms[0]['input2'].value;
      var result = {'reload': 'true','errorMessage':'null','data':
        {'applyMacro':'#MACROID#','applyParameter':
            [{'id':'~MtUi79B5iyF0[Cardinal]','format':'','value':fieldVal1}
            ,{'id':'~Z20000000D60[Short Name]','format':'','value':fieldVal2}
            ]}};
        if(attctl==='')
          external.ContextObject(""PageContext"").SetData(result.data);
        alert('test:'+prm+':'+attctl);
        return result;
      }
</script>
  </head>
  <body>
    <form>
       <table border='0' align='CENTER'>
                    <tr><td><b>Cardinal      :</b></td><td><input      id='input1'
type='text'
                value='"#CARDINAL#"' /></td></tr>
          <tr><td><b>Short Name :</b></td><td><input id='input2' type='text'
                value='"#SHORTNAME#"'/></td></tr>
       </table>
     </form>
  </body>
</html>"
```

In this HTML, #MACROID# should be replaced by a field representing the absolute identifier of the validation macro, #CARDINAL# by the value of the Cardinal MetaAttribute for the form object - `oMegaObject.GetProp("~MtUi79B5iyF0[Cardinal]")` and #SHORTNAME# by the value of the short name - `oMegaObject.GetProp("~Z20000000D60[Short Name]")`

This HTML document generates a warning at validation.

For operation

Options:

HideStatusBar: in Web only. Hides the status bar (and therefore the button enabling element refresh).

Initialize, Refresh, Appliable: indicates presence of an onContainerCmd(prm,attctl) callback function in the HTML document header, enabling interactions between the HTML document and the form.

alwaysCall: when the **Appliable** option is defined, indicates that the validation macro must be called, even if the component is not considered as being modified. This option is currently necessary for Web Front-End validation operation, since it is not possible to dynamically modify component status.

Properties:

Value: HTML content of viewer (currently non-operational)

Data: contains a string in JSON format containing data on interaction between the HTML document and the validation function (see above).

Specific component

The specific component accessible from the AttributeControl object includes the interaction functions described above.

SetData(string): attctl.component.SetData string corresponds to attcl.data = string

SetDirty(bool)

SetError(string): attctl.component.SetData str corresponds to attcl.errorMessage = str

Refresh()

Notify(action)

3.5.2.8 ComboLinks

This element enables creation of the "owner" of an object. It therefore handles the problem of a data entry containing exclusive MetaAssociationEnds, from card-max to 1. The MetaAssociationEnds list included in the drop-down list is discriminated by an _operator of "compound" type, the MetaAssociationEnds presented being tagged "deep" for this operator.

This element is not designed to be overloaded.

Properties:

Value: edit area content (corresponds to short name of associated object)

TargetID: Identifier of associated object

SourceID: identifier of selected MetaAssociationEnd.

Notification:

Change: sent when associated object is changed.

3.5.2.9 Shape

The Image AttributeControl enables display of an image, possibly followed by a static area.

It enables representation of an object; to do this you can define a specific object for this element.

Click on an image can trigger an action, which can be to display the object menu.

The image displayed is a MetaPicture format 32x32, corresponding

- to the image of the object associated with the element with the **ImageFrom:Object** option
- to the image associated with the property of the element with the **ImageFrom:Attribute** option. In this case, if the property is MegaIdentifier type, it is considered as containing the identifier of a *MetaPicture* which is then displayed. If not, it should be an enumerated property of which we display the *MetaPicture* associated with the enumerated value corresponding to the value of the property.

The static area is not displayed if the **Value:void** option is specified. If not, it corresponds

- to the title of the area with the **ValueFrom:Title** option
- to the value of the property of the element with the **ValueFrom:Attribute** option
- to the value of a property of which we specify the field with the **ValueFrom:<PropID>** option

When the area displays its title, or when the **Value:IncludeTitle** option is specified, the value of the static area is concatenated with the title.

When we click the image, no action or notification is triggered with the **Clik=Inactive** option. The **Click=PopupMenu** option displays the object menu.

If you want to display by this element an object seen from the main object of the page, creation of a map is not necessary. You need only specify the collection browsed with the option **FromCollection:<CollectionID>{TypeName|ClassName}** and associate with the element the absolute identifier of the object to be displayed. Most often this requires call to a page generator (see 3.3.1.6). In this context it is possible to include the name of the object type (TypeName) or the name of its MetaClass (ClassName).

This element is present by default without title; it being possible to include the title in the value. Its default style is SCREENUNITS, meaning that its size is expressed in pixels in the configuration (unless otherwise specified).

Options:

FromCollection:<CollectionID>{TypeName|ClassName}: displays image of an object seen from the collection

Click={Inactive|PopupMenu}: disables the click notification, or displays the object pop-up menu

Value :{void|IncludeTitle}: specifies display of the static area: hidden or preceded by the element title.

ValueFrom:{Title|Attribute|<PropID>}: specifies the displayed value (area title, element attribute, or particular property of the object indicated by its <PropID>) field.

ImageForm:{Object|Attribute}: specifies if the image is the image of the object or corresponding to the image associated with the property of the element.

Units=Dialog: enables dimensioning of the image in DialogUnits by disabling the SCREENUNITS style; this can be used if the image should be aligned with other elements – in this case its size can conform to that of the other elements.

Properties:

Value: content of the static area

Data: enables definition or determination of the image to be displayed, in form

"image":"<moniker>"

The moniker generally corresponds to the identifier of the MetaPicture, preceded by character '~'

Notification:

Click: sent when the image is clicked

3.5.2.10 Label

Displays a label. As standard, this label corresponds to the name of the object whose identifier is associated with the element. It is however possible, by option, to display the element title or a property value.

This element is presented by default without title.

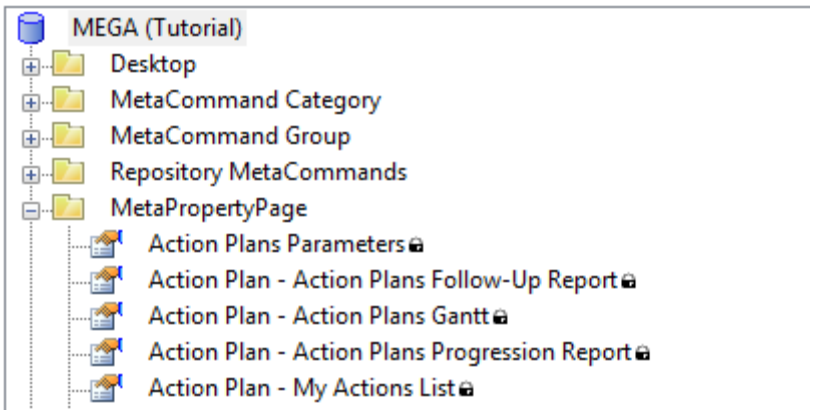
Options:

ValueFrom:{Title|Attribute|<PropID>} indicates that the displayed value corresponds to the element title, to the property value associated with the element, or to another property of which we pass the identifier in the form of a field.

Properties:

Value: content of the static area

3.5.2.11 MetaTree



This AttributeControl enables display of a MetaTree in a properties page.

The identifier associated with the element corresponds to the identifier of the MetaTree. The root of this MetaTree is the object pointed by the element.

A MetaTree can be used to control a Map.

MetaTree options are described in a JSON format character string (character case should be scrupulously respected). Note that an option string containing a bracket must be specified using a Parameter.

Example:

```
treeParam = Parameter({"metalist":"OEb0XL0zF93Q","checkbox":true})
testTree = Item(~ulSYFlOK1XB0[Objects]),Control(MetaTree),Param(@treeParam)
```

The JSON can contain the following parameters:

"checkbox":boolean enables to have check boxes opposite each element. The default value is false.

"selPropagate":{"mode":string,"direction":string,"autoExpand":string} This option is only effective in the framework of a checkbox tree, and enables definition of the propagation mode of a selection of elements in the tree.

mode can be **on**, **off**, **defaultOn**, **defaultOff**. In the two default modes, a button allows the user to enable or disable this behavior as required.

direction can be:

down (default value): when a node is selected or deselected, all children are immediately selected or deselected. Behavior is recursive and should therefore only be activated on trees that are truly hierarchical (not of cycle).

up when a node is selected, all parents are selected, up to the tree root. When a node is deselected, its parents are deselected, except those that have another descendant selected.

both combines both behaviors.

autoExpand can be **on**, **off** (default), **defaultOn, defaultOff** but is only valid for descending propagation. This enables automatic expand of nodes to show selected child elements, with or without a button available to the user. This option should be used with care, since expanding the tree can be costly in terms of time.

Propagation of the selection is not supported in Windows Front-End.

"metalist":string: enables definition of additional columns on the MetaTree. The parameter supplied is a MetaList identifier, of which MetaFields will be interpreted as tree columns. MetaField now inherits from Abstract Property. This enables connection by the Implementation MetaAssociationEnd of a macro implementing the calculated attributes interface.
When value of a MetaField with macro is requested in the framework of a tree, the MegaObject is enhanced by addition of a "virtual" attribute "~MJoWUFOzF92Q[Tree Node Full Path]", which enables access to the full path of the object in the tree, formatted in the same way as the selection.

In Web Front-End, in-place edit is possible in columns by double-clicking the value you want to modify.
Limitations:

Presentation in columns is available in Web Front-End only.

In Windows Front-End, these fields are concatenated with the name of the element.

MetaFields of columns operate only for MetaFields based on a macro or on a MetaAttribute (not MetaAssociationEnd or query)

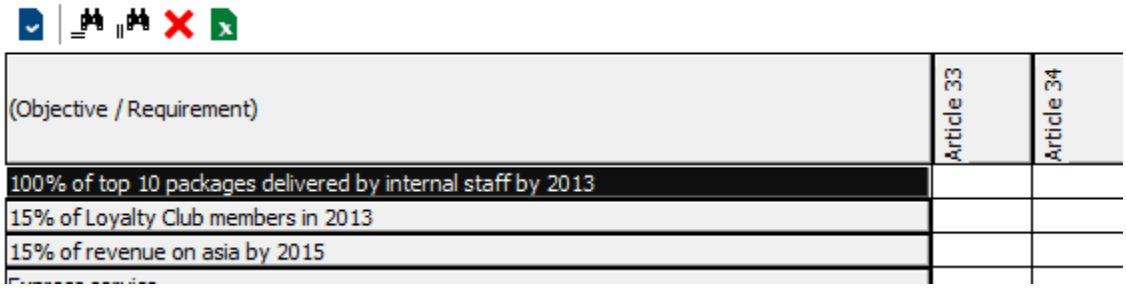
Properties:

Value: lists selected objects. This value is a JSON format character string that gives the full path of objects in the tree. In assigning the value we modify the selection.

Notifications:

sent when the selection changes.

3.5.2.12Matrix



This AttributeControl enables display of a Matrix in the properties page. This matrix can be either display of the content of a Matrix or System Matrix occurrence, or display of the matrix resulting from application of a matrix template to the object pointed by the element (default mode).

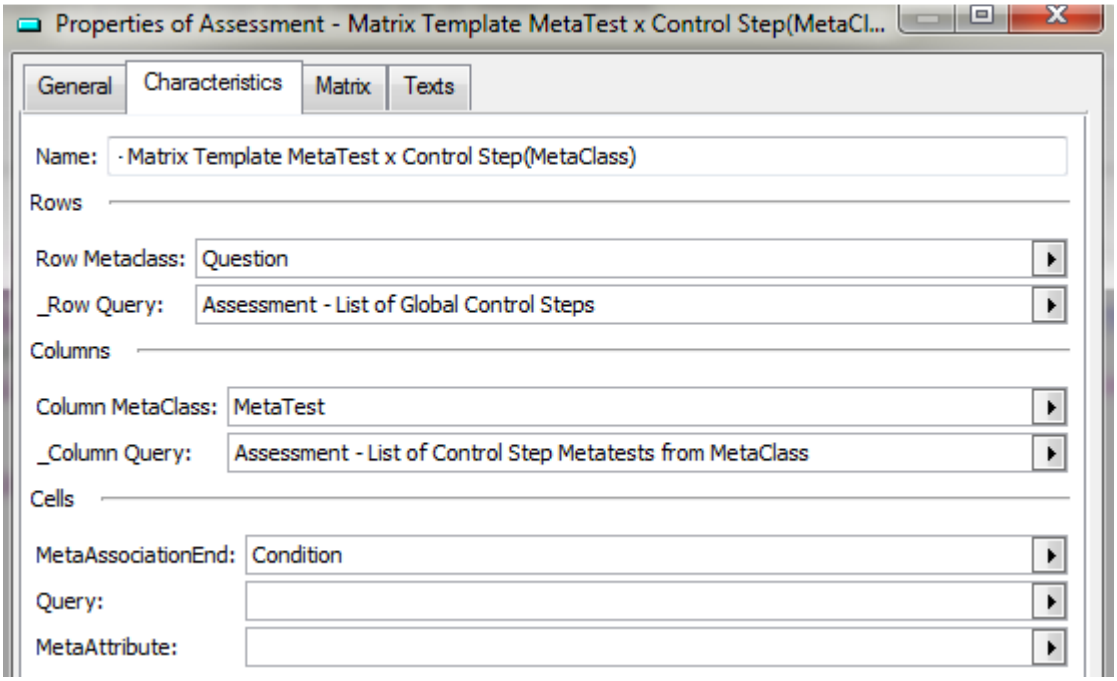
To display content of a Matrix (or System Matrix), the **Object** option should be present – in this case the object pointed by the element is the matrix, and its content will be displayed – or the **Data** option – in this case the identifier associated with the element corresponds to the occurrence of the matrix to be displayed.

- In the case of the Object option, the identifier associated with the element is not taken into account; in the case of the Data option, content of the matrix does not depend on the object pointed by the element.
- The Matrix (or System Matrix) occurrence contains the list of selected rows and columns as well as those that may have been added using addition buttons. These collections are serialized in a technical attribute of the matrix at validation of the element. The matrix occurrence also memorizes display modifications (for example column width).

In the case of matrix template display, the identifier associated with the element corresponds to the identifier of the Matrix Template to be used. A matrix template enables definition of:

- an object collection defining rows of the matrix. This collection is obtained from the object pointed by element.
- an object collection defining columns of the matrix. This collection is obtained from the object pointed by element.
- a MetaAssociationEnd defining cell content: cell content corresponds to the association between column object and row object according to this MetaAssocationEnd.
- a MetaAttribute of the MetaAssociation defining the value displayed in the cell: if this attribute is not defined, the cell will display a checkbox indicating link presence or absence.

- It is possible to specify a query instead of a MetaAssociationEnd: in this case we display in the cell a checkbox indicating presence of the row object in the collection obtained from this query from the column object.



Unlike the case of matrix display, the list of rows and columns corresponds exactly to the content of row and column collections. You must be able to modify content of these collections to be able to modify the list of displayed rows and columns.

Options:

NoBar: the matrix toolbar is not displayed.

Object, Data: indicates that the matrix displays content of a Matrix or System Matrix occurrence.

AutoLink: simple click on a cell enables creation or deletion of the corresponding association.

Component methods:

Specific methods of the AttributeControl can be called from the component object (AttributeControl.Component)

.Fetch(Input As String) As String: this function obtains complete content of the matrix in the form of a JSON format string. The input parameter is ignored.

The JSON restored by this function contains two elements:

- an **hdr** element containing description of the matrix, and
- a **content** element containing the rows.

In **hdr**, columns are presented as a **columns** table of form

```
{ "id": "<idcol>", "format": "<format>" , "image":bool , "name":name };
```

<idCol> contains absolute identifier of the column element.

Also included are elements **columnTotal** (number of columns), **total** (number of rows), **cornerName** (title, displayed in cell top left corner) and **checkBoxCell :true** if cells include checkboxes, indicating presence of the association.

content is a table, each element including a row, of form

```
{ "id" : "<idLine>", "image": "<image>", "<columnid>": "<value>" … }
```

idLine corresponds to the identifier of the row object; only columns effectively specified are present,

<columnid> being the column object identifier and value to be displayed.

3.5.2.13DropDownSelection

This element enables display of content of a selection in the form of a drop-down list. The menu button enables addition of elements to the collection, and access to the menu of the object selected in the list.

The identifier associated with the element is the identifier of a collection (MetAassociationEnd, query or other abstract collection).

Although of appearance identical to a DropDownListMenu (**Error! Reference source not found.**) the two elements are fundamentally different.

- The DropDownListMenu manages an Object type property. The drop-down list enables display of update proposals. Edit area content corresponds to the value (current or after validation) of this property. This element resembles an edit area containing update help features.
- The DropDownSelection manages an object collection directly associated with the object of the element. Edit area content is an object selected in this collection. This element is similar to a single-selection list view presented in compact form.

Like a ListView, a DropDownSelection enables updates on the collection (occurrences addition and deletion). It also enables (it is even its main use) control of a Map pointing to the object selected in the collection.

This element is not designed to be overloaded; in particular it does not generate public notifications. You can however use private notification of number 1, corresponding to change of selection. You cannot directly change the selected element in the list.

Options:

- **NoMenu**: in this case, the menu button is not displayed and the element has the appearance of a simple drop-down list.
- **SelectOnRefresh**: indicates that the collection should be recalculated when the element is refreshed; to be used when we want to take into account collection updates when corresponding to a query – in particular an ERQL query.

Properties:

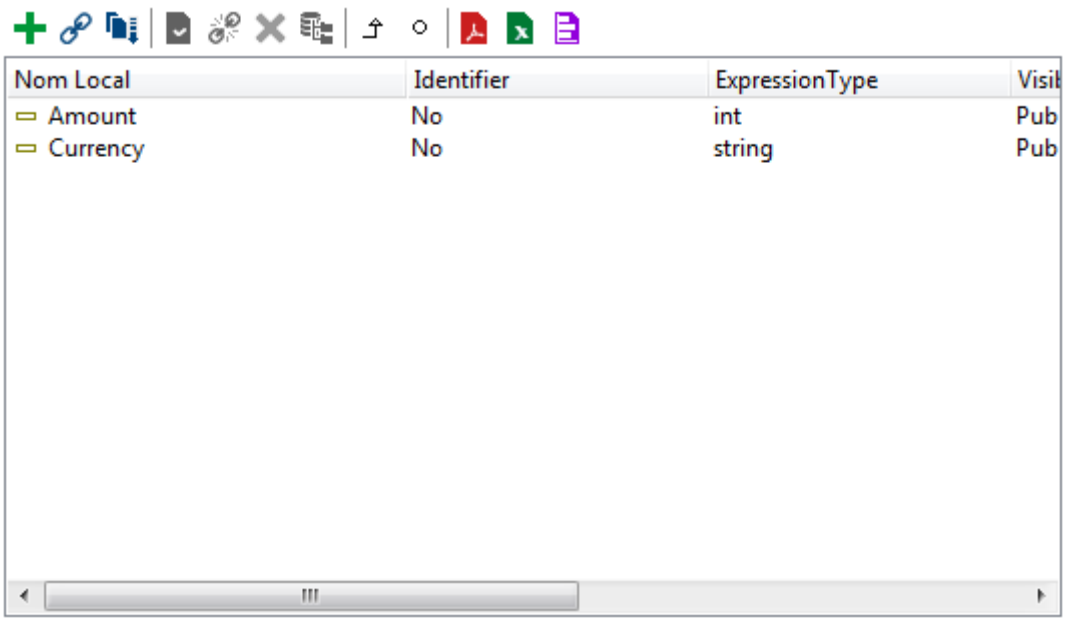
- **Value**: edit area content (corresponds to short name of selected object) This value cannot be modified.
- **TargetID**: Identifier of selected object This value cannot be modified, (except by command SELECT-<id> from the properties page).

3.5.3 Composite AttributeControls

Composite AttributeControls enable presentation of complex data. To define this data, you may decide to associate it with sub-AttributeControls using a composition Map.

Example: sub-AttributeControls, amongst other things, enable definition of columns of a ListView.

3.5.3.1 ListView



A ListView enables display of an object collection. This collection is obtained from the object associated with the ListView element. The columns – properties of displayed objects – can be configured. A ToolBar is defined by default for the ListView, enabling triggering of actions relating to the collection (creating a new element for example) or to an element selected in the list. The ListViews can be single-selection or multiple-selection. Finally, it is possible to define a ListView displaying several selections, known as alternative selections.

A ListView can control a map: in this case the object selected in the list is assigned to the controlled map, and therefore becomes the object associated with items contained in this map.

On this subject, the ListView content map (enabling ListView specification) should not be confused with the map controlled by the ListView (enabling object path definition).

In the basic case, the identifier associated with the ListView is that of the collection to be displayed: the list displayed corresponds to the path by api of the collection `obj.GetCollection(<ItemID>)`

3.5.3.1.1 ListView "content" map

The listview content map enables definition of buttons, columns and alternative collections. Options defined for these elements indicate for which use it is reserved:

- An element corresponding to a toolbar button contains the "**PushButton**" option.
- An element corresponding to an alternative collection contains the "**AlternateSelection**" option.
- An element corresponding to an additional column contains the "**Extra**" option.

- The other elements are considered as column overloads.

3.5.3.1.2 Configuration elements

3.5.3.1.2.1 Specification of displayed columns

Displayed columns of a ListView are obtained from description of the displayed collection; it is possible to define filtering so as not to display all properties defined for this description. For information, it is possible to obtain by api the elements of this description by `obj.GetCollection(« ItemID »).GetTypeObject.Properties`

With each column is associated the identifier of the property represented. This identifier is unique, so two columns displaying the same property cannot be inserted.

If no column filtering option is present, all columns of the description are displayed, except:

- columns corresponding to Properties not visible according to metamodel access of the current user
- columns corresponding to MetaAttributes hidden at MetaClass level for the profile of the current user
- columns corresponding to translations of MetaAttributes (only the root MetaAttribute is displayed, corresponding to display of the value in the user language)
- columns corresponding to an administration property, a local property, or a property hidden in edit – corresponding to flags "in administration tab" (0x4), "never appears in list column" (0x1) and "hide on edit" (0x20000000) of the "extended properties" attribute.
- The "ShortName" attribute - we directly display the attribute that substitutes the name, beside which ShortName would be redundant.

If this list is not suitable, you can define generic filtering using the following options:

NoDefaultColumn: in this case, only the naming attribute is displayed; this is the one serving as name for description of the collection. It can then serve:

- as name if the name attribute is not substituted, or if the collection displays the long name by default. This is determined by the "**Name Appearance**" MetaAttribute which is read either on the browsed *MetaAssociationEnd* or on the *MetaClass*.
- as the attribute that substitutes the name if the collection is configured to display the short name. The name of this attribute generally being esoteric, we give it the conventional name ("Local Name").
- of the ShortName conventional attribute for collections derived from an abstract class or calculated
- of another attribute in the case of informal collections for which a naming attribute has been explicitly defined.

StandardColumn: filtering is identical to default filtering, but only the default columns (properties having enabled the "Default Column" (0x2) flag in their "extended properties") are visible by default.

AssocColumn: besides the naming attribute, the listview displays the properties defined on the browsed MetaAssociation.

More precise filtering can be defined by overloading columns in the ListView content map. When the identifier of a content map element of overload type (that is, not having as option "**Extra**", or "**AlternateSelection**", or "**PushButton**", or "**Cookie**") corresponds to a property identifier

of the collection description, a new configuration is applied on this column according to the following rule:

- If the element is hidden (keyword **HiddenOn**()), it is filtered. Note that if the hide condition is likely to change at object update, the columns list should be recalculated at ListView refresh; this is not done automatically, and to do this you must supply the ListView (and not the element) with the **RefreshColumns** option.
- otherwise, and if the column was filtered by default, it is made visible; this is not however done if filtering followed metamodel filtering or was defined by user rights.
- If the element is disabled (keyword **DisabledOn**) the column is not forced in read-only mode
- If the element is defined with keyword **Title(Up),** the column title is replaced by the **Name()** of the element.
- If the element is defined with a particular size (**Size(x)**) this size is used as initial size of the column. If the size is empty (**Size(0)**) the column is initially hidden.
- If the element is defined with a **Control(CheckBox)** it is displayed in the form of a check box; with a **Control(ComboBitmaps)** it is displayed preceded by its image.

Columns not included in the description can be added, known as "ExtraColumns". These columns are defined by content map elements provided with the **Extra** option. See 3.5.3.1.2.4. Finally, you can define columns specifically calculated for the ListView which are also content map elements, but provided with the **Cookie** option.

Default columns appear in the order defined for the description; it is however possible to reorder all or certain of the listview columns using the **ReOrder** option. When this option is specified, it is content map element declaration order that is used to sort columns, elements not obtained from this map being listed afterwards. Note that the name is always placed first.

3.5.3.1.2.2 Images associated with listed objects

By default, the first element of each line in the list is the image of the listed object. If you do not want this image to appear, specify option **NoLineBitmap**.
The image to be displayed can be redefined, either by stamping or by replacement. To do this, you must have available on the list element one or several properties for which an image is defined; these can be:

- enumerated properties of which values are associated with a MetaPicture
- calculated properties of megaidentifier type, of which values correspond to a *MetaPicture*.

To replace the image of the line:

➔ specify the **BitmapSelector=*Field*** option where <*Field*> is a reference to the property to be used.

To stamp the image of the line:

➔ specify the **BitmapStamper=*Field {,Field}*** line. Several stamps can be used. Stamped images do not replace the list image, but are placed above transparently.

3.5.3.1.2.3 Element sorting

At initial display, elements in a ListView are sorted:

- according to their order number
- at equal order number, according to name.

The order number and name are properties defined as those of the collection description. Name calculation has already been mentioned in section 3.5.3.1.2.1 and mainly depends on the target MetaClass and the collection browsed. The order number is the 'Order' MetaAttribute when the collection displayed is a MetaAssociationEnd. When it is an ERQL Query – which does not allow order number retrieval – the object creation date is used.

To sort initially in lexicographical order, that is not use the order number and sort directly on the name:

➔ specify the **NameSort** option.

To redefine the attribute used to determine order number:

➔ specify the **OrderOverloadProp=*PropID*** option.

3.5.3.1.2.4 Definition of Columns outside description

You can insert in a ListView columns not included in description of the browsed collection. These columns are represented by listview content map elements, with options **Extra** or **Cookie**.
An **Extra** column has a corresponding property identifier which should be accessible for all listed elements, independent of the ListView. Similarly, if you define the following element in the content map:

```
Extracol = Item(PropID),From(contentMap),Param(Extra)
```

Then for all elements listed in the collection, it should be possible to execute (in script)

```
oListItem.GetProp(«<PropID>»)
```

It is for this reason that these columns are not in principle MetaAttributes, since these are normally included in descriptions derived from the metamodel. However, it is possible to include _AbstractProperties (generally TaggedValues).

- If listed elements are 'Elements with TaggedValue' or 'System Elements with TaggedValue', all non-calculated _AbstractProperties can be applied to them.
- If an _AbstractProperty is calculated, the possibility of inclusion as a column depends on its implementation, and more specifically on its compatibility with the browsed elements.

You can include a column whose calculation is specifically executed in the ListView: in this case, the content map element has the **Cookie** option, and other elements enabling definition of the calculation mode of this column as defined below.

```
Param(Cookie{,BasedOn=BaseId}{,IsTrigger=n},DataSource=DSId{,DSOptions})
DSOptions that can be: From:CollectionId
                       Path:ObjectPath
                       Macro:MacroId InitString
```

The principle of this calculation is to search for the column value not on the line object, but in another collection, called here DataSource.
In the standard case, this collection is constituted from the source object of the ListView.

To obtain the value of the column, we first search this DataSource for an object whose identifier corresponds to that of the line object.

If the column is defined as trigger (keyword **IsTrigger=1** or **IsTrigger=2**), the collection derived from DataSource is managed as a link controlled by the column. If **IsTrigger=2**, the column does not reference a property, but is considered as a boolean value, displayed by a check box, whose value indicates presence of the object in the DataSource.
In other cases:

- if the object is not present in the DataSource, the column is unspecified.
- otherwise, value of the column corresponds to the value of a property of the DataSource object, corresponding either to the absolute identifier defined in <BaseId>, or by default to the absolute identifier of the column.

The following is the equivalent of the column value calculation in script.

```
' oRootObject  : object associated with ListView  (ListView.GetObject)
' oLineObject : object corresponding with line
' sColumn : column value

Dim cDataSource as MegaCollection
Set cDataSource = oRootObject.GetCollection("<DSId>")
    'this example relates to a simple DataSource, that is without <DSOptions>

Dim oDataSourceObject
Set oDataSourceObject = cDataSource.Item(oLineObject.GetID)
If oDataSourceObject.Exists Then
  if "<BaseId>" = "" Then
    sColumn = oDataSourceObject.GetProp("<ColumnId>")
  Else
    sColumn = oDataSourceObject.GetProp("<BaseId>")
  End If
Else
 sColumn = ""  ' No reference to object in DataSource, column not specified
End If
```

Definition Options of a DataSource enable management of a DataSource not corresponding to a collection directly accessible from the source object of the ListView

From :*CollectionId* enables access to the DataSource collection indirectly from a link considered unique. It could be that the DataSource collection is not defined.

```
Dim cDataSource as MegaCollection
Set oInter = oRootObject.GetCollection("<CollectionID>").Item(1)
If oInter.Exists then
  Set cDataSource = oInter.GetCollection("<DSId>")
Else
  Set cDataSource = Nothing ' in this case DataSource is not defined
End If
```

Path:*ObjectPath* In this case the DataSource collection is not obtained from the source object of the ListView, but from the object of which we specify the *MegaPath*. It could therefore be that the DataSource collection is not defined if the MegaPath does not correspond to an object.

```
Set oInter = oRoot.GetObjectFromPath ("<ObjectPath>")
If oInter.Exists then
  Set cDataSource = oInter.GetCollection("<DSId>")
Else
  Set cDataSource = Nothing ' in this case DataSource is not defined
End If
```

Macro:*MacroId InitString* enables access to the DataSource collection from a Macro implementing a standard collection as follows:

```
Function GetStandardCollection(      MegaObject as MegaObject,
RetType as Variant,
InitString as String) as MegaCollection
```

MegaObject is the source object of the ListView

InitString is the character string specified in the option

RetType is an optional parameter enabling overloading of the returned collection type, when it is specified with an identifier corresponding to a collection type.

When a column derived from a DataSource can be modified, update affects the DataSource collection. If the column is not defined as being a trigger:

- When an object corresponding to the line element exists in the DataSource, the value of the property of this object corresponding to the column is updated with this value.
- When this object does not exist, we try to insert it in the DataSource collection by providing the identifier of the line object (except of course if the update value is empty: in this case we do nothing). If creation is successful, we update the property corresponding to the column of this new object with the update value.

If the update column corresponds to a trigger:

- In the case of an indicator (**IsTrigger=2**), we insert or remove the collection object according to the boolean value of the update (nothing is done if the value supplied corresponds to the calculated value).
- In the case of a simple trigger, update with an empty value removes the object from the collection if it was included; in other cases, update corresponds to the standard case (insertion if required and property update).

In the case where the DataSource collection is not defined, update fails.

3.5.3.1.2.5 ToolBar content specification

A ListView toolbar includes:

- standard buttons, which can be filtered
- specific buttons, which should be defined in the content Map.

The ListView toolbar can be hidden using the **NoBar** option.

Standard buttons of a ListView are:

Create (**C**): Starts the tool for creation of a new element in the list.

Link (**L**): Starts the tool for creation of a new element in the list (connect).

Reorder (**R**): Starts the tool enabling reordering of elements of the list according to standard order (see 3.5.3.1.2.3).

The following buttons concern the occurrence selected in the list:

Destroy (**D**): Starts the tool to delete the selected object.

Unlink (**U**): Starts the tool to remove an element from the list (disconnect).

Properties (**P**): Display of the properties dialog box of the object selected in the list.

Explore (**E**): Starts the explorer on the selected object (available only on Windows Front-End).

Open (**O**): Executes the default command of the menu of the selected object. This command is not present by default in the toolbar.

Standard buttons are automatically disabled when the action concerned is not possible or prohibited.
To filter standard buttons use the **ToolBar[]** option. Between the toolbar brackets are the names (or abbreviations) of buttons concerned, preceded by symbol "+" if you want to show these, and "-" if you want to hide them.

Example: the **ToolBar[-RDU]** option specifies hiding of buttons Reorder, Destroy and Unlink.

When a standard button is hidden, the corresponding command in the pop-up menu of the selected element is automatically removed (if it exists). If the toolbar is integrally hidden by the **NoBar** option, but you do not want commands corresponding to the buttons to be removed from the pop-up menu of the selected object, you must include the **ToolBar[]** option which will in this case enable forcing of display of corresponding menu commands.

When alternative lists have been defined in the ListView, selection of the list to be displayed can be by means of dedicated buttons (see 3.5.3.1.2.6)

Export buttons enable production of a document using content of the ListView. Export in Excel (X) and PDF (P) are available as standard. The **ExportCommands=P|X|0** option enables control of display of these buttons.
You can redefine the list of these buttons in a given ViewPort PropertyPage (see 2.3.4). In this context, the default list of buttons will follow configuration of this viewPort, defined in its *_Parameterization.* text.
In this configuration, we define export buttons in the **[ListViewExport]** section:

```
[ListViewExport]
   <ExportName>=Item(<ButtonID>),Param(<extraParameter>),Name(<Name>),Picture=<Pi
ctureID>,Method=<MethodID>
```

Each export button is identified by its **ExportName**. The first letter of each of these ExportNames should be distinct, since it identifies the button in the toolBar and can be used in ExportCommands to specifically filter export buttons in the ListView.

Example: if you define an export named "Html", **ExportCommands=H** will indicate that only this export button will be present in the listView.

Identifier of the button will be <ButtonID>; the name displayed will be <Name>, which can be a character string or a field – in this case the name will be that of the referenced object -. The image associated with the button corresponds to <PictureID>, and the macro to be invoked <MethodID>. <extraParameter> can be used to specify an initialization string for this macro.

You can define additional buttons or modify behavior of standard buttons, using the content map. To do this, elements with the **PushButton** option should be added to the content map. Clicking one of these additional buttons generates a notification TBN_DROPDOWN (64826) relating to the ListView, which can be intercepted by the trigger macro of the property page (see 3.4.3). The **PushedButton** ListView specific function enables determination of which button has been clicked: It returns the identifier of the content map element associated with the button.

The name associated with the button corresponds to the title defined for the element. Configuration of a button element uses the following options:

```
Param(PushButton{,OnSelection}{,CheckFlags}
    {,Picture=PicId}{,Method=MacroId}
    {,ReplaceStandard[<ButtonName>]}{,OverloadStandard[<ButtonName>]})
```

OnSelection: indicates that the button only ungrays if an occurrence is selected in the list.

CheckFlags: indicates that the button should gray if the element is disabled (for example by DisabledOn).

Picture=*PicId*: defines the image associated with the button.

Method=*MacroId*: indicates that the button should not generate a notification on the trigger macro of the property page, but on an instance of the *MacroId* macro created specifically for this button.

ReplaceStandard[] and **OverloadStandard[]** enable overload of a standard button. With **OverloadStandard**, only the image of the standard button is affected. With **ReplaceStandard**, standard processing of the button is replaced by the specific processing associated with the element. The name of the standard button to be overloaded corresponds to the name defined in **ToolBar[]**

3.5.3.1.2.6 Alternative list specification

Specification of alternative lists enables a unique ListView to display several distinct collections. Advantages here are more compact page size and faster loading, since only the selected collection will be queried.
When a ListView has alternative collections, we add to it a user interface enabling selection of the current collection:

- This selection can be made using additional buttons added to the ToolBar of the ListView. These buttons are RadioButtons – only a single selection can be made. This is default behavior.
- It can be achieved by means of a DropDownList inserted in this same ToolBar; to do this, we should include the **ShowAlternate=DropDown** option.
- It can be done – in Web Front-End only, by means of a list of buttons including tabs located above the ListView (and therefore outside the ToolBar). To do this, include

the **ShowAlternate=Folder** option. This function not being available in Windows Front-End, the option is ignored in this case and buttons are displayed. If you want to display tabs in Web Front-End and a DropDownList in Windows Front-End, you can include the **ShowAlternate=FolderOrDropDown** application.

To define alternative collections:

➔ Define in the ListView content map an element that has the **AlternateSelection** option. The identifier of this element will be that of the browsed collection. Options of such an element are:

```
Param(AlternateSelection{,Picture=<PicId>}{combinateOptions}{plugin})
```

<PicID> picture associated with the button. If this option is not specified, the picture defined for the collection will be displayed.

combinateOptions concerns combine alternative lists, see 3.5.3.1.2.7.

Plugin enables definition of a specific macro enabling population of the collection to be displayed, see 3.5.3.1.2.8

Column Filter enables definition of generic column filtering specific to the alternative Selection. This option is of form:

```
ColFilter=NoDefColumn|StandardColumn|AssocColumn|None
```

When this option is present, it replaces the column filtering defined on the ListView (**NoDefaultColumn**, **StandardColumn**, **AssocColumn**). The value **None** deactivates filtering defined on the ListView.

When alternative collections are defined for a ListView, the ListView identifier itself is no longer used to define a collection, except if for questions of visibility none of the alternative selections is available. In this case the ListView becomes a standard list. However, visibility of the ListView itself will be subject to availability of this identifier.

Although this ListView can display several distinct collections, it has only one content map. Elements contained in this map (additional columns, buttons) may apply to only one of the alternative collections, and filtering should be defined so that they are only taken into account in suitable collections. Filtering is carried out by means of the **ForAlternate[*collectionID{,CollectionID}*]** option, which should appear in the element to be filtered. If this option is specified, the element (column or button) will only be visible when a *collectionID* defined in the option corresponds to the identifier of the current alternative collection.
Note: in Windows Front-End, the platform does not allow dynamic hide of a button in a toolbar: inappropriate buttons will not be hidden but simply disabled.

3.5.3.1.2.7 Combined alternative lists

This system is a variant of alternative lists, affecting their selection mode. It proposes replacement of RadioButtons by CheckBoxes. In this case, the total number of alternative selections corresponds to the number of possible combinations. In this system, the total number of alternative collections therefore corresponds to the square of the number of CheckBoxes.

Example: two CheckBoxes enable selection of four collections (no selection, first button selected, second button selected, both buttons selected).

To specify a combined alternative list:

➔ define as many **AlternateSelection** elements in the content map as there are possible selections, minus one: the identifier associated with the ListView is that of the collection used when none of the CheckBoxes is selected.

We then distinguish between elements corresponding to unitary collections (corresponding to click of a single button) and elements corresponding to combined collections (corresponding to a combination of buttons).
Elements corresponding to unitary collections have the **PushMask=n** option, where n is the bit value (1,2,4,8).
Elements corresponding to combined collections have the **MaskValue=c** option, where c corresponds to the combination of buttons.

Example: when there are two buttons:

- ***PushMask=1*** *indicates the collection activated by selection of the first button*
- ***PushMask=2*** *indicates the collection activated by selection of the second button*
- ***MaskValue=3*** *indicates the collection activated by selection of both buttons*

(If no button is selected, the collection activated corresponds to the identifier associated with the ListView).
These ListViews necessarily using CheckBoxes, the **ShowAlternate** option is not taken into account.
Note: the generic system of display of inherited and substituted objects, which applies as standard to all ListViews configured by a MetaAssociationEnd for which inheritance of variations is enabled, automatically uses the system of combined alternative lists. This system is exclusive, and will be disabled if alternative collections are explicitly defined for the ListView.

3.5.3.1.2.8 Specific collections: plug-ins and ERQL queries

You can define a ListView that does not browse a collection defined in the metamodel. Two cases may require such a system:

- *You may want to display the result of a dynamically defined ERQL query.*
- *You* may want to define a collection defined by a Macro.

To display the result of an ERQL query:

➔ Supply the ListView with the **RequestMode** option.

In this case, the identifier of the ListView must correspond to a property of type VarChar. The text contained in this property should correspond to an ERQL query, which will be evaluated when completing the ListView: the query result is then displayed in the list. In HOPEX 1.0 version, queries with parameter cannot be executed.

To display a collection calculated by a macro:

➔ Supply the ListView with the option:

```
PlugIn<macroID{:InitString}>
```

macroID corresponds to a Macro identifier. This macro should implement a standard collection as follows:

```
Function GetStandardCollection(      MegaObject as MegaObject,
                                     RetType as Variant,
                                     InitString as String) as MegaCollection
```

MegaObject is the source object of the ListView

InitString is the character string specified in the option

As *InitString* can be anything, it is possible to generate pages with macros having dynamically defined behaviors.
Another difference between such a Macro and one used in a Query or *AbstractCollection*, is that this macro can use the association browsed by a source object of the ListView. This is in fact the case for the *BridgeCollection* macro supplied as standard by MEGA, which can be used in all ListViews. This macro enables display of objects according to two links, one of which is from the source object of the ListView object.

```
PlugIn<BridgeCollection:MainMAEID,SecondaryMAEID>
```

In this case, the collection browsed is oObject.getCollection(«MainMAEIe»)
If the ListView object is derived from browsing of a link (oObjet.getSource.Exists): we therefore have the collection oObject.getSource.getCollection(«SecondaryMAEId»).
In this case the attributes of the MetaAssociation SecondaryMA are added as columns in the ListView, and we search for the corresponding values of these attributes in the second collection.
If the object is not a source object, only the first collection is displayed.

3.5.3.1.2.9 Web Front-End specificities

ListViews displayed in Web Front-End benefit from functionalities not accessible in Windows Front-End. The main innovation is that lists are by default presented in paginated mode, enabling faster loading. It is also possible to implement filtering via columns. Finally, presentation mode benefits from more comprehensive configuration.

- Display of alternative collections (**ShowAlternate**) is more complete, since it uses Folder mode.
- The **hidePagingToolbar** option deactivates pagination. It can be used on lists which we know will contain only a few objects.
- **hideLabelButton=1**: in Web Front-End, toolbar buttons have a label. This option enables their removal
- **GridMode**: indicates that use of ListView is data entry oriented. In particular, cell modification access is simplified.

3.5.3.1.2.10 MultiSelection list

The **MultiSelection** option enables definition that a ListView is multiselection. This new operating mode has been introduced in HOPEX.

In this mode, the ‘'Property' command displays a box containing all columns in update not corresponding to Unique Index Attributes: this box enables simultaneous modification of all selected objects.

In Web Front-End, entry of a cell in this mode modifies the column value for all selected objects.

When the column is managed by a specific updateTool, the latter may not adapt to the fact that it is not instanced on a specific object, but on an object collection.

MultiSelection mode is not the default mode, but it is possible to configure the active PropertyPageViewPort to make it the default mode for all ListViews.

For this:

➔ Configure the PropertyPageViewPort concerned by inserting the following keyword in its *_Parameterization* text (as 2.3.4).

```
[ListViewDefault]
    MultiSelection=1
```

Note: with this configuration, certain ListViews may exhibit inconsistent behavior, in particular when they control elements, or when you have defined specific buttons in their toolbar that do not operate correctly in this mode. You can alleviate these problems by providing such ListViews with the **MonoSelection** option, which deactivates multiselection.

3.5.3.1.2.11 Specific Configurations

Filtering: the **ShowAbstract** option applies to object collections of an abstract class, and enables deactivation of filtering relating to visibility of the concrete class of the element, filtering applied as standard. For lists displaying system repository objects, the **NoMetaFilter** option enables deactivation of filtering relating to visibility of these objects.

Entry mandatory: in a wizard, it can be useful to impose constraints on a ListView. The Mandatory parameter, applied to the list, indicates that a selection must be made in the list to validate the page. A less restrictive option, **NotEmpty**, indicates that the list should not be empty for the page to be validated.

A list can be used specifically to reorder its elements. Any other form of update is prohibited, and it is not possible to resort the list according to a column. The new sort order is saved in the repository when the page is validated. This mode is activated by the **ReOrderDrop** option.

The pop-up menu of elements displayed in the ListView can be redefined. For this, define a *MetaCommand Manager* and associate this with the ListView by means of the **ExtraCommands=<CommandAccessorID>** option. Commands defined in this manager will be inserted in the pop-up menu of objects in the list.

3.5.3.1.2.12 Cell Edit

When you edit a cell of a ListView, you initiate a system called *in-place editing*. This triggers specific start of an editing tool superimposed on the ListView without directly affecting it.

This editing tool operates according to the updateTool defined for the column of the cell to be edited. If necessary, the UpdateTool will adapt to in-place mode; it can adapt its behavior when in-place mode imposes data entry space limited to cell size.

A column declared in read-only does not allow activation of editing of these cells (excepting the particular case where the AttributeControl enables actions other than cell modification; in particular, this is the case for editMenus displaying the object menu).

The **NoInPlaceEdit** option enables deactivation of cell editing for the complete ListView

3.5.3.1.2.13 List view content refresh

ListViews can display collections of different types: certain are "live" and can react directly to repository updates that could modify their content; this is the case for example for collections derived from a non-calculated MetaAssociationEnd. Other collections however cannot react to these updates, as for example the result of an ERQL query. The query must be run again to reflect the consequences of these updates in the list.

The ListView cannot know behavior of the collection displayed at update, and using an option you should indicate to it the required refresh modes if necessary. Available options are:

RefreshOnChange,RefreshOnInsert,RefreshOnDelete: these options indicate that when the collection is modified, we must reselect before refreshing content.

RefreshOnCommand: indicates that the **Refresh** function applied to this ListView results in reselection of this list. Otherwise, the **Refresh** function only results in fetch of the existing collection, without reselection.

Reselection of a collection can be costly in terms of performance, and options should be specified carefully.

3.5.3.1.3 Options

Column Filter Definition

NoDefaultColumn: only the naming attribute is displayed.

StandardColumn: displays only standard columns, other presentable columns are initially hidden.

AssocColumn: displays the naming attribute and properties of the browsed association.

ReOrder: orders columns according to content map.

RefreshColumns: recalculates column list when page is refreshed.

Definition of image associated with each listed object:

NoLineBitmap: no image.

BitmapSelector=*PropID*: defines property serving to calculate image.

BitmapStamper=*PropID* {,*PropID*}: adds stamps on image

Definition of initial sort:

NameSort: sort by name, order number is not used.

OrderOverloadProp=*PropID*: enables order number attribute redefinition.

Refresh collection: indicates when collection should be re-queried.

RefreshOnChange

RefreshOnInsert

RefreshOnDelete

RefreshOnCommand

Configuration of background loading of the ListView (Windows Front-End).

NoBackgroundLoading: loading is synchronous.

NoBackgroundEscape: background loading cannot be interrupted by Escape key.

Defining Toolbar:

NoBar enables toolbar hide.

ToolBar[] enables hide/show of ListView standard buttons.

hideLabelButton=1 (Web Front-End only): does not display button labels.

ExportCommands=P|X|0: export buttons display commend (0: no button, X: Excel, P: Pdf).

Specific Selections:

RequestMode: Selection obtained from ERQL query.

PlugIn<*MacroId{:InitString}*>: Selection obtained from macro.

Filtering objects displayed in list:

ShowAbstract: deactivates filtering related to visibility of concrete class of the element

NoMetaFilter: deactivates filtering related to visibility of system objects.

Alternative collection selection display mode:

ShowAlternate=DropDown | Folder | FolderOrDropDown

Other options:

hidePagingToolbar: (Web Front-End only): deactivates pagination

MultiSelection: activates multiselection mode

NotEmpty: validiity condition of the ListView imposing that this should not be empty

EnableDropOrder: ListView designed for sorting its elements

NoInPlaceEdit: deactivates cell editing

GridMode: (Web Front-End) indicates that the ListView is principally designed for editing.

AssociativeCollection: collection obtained from RelationShip object
ExtraCommands=<CommandAccessorID> redefines menu of listed objects

3.5.3.1.4 Properties

Value: this property contains absolute identifiers of the selected object in the ListView. In the case of a list with multiselection, the concatenation of selected absolute identifiers is returned, separated by a space. In assigning the value you modify the selection.

3.5.3.1.5 Notifications

Select: sent when the selection changes.

Open: sent at double-click (Windows Front-End only)

BarClick: sent at click on one of the specific buttons of the toolBar. The **PushedButton** method enables determination of which button has been clicked.

3.5.3.1.6 Component Methods

Specific methods of the AttributeControl can be called from the component object (AttributeControl.Component)

.PushedButton As Variant: this function should be called at processing of a **BarClick** notification and returns the identifier of the clicked button. This identifier corresponds to the Item of the button or to a conventional button identifier (for example for PDF Export)

.GetCollection As MegaCollection: returns the collection displayed in the ListView

.GetAlternate As Integer: returns a distinct number according to the alternative collection displayed.

.Fetch(JSONCommand As String) As String: Obtains content of the list as it is displayed in the ListView. This function returns a JSON format string. The Command is a JSON format string indicating what you want to obtain. If you want to obtain the complete list with all columns displayed, this parameter should be exactly **{"currentView":true}**. If not, it should contain information relating to what you want to obtain conforming to the **hdr** structure as described below. This can be useful if you want to partially retrieve the collection by specifying fields **firstLine** and **lineCount**.

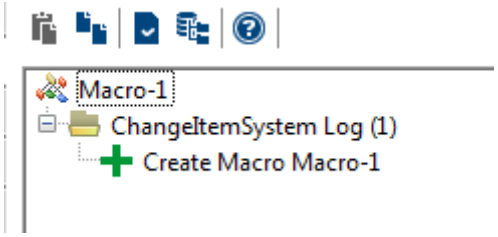
The returned JSON contains two structures:

- An **hdr** structure containing restored data (list of columns **columns**, sort method **sortCriteria**), as well as the number of restored rows **lineCount**, the number of elements of the collection **total**, and the index of the first restored row **firstLine**.
- A **content** structure, table containing the list of rows. Each row is a structure for which conventional fields are specified **id** (object absolute identifier, **classId** (object class), **image** (object image moniker) and as many fields as columns specified for the object (the name of the field corresponding to the column identifier) of which the value corresponds to the value to be displayed (when the column displays an image, its value is an object containing the image moniker and the string to be displayed). If the ListView is multiselection, we also indicate if the object is selected with the boolean field **isSelected**

Example of a restored JSON:


```
{
 "content":[

{"id":"f10000000b20","classId":"O20000000Y10","image":"~gYNHjd5mzKQ1","2100000009
00":"Commentaire"},

{"id":"f20000000b60","classId":"O20000000Y10","image":"~gYNHjd5mzKQ1","2100000009
00":"Journal"}],
  "hdr":  {
   "firstLine":0,"lineCount":2,"total":2,"noEmpty":true,

"columns":[{"id":"210000000900"},{"id":"Q10000000f10"},{"id":"S20000000n50"}}],
   "sortCriteria":[{"id":"210000000900"}]
  }
}
```


3.5.3.2 TreeView



The TreeView element enables tree display. Unlike MetaTree, it is not necessary to define a MetaTree to use it. In addition, it also operates with informal objects.

This object enables display of Collections from the object associated with the element. These collections are presented in the form of folders. In expanding these, you can list elements of a collection.

3.5.3.2.1 Specifying collections to be displayed

The identifier associated with the TreeView can be:

- the identifier of an _Operator.

  In this case Collections are filtered according to behavior of this operator related to the collection.

  By default, only collections with "Abort" behavior are filtered. If you specify the **ScanStandardOp** option, "Abort" and "Link" behaviors are filtered. With the **ScanDeepOp** option, only "Deep " behaviors are displayed.

- the identifier of a MetaAssociation type.

  In this case collections of this type are displayed.

- the identifier null (~000000000000[Null]).

  In this case no collection is filtered.

To explicitly define collections:

➔ Use the content map associated with the TreeView.
Identifiers associated with elements of this map can be *MetaAssociationEnd* or *Abstract Collection* identifiers (therefore in particular of *queries*).

When an object is expanded in the tree, we collect in the content map the elements of which the source is compatible with this object, in order to create the list of collections to be displayed.

Whatever the collection display definition mode, only those collections visible from the user technical level viewpoint, and from the viewpoint of his/her associated rights are displayed.

3.5.3.2.2 Tree configuration

3.5.3.2.2.1 Multi-level trees

By default, the tree only expands at a Collection level. The **IsDeep** option enables specification of a multi-level tree. In this case, the filtering mode on child levels is identical to that of the root; in particular the content map is also used for child elements.

To increase or decrease the number of expanded element levels at element display:

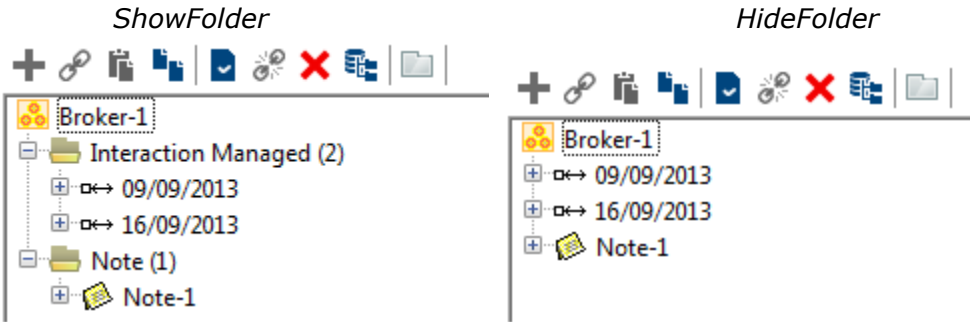
➔ Use the **UnfoldedAtStart=n** option, where n is the number of levels to expand (by default, n = 2).
This option should be used with caution, since the number of levels open to initialization significantly affects tree loading time…

When the tree is configured with a content map, it is possible for only certain types of element to be multi-level by specifying the **IsDeep** option on the element of the map itself. On the other hand, **IsLeaf** on an element enables deactivation of the **IsDeep** option global to the tree.

3.5.3.2.2.2 Direct display

When there are no ambiguities on displayed objects, it is possible not to display folders corresponding to collections, but to directly display objects listed in collections. For this use the **HideFolder** option.

ShowFolder *HideFolder*



When you expand the element, all visible collections are therefore expanded; nodes however remain grouped by collection.

This option results in disappearance of folder level, commands appearing on folders (in particular create and connect menus) are no longer accessible; in addition it is no longer possible to explicitly drag-and-drop objects in the folder representing the collection. **HideFolder** mode must therefore take account of this.

When the tree is configured with a content map, it is possible for folders to be hidden only for certain types of element, by specifying the **HideFolder** option on the map elements themselves. On the other hand, the **ShowFolder** option on an element enables deactivation of the **HideFolder** option global to the tree.

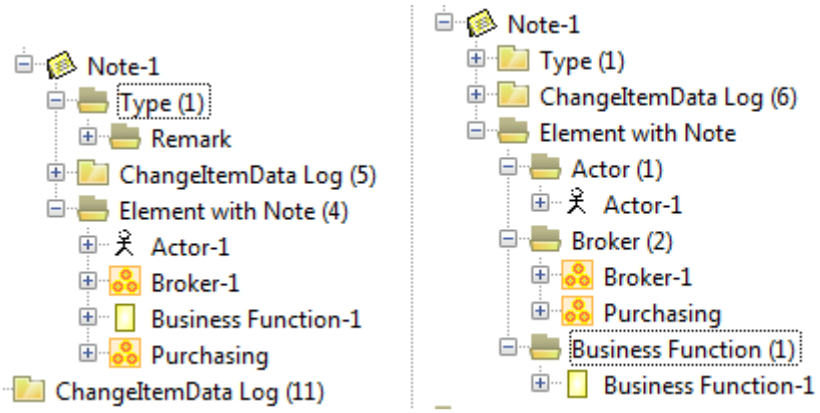
3.5.3.2.2.3 Displaying generic collections

When target of a browsed collection is an abstract MetaClass, objects presented under a folder can be of different types. By default, the tree displays objects on a single level, without grouping by MetaClass.
To group them by MetaClass:

➔ Use the **ShowConcrete** option.

In this case, another level is added, grouping child nodes by MetaClass.

ShowConcrete



When the tree is configured by a content map, you can define this display specifically for each collection using the **ShowSpecialized** and **HideSpecialized** options specified on the map element.

3.5.3.2.2.4 Object sort and display

As standard, elements expanded in the tree are sorted according to the order number defined for the collection (generally the Order attribute). When two elements have the same order number, the second sort criterion is the label associated with the node. This label is itself the collection naming property.
For folder level, we use sort order defined for MetaAssociations in the Metamodel.

To display a label other than the object name:

- ➔ Use the option NameComputer=<PropID>.
  In this case, the PropID property will be used as label. When the tree is configured by a content map, you can deactivate this behavior for a given element by specifying the **NoComputedName** option.
- ➔ In this case you can also deactivate sorting by order number using the option **NoOrder** (objects will then be ordered lexicographically).

Folders are represented by homogeneous folders.
To differentiate their display:

- ➔ USe the **DefaultFolderBitmap** option.
  In this case the color of the folder depends on collection type (major, minor, calculated).

3.5.3.2.2.5 Defining the toolbar

The toolbar of a treeview cannot be extended and displays only standard commands. These commands differ depending on whether the selected element is a folder or an object. In particular, commands "New", "Connect" and "Paste" are only available at Folder level, and are grayed when the selected node is a MEGA object.

Standard buttons of a treeview are:

Create (**C**): Start the tool for creation of a new element in the collection corresponding to the selected folder

Link (**L**): Start the tool for insertion of a new element (connect) in the collection corresponding to the selected folder

Reorder (**R**): Starts the tool enabling reordering of elements of the selected folder according to standard order (see 3.5.3.1.2.3). This option is not present by default.

Destroy (**D**): Start the tool for deletion of the selected object. When a Folder is selected, proposes the mass deletion tool for all collection objects.

Unlink (**U**): Start the tool for disconnection of the selected object. When a Folder is selected, starts the mass disconnection tool for all collection objects.

Properties (**P**): Display of the properties dialog box of the object selected in the list.

Explore (**E**): Starts the explorer on the selected object or collection (available only on Windows Front-End).

HideEmpty (H): Enables show or hide of folders corresponding to empty collections. The **HideEmptyAtStart** option enables initial hide of empty collections.

The **ToolBar[]** option enables show or hide of elements of this toolbar. It is configured similarly to the ListView toolbar (see 3.5.3.1.2.5**Error! Reference source not found.**)

3.5.3.2.2.6 Notifications

Select: sent when a new node in the tree is selected.
Generally this notification is only sent when the selected node is an object. A folder does not correspond natively to an object, since it corresponds to a collection. It is however possible to make a folder selectable using the **SelectFolder=<CollectionID>** option. When this option is activated, an informal object is created corresponding to the folder description, to which is added the <CollectionID> collection corresponding to the object collection seen from the folder. This object is then considered as the selected object. Such a system enables specification of a user interface displaying a list corresponding to the list of objects of the Folder.

3.5.3.2.2.7 Options

ScanStandardOp, ScanDeepOp: enables modification of filtering of collections when the tree is configured by an operator.

IsDeep: multi-level tree

UnfoldedAtStart=n: number of levels initially expanded.

HideFolder: hides folders level

ShowConcrete: displays folder level for concrete MetaClasses

NameComputer=<PropID>: tree elements label calculation

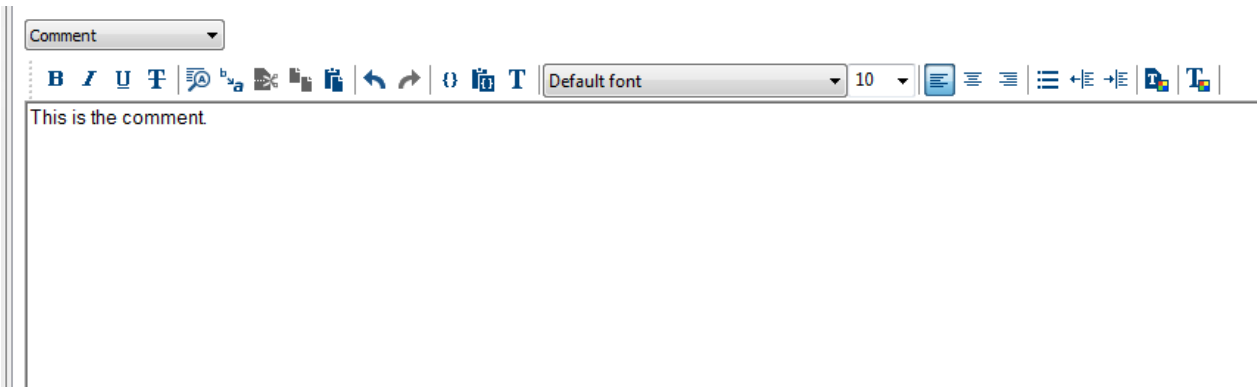
DefaultFolderBitmap: color display of folders.

HideEmptyAtStart: enables initial hide of empty collections.

ToolBar[]: specifies toolbar content.

SelectFolder=<CollectionID>: makes folders selectable

3.5.3.3 Text



This element enables display and editing of text type properties. The text editor displayed depends on the type of text associated with the property.

The identifier associated with the element corresponds to the editor property.

You can group editing of several texts in the same screen area. In this case a dropdownlist located above the edit area enables selection of displayed text. This is default behavior, when several texts are associated with the displayed MetaAttributeGroup.

When you want to explicitly specify a text grouping, you can do this using the content map of the element; for each map element there is a corresponding property to be edited. This map also enables forcing read-only display mode.

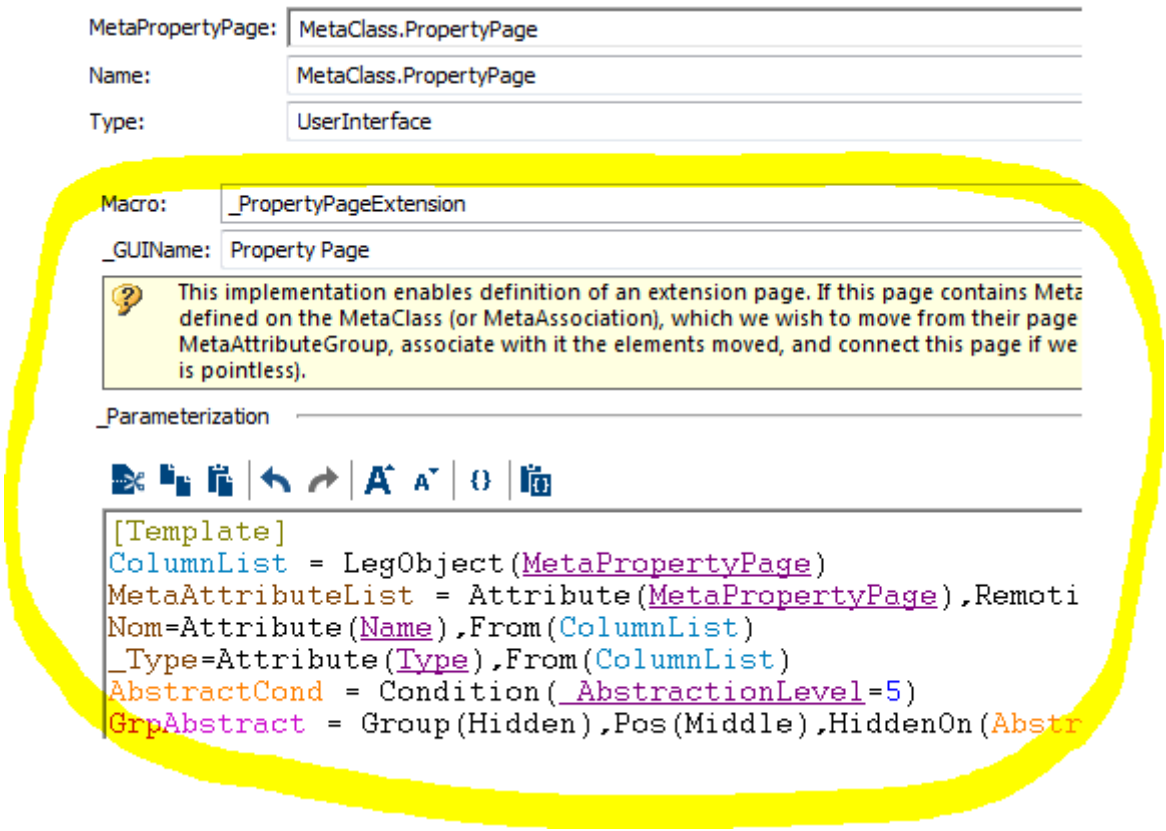
Options:

NoBar: removes the text editor toolbar.

Notifications:

Change: sent when text content has been changed

3.5.3.4 SubPage



An AttributeControl of" SubPage" type enables inclusion of one page in another. This system is particularly useful in the case of an AttributeControl controlled by a Map, since it elegantly manages the fact that objects of different types can be displayed differently in the same page. Content of the SubPage is dynamically calculated according to the object associated with the element. This calculation can also be executed in deferred mode, and therefore this system can facilitate form display (particularly in Web Front-End).

Typically, the identifier associated with the element corresponds to a MetaPropertyPage which is instanced on the element of the page. This identifier can however be of another type, depending on the options defined on the element, particularly the **Computed** element, which indicates that the page to be displayed is derived from a calculation. This option should be systematically present if you are not in the typical case.

If the MetaPropertyPage indicated in the element is not a standard property page of the object, you should include the **External** option.

The first calculation mode can be defined when the MetaClass of the object associated with the element is not fixed; in this case it is possible to define a specific MetaPropertyPage for each of the MetaClasses possible for the element, using its content map. Elements of this map should be of form:

```
SubPageItem=Item(<MetaClassID>),From(SubPageMap),Param(<PageID>,Default)
```

If the MetaClass of the associated object corresponds to <MetaClassID>, the SubPage displays <PageID>.

The **Default** option should be used when <MetaClassID> is an abstract MetaClass; in this case, if no sub-element corresponds to the class of the element, a second pass will enable association of the element with a <PageID> if the MetaClass of the element is a sub-class of <MetaClassID>.

Another calculation uses the _Type associated with a standard page of the object associated with the element. For this include the **Type** option, and associate with the element an identifier of _Type. In this case the object page corresponding to this _Type will be displayed (if several pages correspond to this _Type, the first one found will be displayed).

To request display of a standard page of the object:

➔ Specify the CLSID of the macro of this page (corresponding to **`MacroCLSID`** mentioned above), using the **CLSID={*clsid of the macro*}** option.

To request display of a virtual page including all object attributes:

➔ Use the **CompleteDescription** option.

When no page can be determined, or when no object is associated with the element, the SubPage displays nothing.

If size of the element has not been specified, the Control is resized to contain at least the page at its minimum size.

Options:

Computed, External, Type, CompleteDescription, CLSID={*clsid of the macro*}: options enabling definition of page to be displayed

Owner=off, **Name=off**: these options can be used when you display the object characteristic page and enable hide of Name and Owner fields of this page.

SetMinMax=1: indicates that the size specified for the element should be used to define size of the SubPage.

Maximized=0: deactivates BOTTOMALL style of control (this style is present by default)

Abstract: when this option is present, the object of the page corresponds exactly to the object associated with the element, even if instanced on an abstract class (for example when this object is controlled by the browsing path of a generic MetaAssociation. In the opposite case and therefore by default, the page is initialized with the concrete object.

Associative: enables display of an associative page. The object associated with the page corresponds to the object derived from the association browsed by the object of the element.

Component methods:

Specific methods of the AttributeControl can be called from the component object (AttributeControl.Component)

.SubPage As MegaPropertyPageComponent: returns the object corresponding to the displayed sub-page. It can be **Nothing** if no page was instanced for this control.

APPENDIX: COMPATIBILITY

This documentation is valid for HOPEX 1.0 SP 1 version.

In version HOPEX 1.0 the following points are not functional:

It is not possible to use viewPort to overload standard behavior of forms (see 2.3.4)

Keyword **Refresh(Always)** (see 3.3.1.4**Error! Reference source not found.**) is not systematically recognized and does not operate.

Notification **BarClick** of ListView (see 3.5.3.1.5): the notification is not named in HOPEX 1.0, its internal number 64826 should be used

The Value property of a viewer is not managed

Options **direction** and **AutoExpand** of the **SelPropagate** parameter of MetaTree

Options **DefaultFolderBitmap**, **UnfoldedAtStart=n** and **SelectFolder** of Treeview.

Options **Maximized** and **ValueFrom** of HelpComment

Options **SetMinMax** and **Maximized** of SubPage

Option **MultiLine** of RadioButton

Option **DirectMacro** of Viewer

Option **ValueFrom:<IdProp>** of Label

Option **AssocColumn** on ListView, and option **ColFilter** on alternative selections.

In updatesTools, options **ManageReadOnlyMenu**, **ManageValueID**, **SingleOnly** and management of specific sub-menus commanded by **SPECIFICCHILDMENU**.

The global MegaMacroData used in MetaCommands of UpdateTools is not available in Java.

HOPEX Forms Property Pages

The purpose of this document is to familiarize you with the customization of Properties pages of a MEGA object.

From knowledge to performance

Initializing the courseware

In order to be independent of evolution of the **MEGA** metamodel, this courseware is based on a specifically designed metamodel extension.

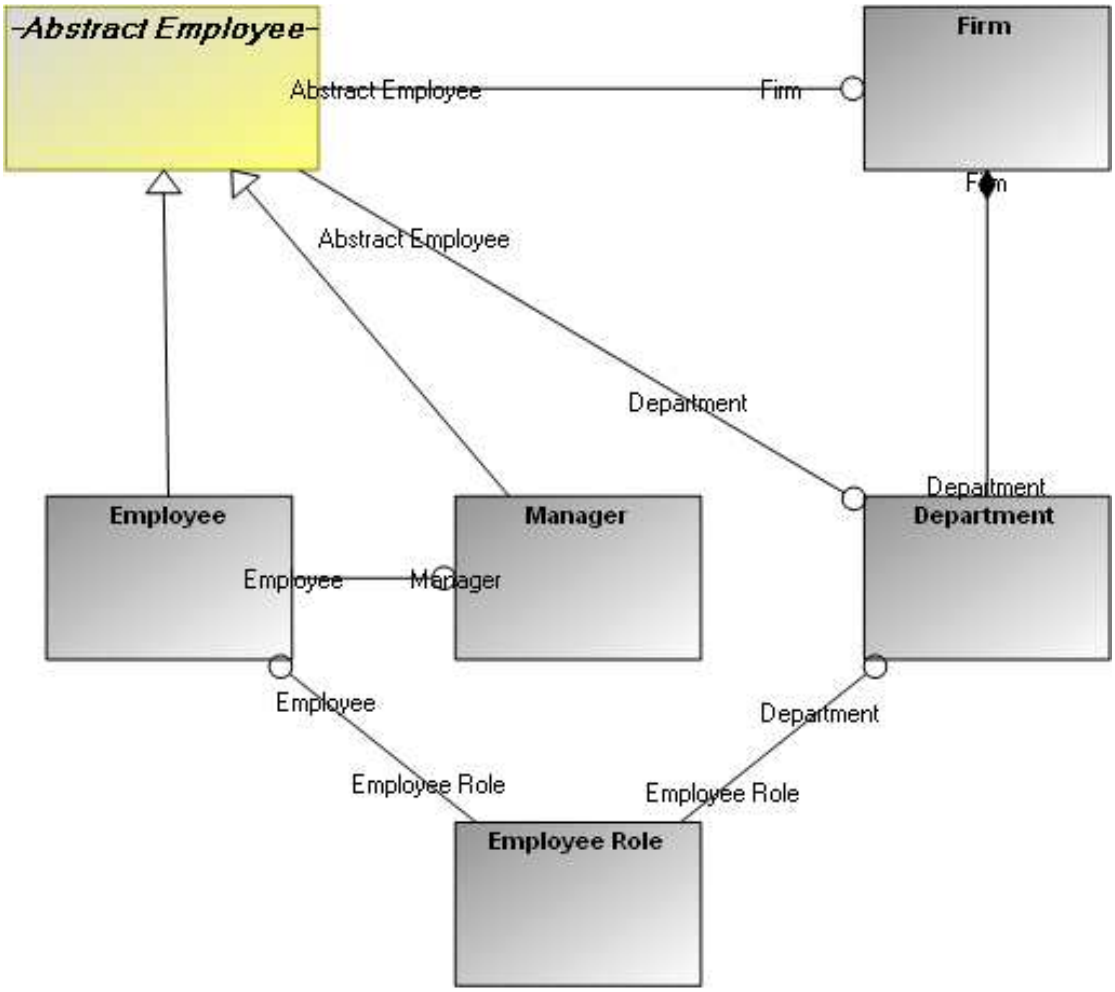
Before starting work, you must initialize your environment:

1. From MEGA, import into the system repository the command file below (this file is attached to PDF).

MetaModel Extensions for training courses.mgr

2. From the MEGA Administration module, recompile the metamodel.

Content of this specific metamodel is the following:


-Abstract Employee-
Abstract Employee
Firm
Firm
Abstract Employee
Department
Department
Employee
Manager
Department
Employee
Manager
Employee
Department
Employee Role
Employee Role
Employee Role

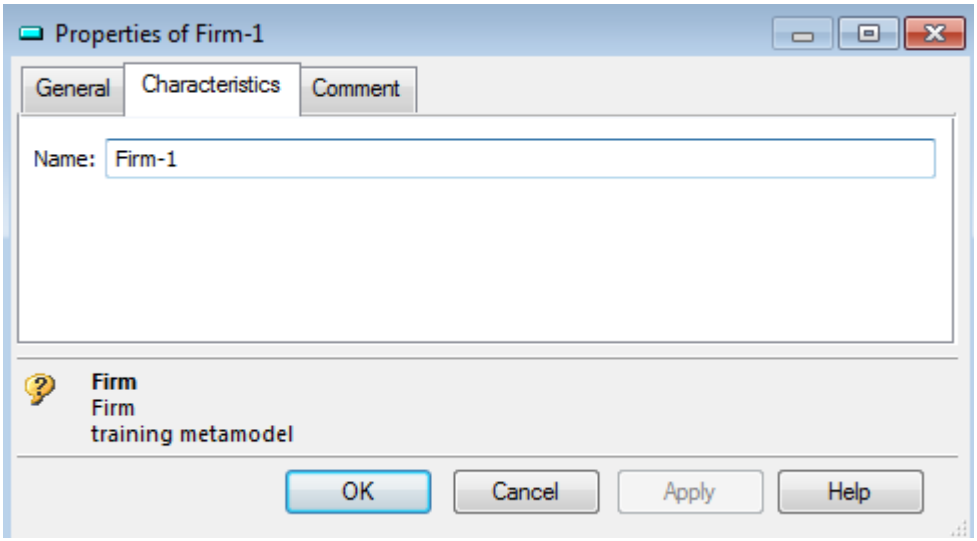
Object standard Properties pages

In the extension provided below, there is no customization relating to Properties pages of defined MetaClasses.

However, when you explore an object of this metamodel, you will find that it has a Properties window already structured.

Example with MetaClass *Firm*:

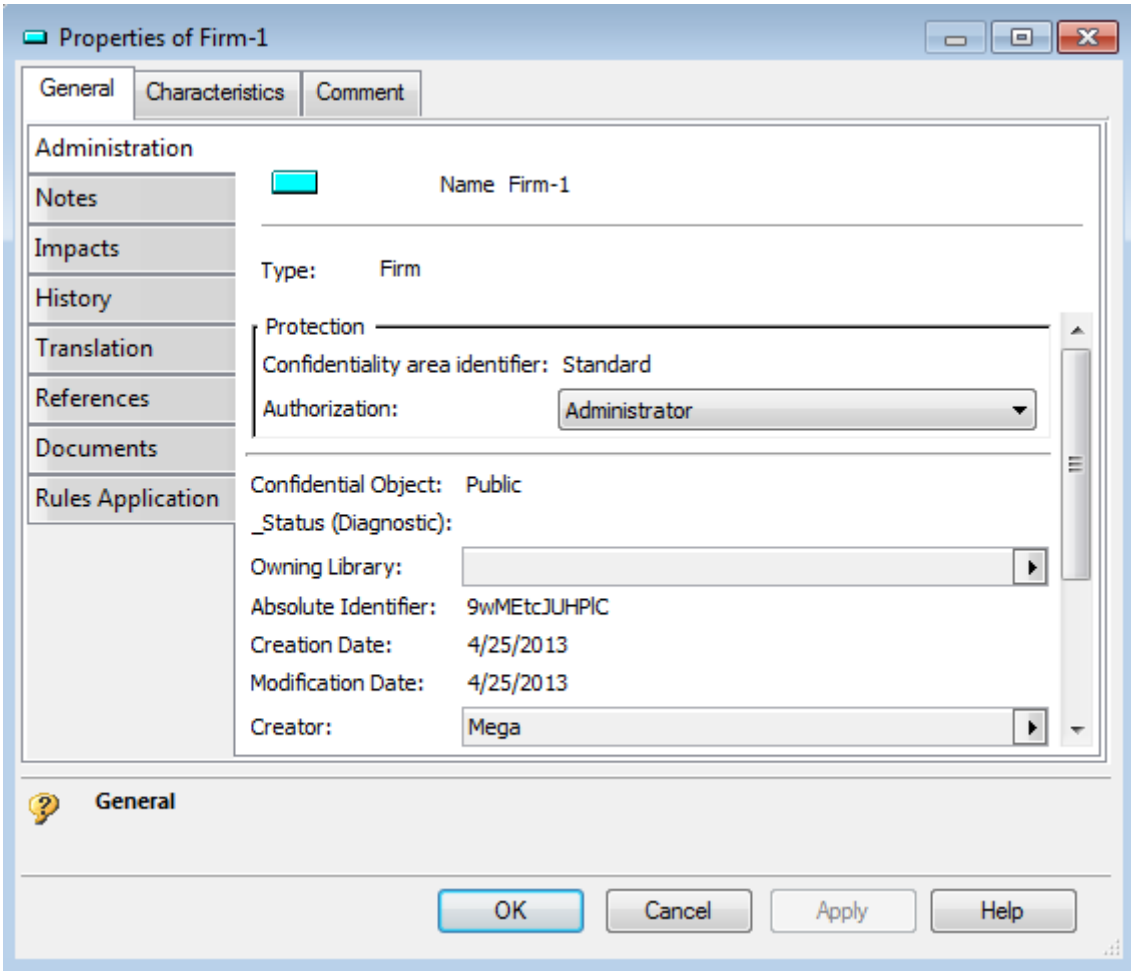
1. Login to MEGA.
2. Run the explorer from the desktop  .
3. If no *Firm* has previously been created, select **Explore > Create** and select the Firm MetaClass.
4. Display Properties of this object.



This MetaClass contains only two "business" MetaAttributes, Name and Comment.

- The name has been automatically entered in the **Characteristics** tab, which is the main tab of the object.
- The comment is located in the **Comment** tab, designed to host all MetaAttributes of Text type (varchar).

General tab already includes subtabs which have been automatically inserted by **MEGA**.

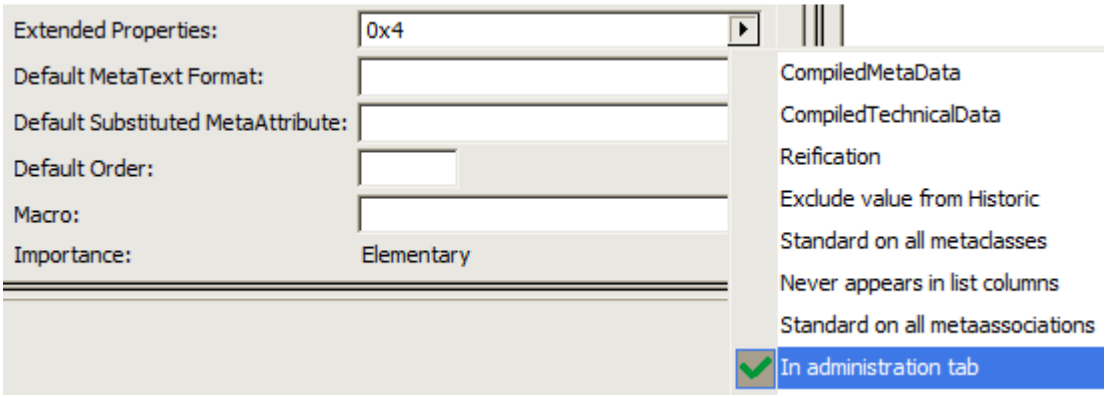


The **Administration** subtab groups non-business MetaAttributes of the MetaClass, which are automatically associated with the latter.

The **Translation** subtab handles the issue of translatable attributes.

Other subtabs are added automatically according to standard roles played by the MetaClass.

> Note: administration MetaAttributes are marked with a flag in their extended properties.



You can see that as standard:

- MetaAttributes of the MetaClass are grouped according to their type.
- Tabs are associated with attribute groups.
- Standard roles of the class (for example inherited from abstract classes) can produce to tabs.

  Note for example that the **Impacts** page appears as a function of behavior of MetaAssociations of the MetaClass relating to the "Impact" operator.
- Automatic tabs are inserted (for example **History** and **Rules Application** tabs).

For more information, see "*HOPEX Forms*" document.

Object Properties pages customization principles

Customization of an object Properties pages consists of being able to redefine all or part of this standard. To do this, you must be able to:

- add specific pages
- modify or substitute generic pages, either to add elements, or to modify their appearance.

Two concepts have been introduced in the MetaModel to meet these demands:

- MetaAttributeGroup
- MetaPropertyPage.

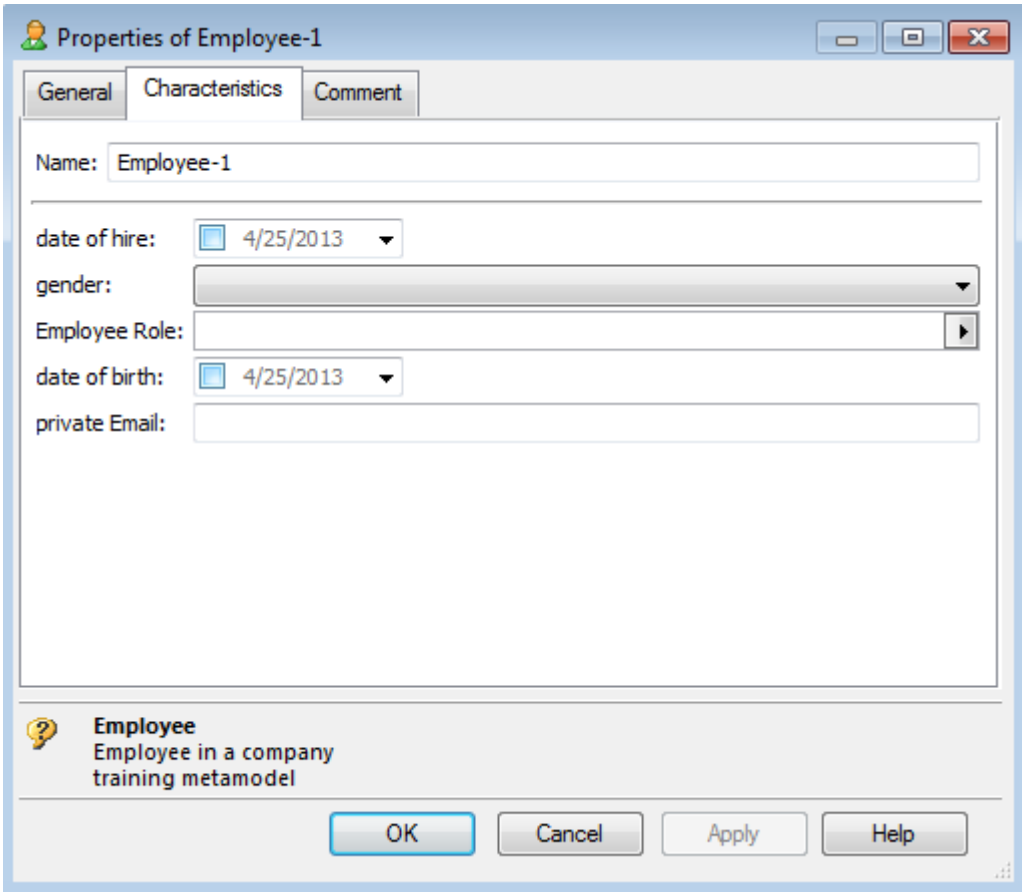
MetaAttributeGroups enable redefinition of the grouping of the MetaClass attributes. Attribute is interpreted here in its broadest sense, including the concepts of MetaAttribute, TaggedValue and LegAttribute:

- a taggedValue is a named value which you can associate with a MEGA object (on condition that it inherits from the "element with tagged value" MetaClass) without creating a MetaModel extension;
- a LegAttribute enables inclusion of a role (ie. MetaAssociationEnd), of maximum cardinality 1, as an attribute of object type.

Usually, any attribute of the MetaClass associated with a specific MetaAttributeGroup of this same MetaClass is removed from the generic grouping mentioned in the previous chapter.
To insert specific elements in the object Properties page which are not attributes of the MetaClass, or to redefine display or behavior of these attributes, use a MetaPropertyPage.

Moving attributes using a MetaAttributeGoup

The objective of the following customization is to clarify use of a MetaAttributeGoup. To do this, consider the Characteristics page of the **Employee** MetaClass:

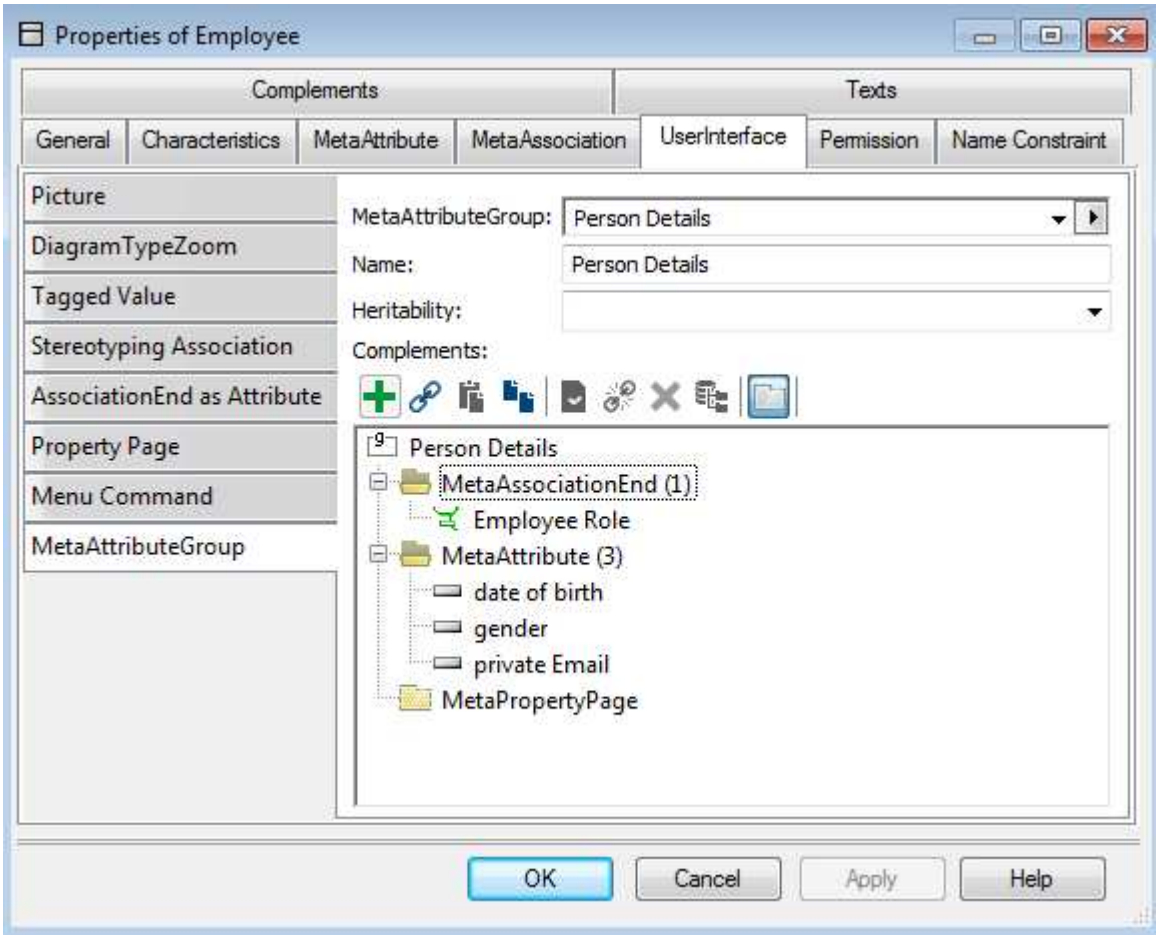


The objective of customization is to remove attributes *gender*, *date of birth* and *private Email* (which are MetaAttributes) and *Employee Role* (which is a LegAttribute) from the Characteristics page and include these in a specific page.

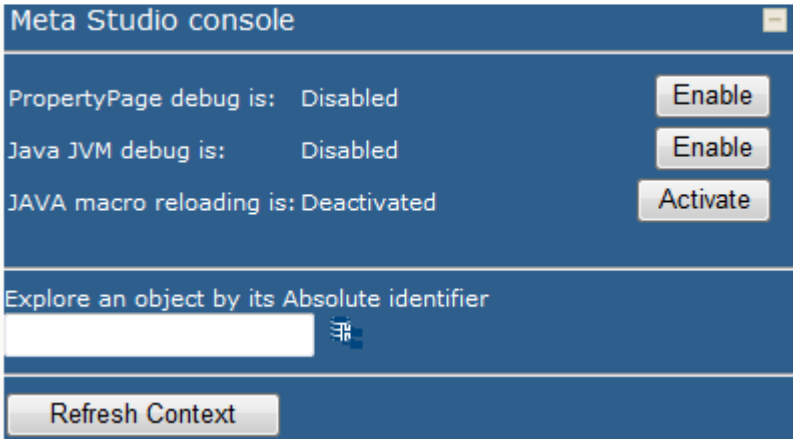
To do this, create a MetaAttributeGoup with which associate these attributes.

1. Open the Properties window of the **Employee** MetaClass.
2. In the **User Interface** tab, select the **MetaAttributeGroup** subtab.
3. In the **MetaAttributeGroup** field, click the button with the right-facing arrow and select **New** to create a new MetaAttributeGroup.
4. Enter the **Name** of the MetaAttributeGroup (for example "Person Details") and click **OK**.

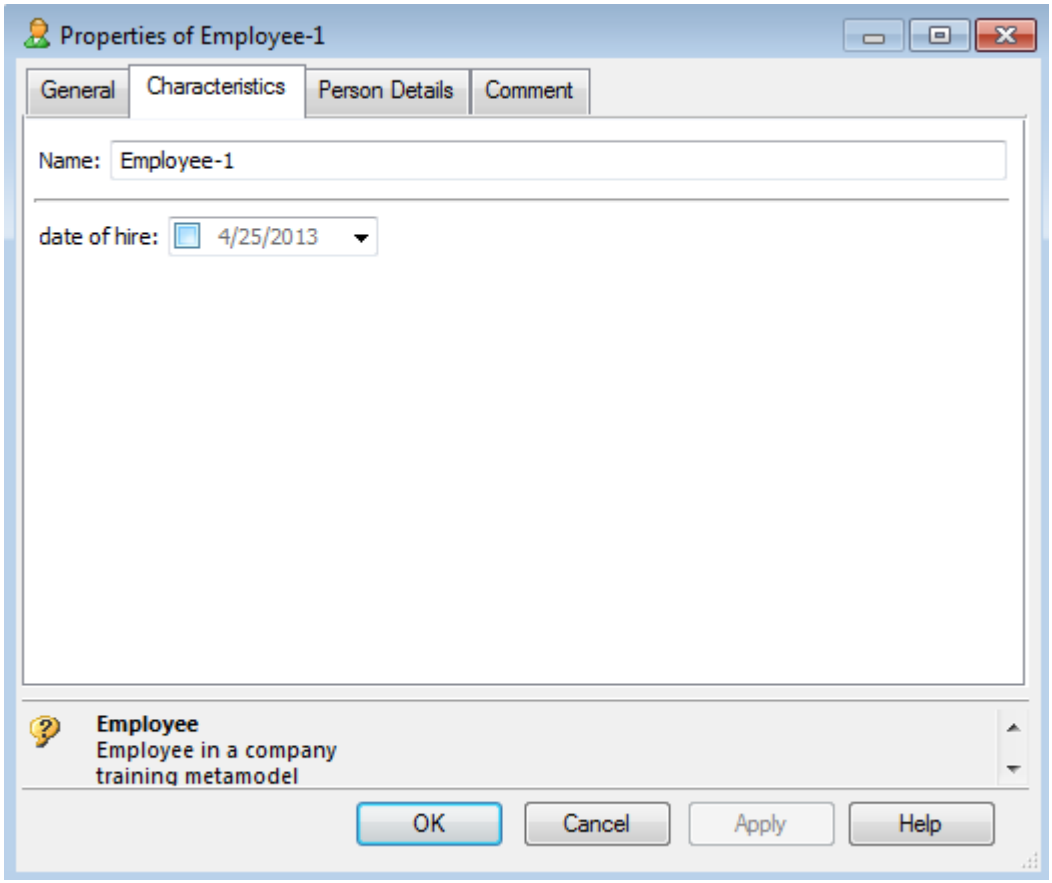
5. In the **MetaAttributeGroup** field, select the MetaAttributeGroup you just created (for example « Person details »).
6. Associate with this MetaAttributeGroup:
    a. the MetaAttributes (date of birth, gender, private Email): copy/paste from the **MetaAttribute** tab, **SuperMetaAttribute** subtab.
    b. the MetaAssociationEnd (Employee Role), which you copy in the same way from the **AssociationEnd As Attribute** tab.



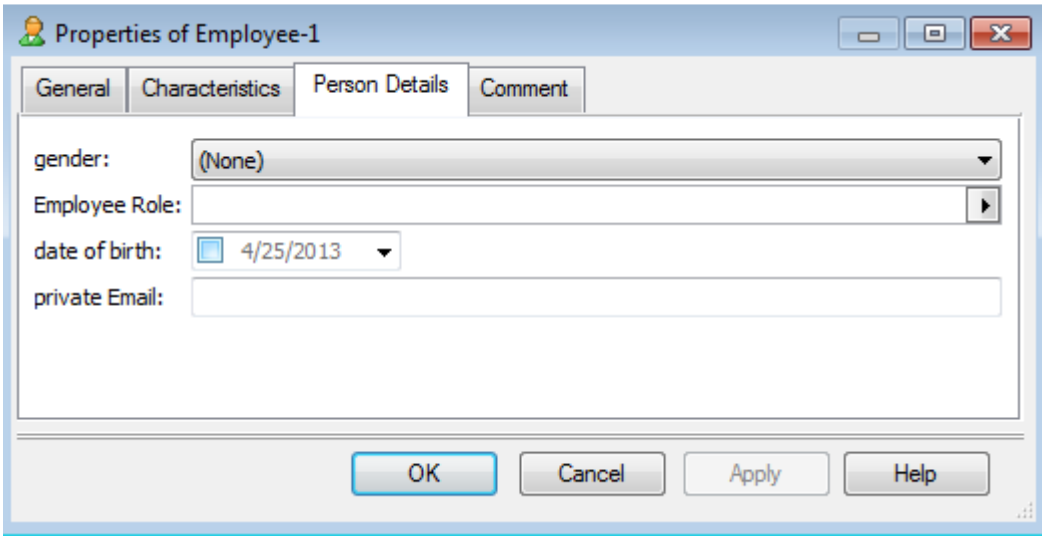
7. Refresh the metamodel view so that these modifications are taken into account. To do this:
    a. Display the MEGA home page.
    b. Display the **Meta Studio Console** element (if not already visible).
    c. Click **Meta Studio Console**.
    d. Click **Refresh Context**.



8. Open the Properties window of an Employee. This shows that:



Attributes associated with the MetaAttributeGroup have disappeared from the **Characteristics** tab. They appear in a new tab, created automatically from the MetaAttributeGoup, and carrying the same name "Person Details".



Note: When the MetaAttributeGroup has a **GuiName**, it is this attribute that is used to name the tab.

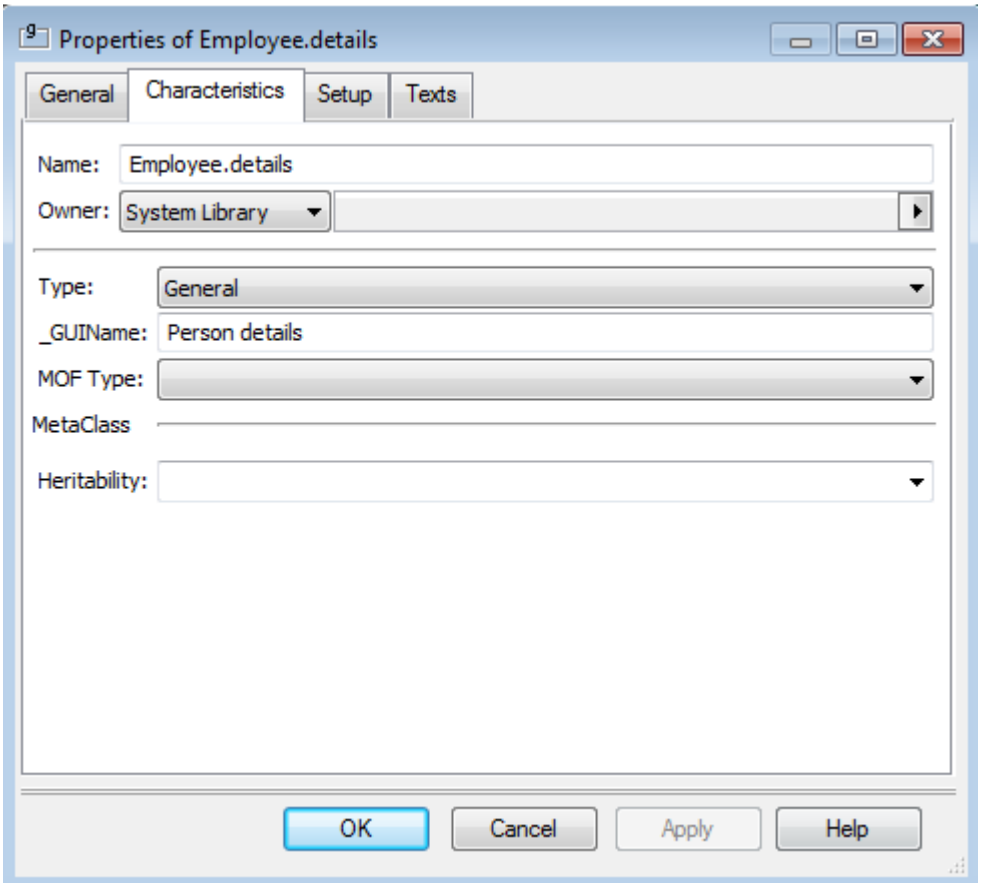
Grouping pages as subtabs

Consider that pages previously defined are satisfactory, but you want to include "Person Details" in the **General** tab.

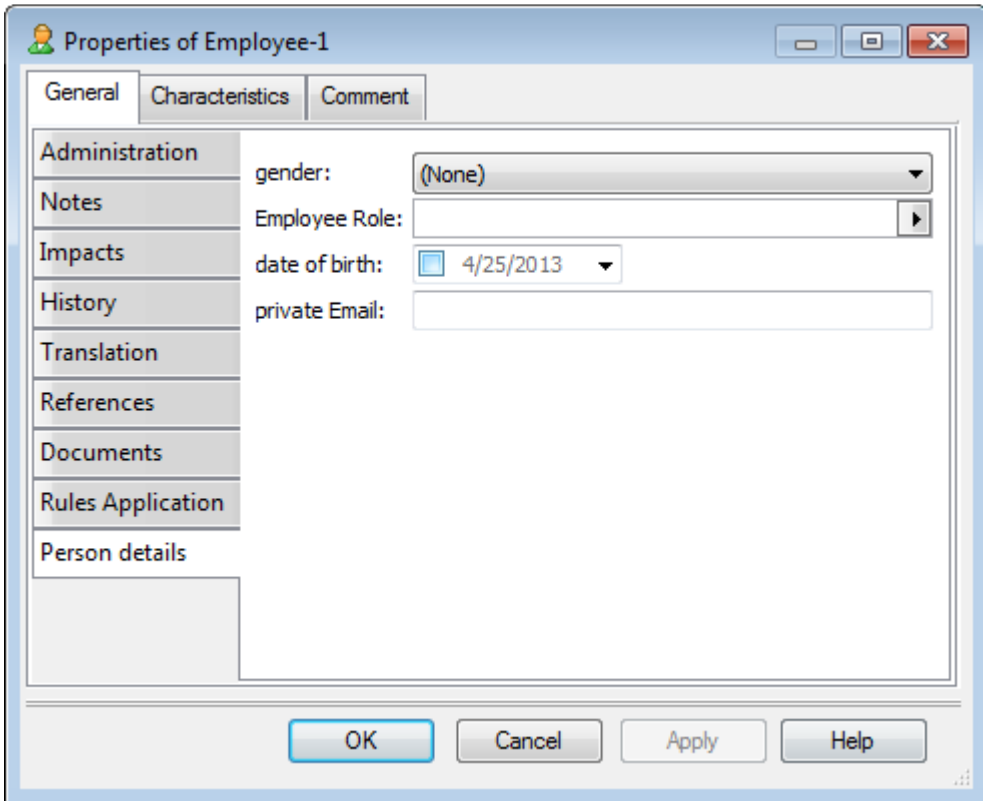
To do this, be aware that pages are grouped in this way according to their Type. It is the name of the Type which is used to name the main tab. When a page is automatically deduced from a MetaAttributeGoup, it is the _type associated with the MetaAttributeGroup which is considered.

Define your MetaAttributeGroup **Type** as « General ».

You can rename the MetaAttributeGroup and define the name displayed in its _GuiName.



After context refresh, the Properties window of an Employee becomes:



You can add new object types.

Reordering pages

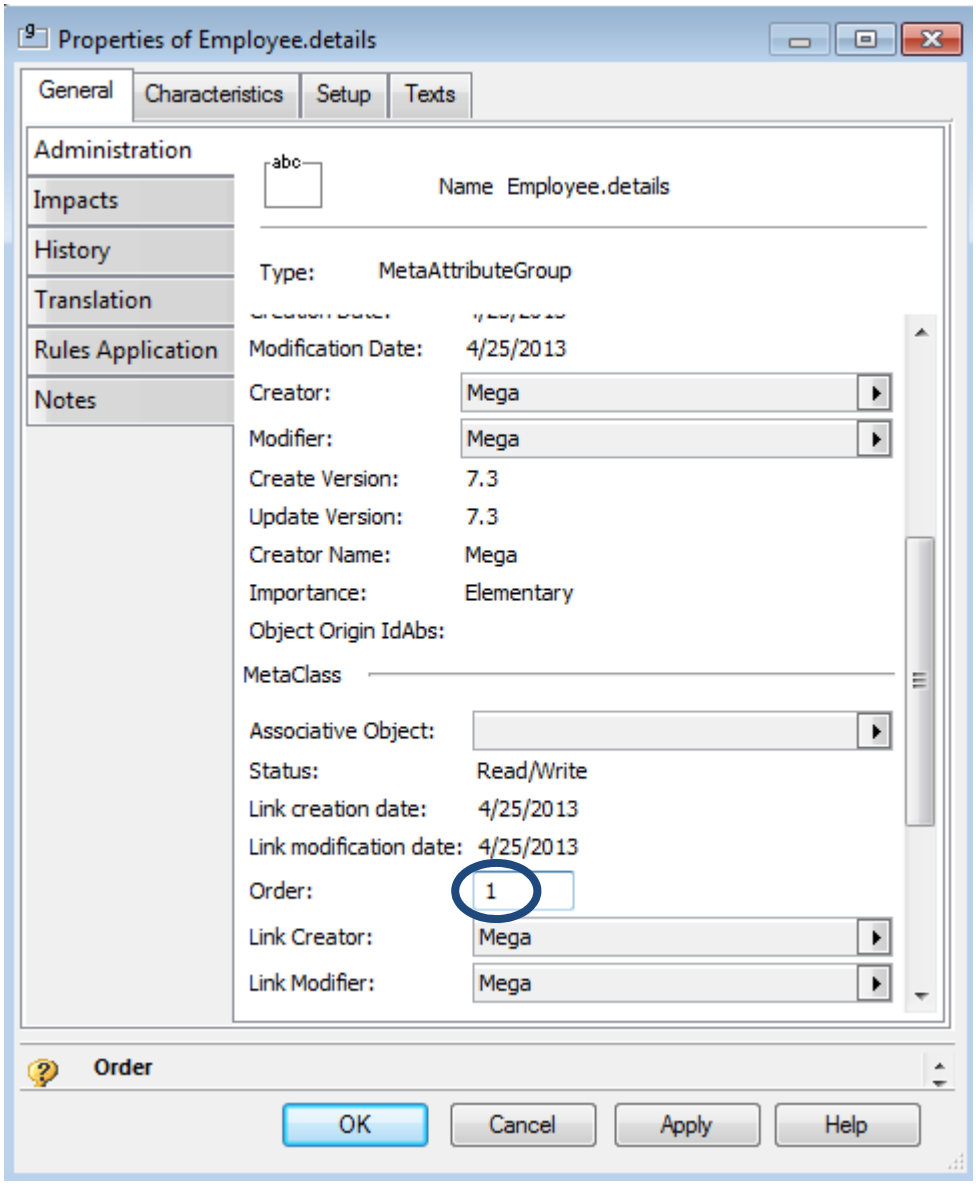
The above arrangement is almost satisfactory, but you want the 'Person Details' tab to be placed first in the list.

To do that, you must know the following rules:

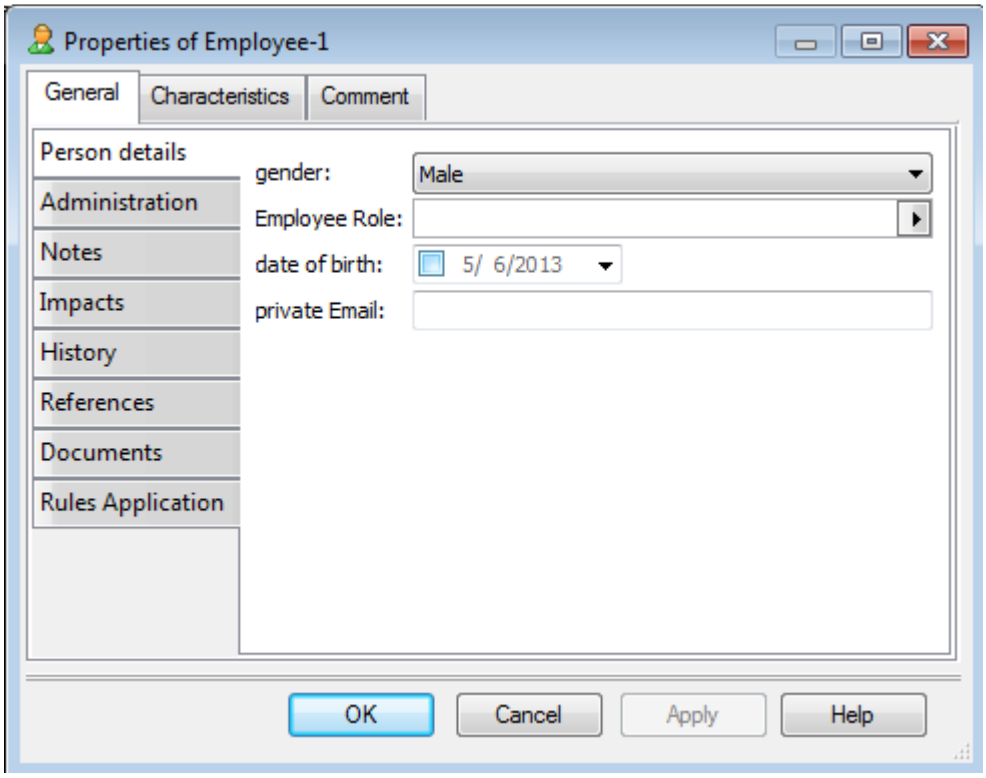
- Each Properties window page has an associated order number which defines the tab order.
- Depending on page implementation, this order number can be specified in different ways.
- Each generic page has an associated intangible specific order number. For more information, see "*HOPEX Forms*" document.
- When a tab groups several pages, its order number is the smallest order number from among those of the pages it contains.

- When a page is automatically deduced from a specific MetaAttributeGroup, its order number corresponds to the value of the Order' attribute between the MetaClass and the MetaAttributeGoup

In the present case, you can modify the order number between the MetaAttributeGroup and the MetaClass:



After refresh, the general tab becomes:



Defining a new page displaying a non-standard element

You can define a new page in the Properties window of the *Firm* MetaClass, enabling definition of the list of its *Departments*.

This list is obviously not included in the standard attributes of the *Firm* MetaClass. It is not a case of moving standard elements, but of inserting a new element. The MetaAttributeGroup concept is therefore not used in this extension.

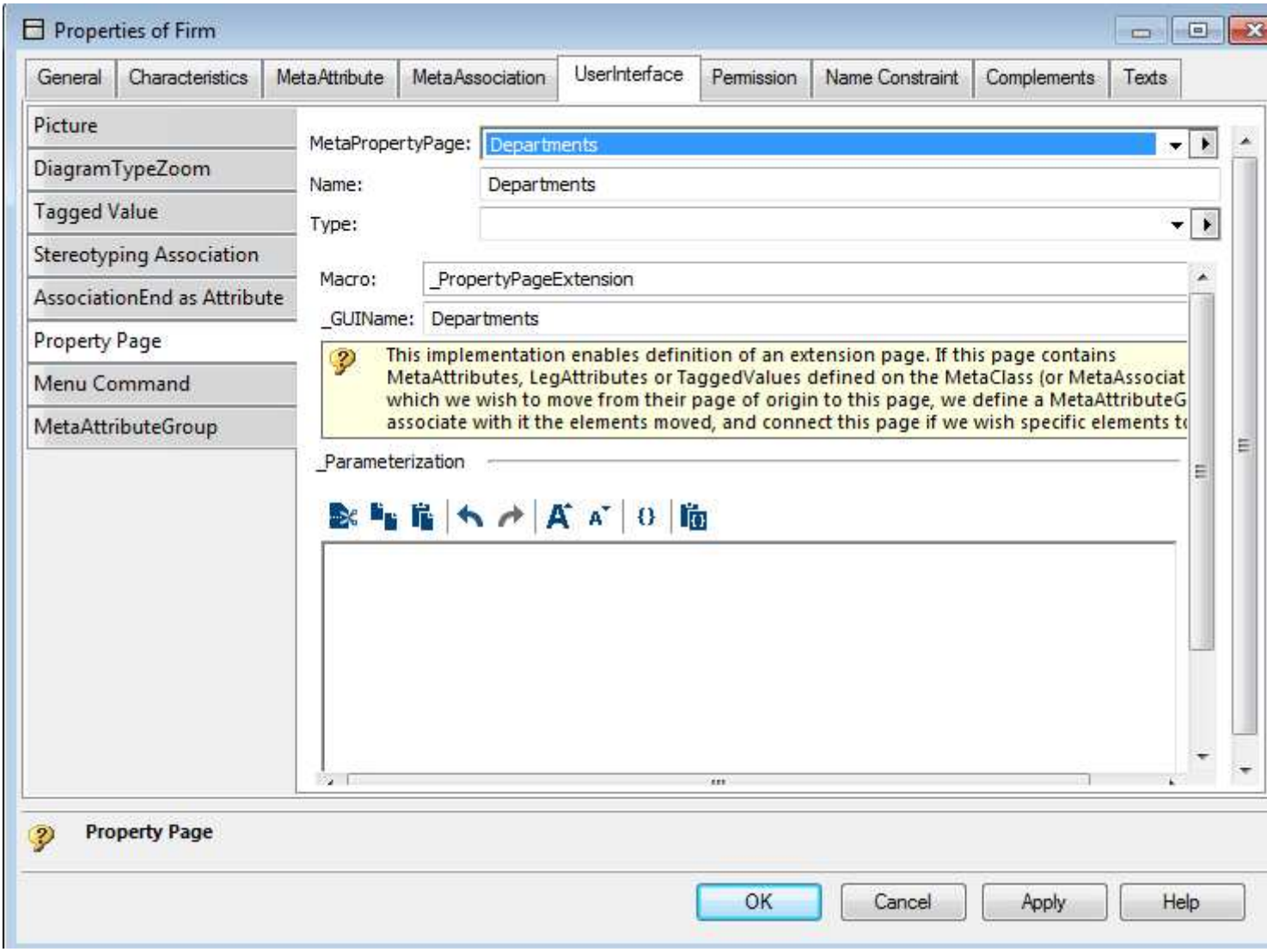
To do this, you will use a MetaPropertyPage implemented by the " _PropertyPageExtension" Macro. This macro will build a Properties page, consulting the content of the "parameterization" text of the MetaPropertyPage.

This configuration text is a setting text, that is a text which specifies a list of key/value pairs, grouped in sections:

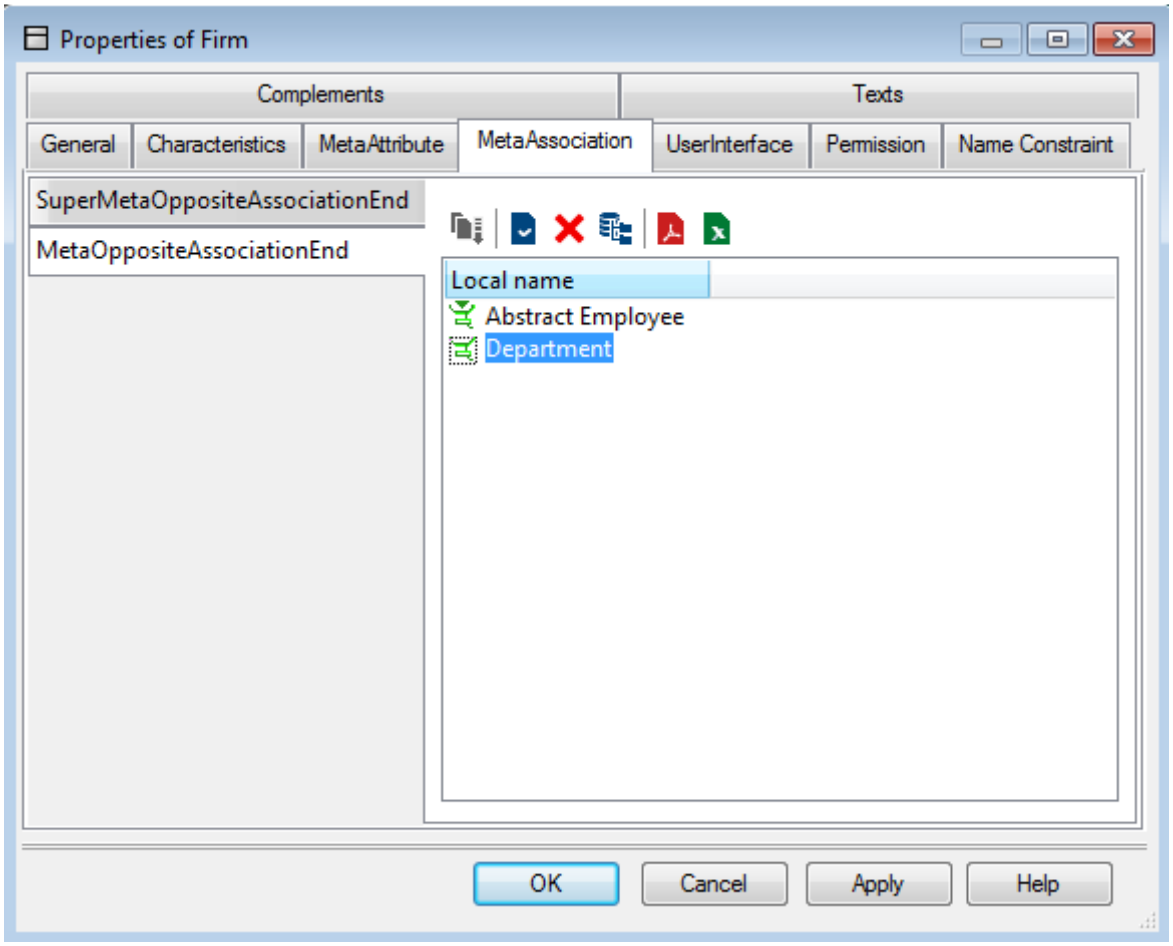
```
[Section]
Key1 = Value1
Key2 = Value2
```

This text is auto-documented and entry help is available. The list of elements you want to include in the page is specified in the [Template] section, and the page order number can be defined in the [Page] section.
The customization envisaged consists of displaying a ListView browsing the *(Firm/Department)* MetaAssociation view from the *Firm*, therefore by the "*Department*" MetaAssociationEnd.

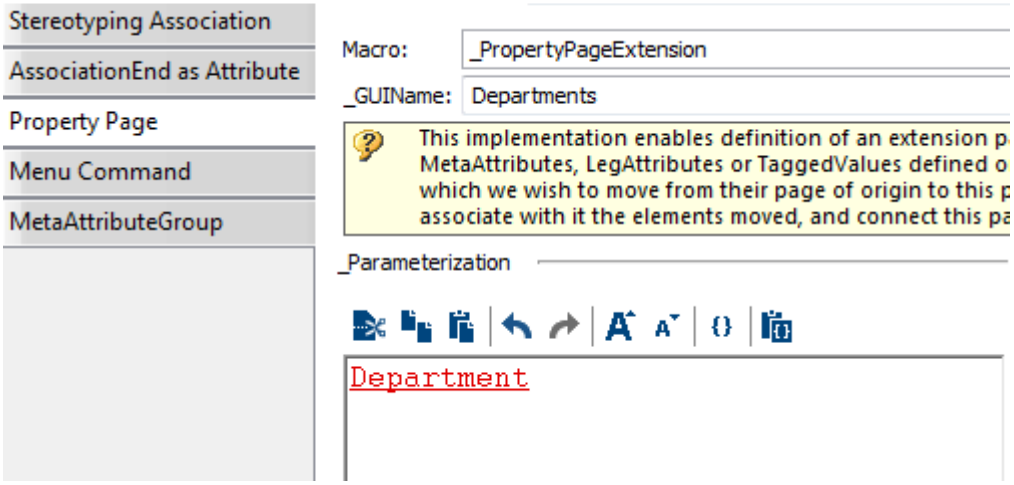
1. Open the Properties window of the **Firm** MetaClass.
2. Select the **User Interface** tab, then the **Property Page** subtab.
3. To create a new Properties page, in the **MetaPropertyPage** field, click the arrow and select **New**.
4. Name the new page « Departments » and click **OK**.
5. In the **MetaPropertyPage** drop-down menu, select the new page « Departments ».
   It is now possible to define the _GuiName of this page, that is the name displayed in the tab: for example "Departments", as well as the Macro which implements this page.
6. In the **_GUIName** field, enter « Departments ».
7. In the **Macro** drop-down menu, select PropertyPage.Kind, then the *_PropertyPageExtension* macro.
8. Click **Apply**.
   The page will then allow you to define configuration of your new page.



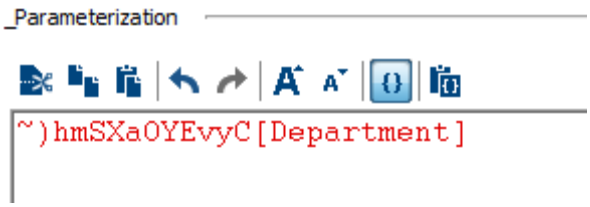
9. To configure your page, you simply need to know the identifier of the MetaAssociationEnd which must be browsed by the listview. To do this, from the **MetaAssociation** tab, **MetaOppositeAssociationEnd** subtab, copy the "Department" MetaAssociationEnd.



10. Paste the result in the **_Parameterization** frame.



The pasted text is a "field", in which the absolute identifier of the pasted area is carried. Button {} enables display of this identifier.

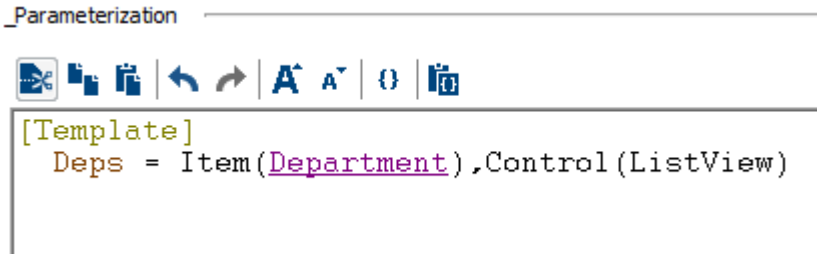


This field will serve as parameter for the 'ListView' type element which you will define in the page [Template].

11. In the **_Parameterization** frame, paste:

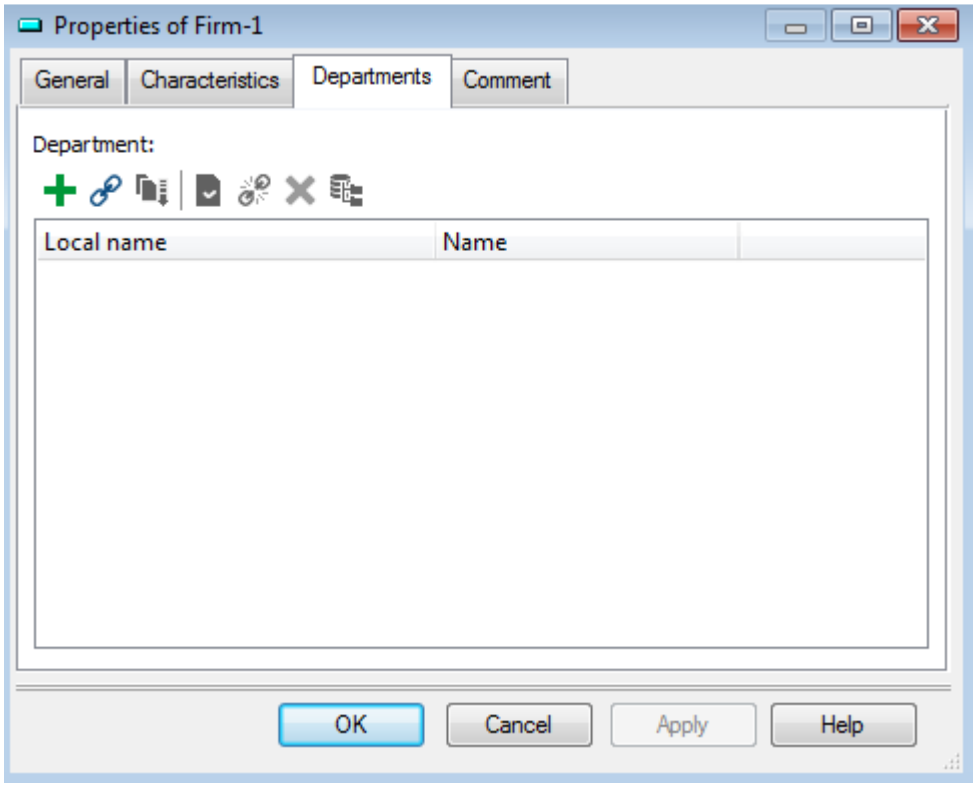
```
[Template]
  Deps = Item(~)hmSXaOYEvyC[Department]),Control(ListView)
```

Text color indicates that your definition complies with syntax.



12. Click **Apply**.

After refresh, the list appears in the *Firm* Properties window.



Adding a non-standard element in the Characteristics page

You can include this *Departments* list not in a specific page, but in the object Characteristics page. Ensure that this addition does not interfere with the list of attributes naturally included here.

The standard **Characteristics** page is not an extension page, and not therefore implemented by the same macro; the macro used is _PropertyPageStandard. In addition to the content of the "Characteristics" implicit MetaAttributeGroup, this page displays:

- The name or local name as page header, depending on whether or not the MetaClass has specific name implementation.
- Object owner if applicable.
- MetaAssociation attributes when the object is seen from another object, if applicable.
- Extensions defined on the MetaClass if these have not been stored in specific MetaAttributeGroups, if applicable.

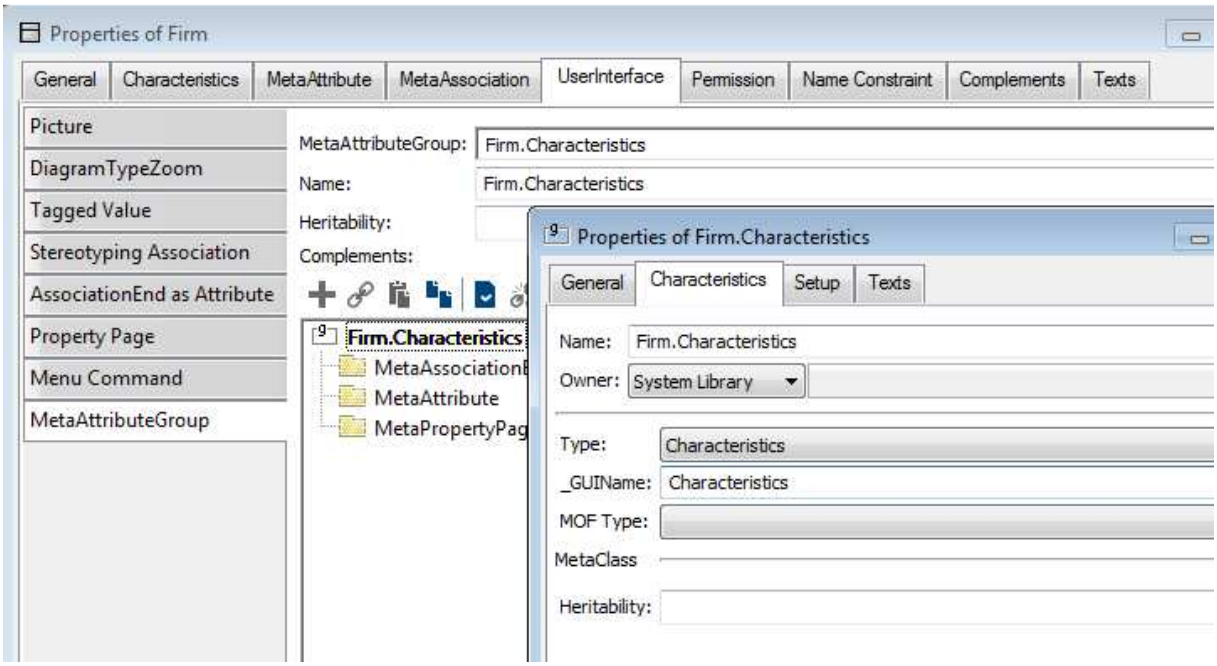
To overload the **Characteristics** page, you must first overload the implicit MetaAttributeGroup with an explicit MetaAttributeGroup: To do this, simply create a MetaAttributeGroup from the MetaClass, and specify "Characteristics" type.

When this MetaAttributeGroup has been created, two cases are possible:

- either you do not want to modify the list of attributes implicitly present in this MetaAttributeGroup: in this case, no element should be defined. This is our example case.
- or you want to explicitly define the list of attributes visible in the **Characteristics** page, and in this case you should populate the MetaAttributeGroup with these attributes.

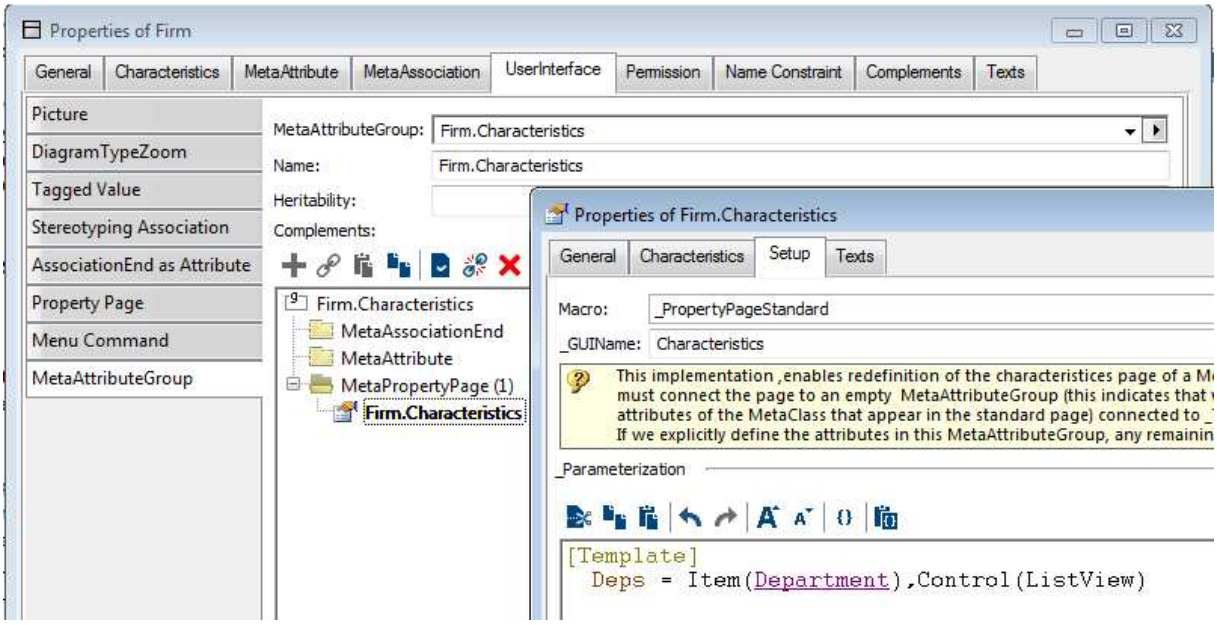
Note: In this case, standard processing can result in attributes not assigned to a MetaAttributeGroup, and which will not therefore be displayed. These attributes are therefore assigned to the "Extension" standard MetaAttributeGroup. You can detect this phenomenon in noting appearance of an 'extension' tab.

Create this MetaAttributeGroup, name it *Firm.Characteristics*, and assign it Characteristics type.

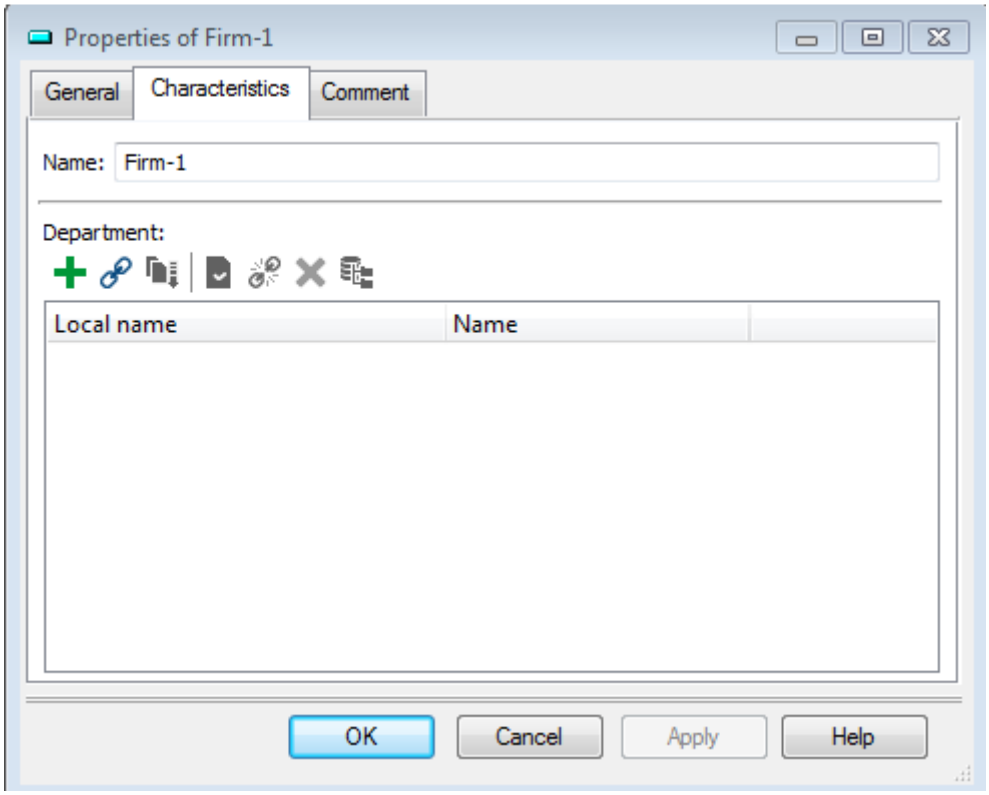


To achieve your objective, you must customize display of the Properties page derived from the MetaAttributeGroup.

1. To create a MetaProprertyPage dedicated to the MetaAttributeGroup, in the **MetaAttributeGoup** configuration page, click **Create**.
   This MetaPropertyPage will replace the implicit page associated with the MetaAttributeGoup. In addition, if specified:
   - the page Type will be taken into account rather than the MetaAttributeGroup type, allowing you to display the page in another tab.
   - the GuiName of the page will be used as page title rather than the GuiName of the MetaAttributeGroup.
2. As previously indicated, implement this page with the _PropertyPageStandard macro to functionally replace the **Characteristics** page.
3. Add the Departments list in this page, using the same [Template] paragraph as in the previous exercise.



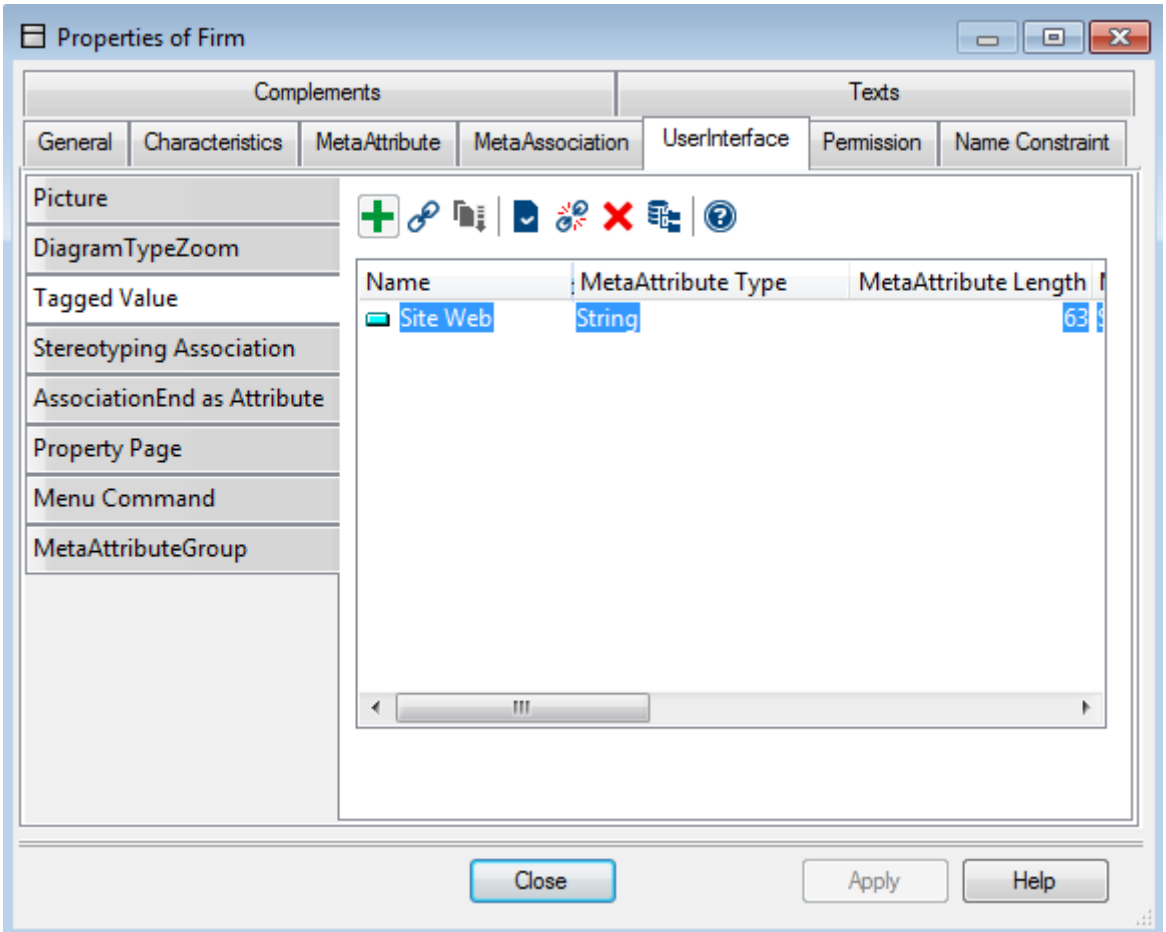
Your work is completed and the **Characteristics** page of a firm is now (remember **Refresh Context**):



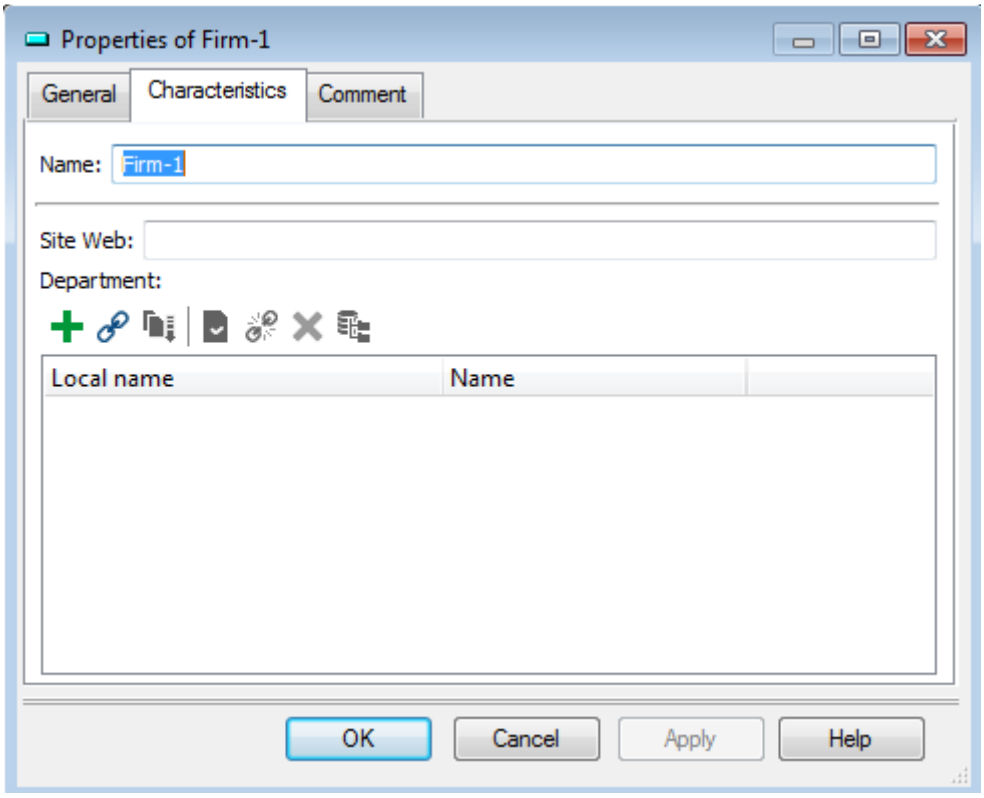
Note that the **Name** field appears in this page, though you did not define it in the [Template] paragraph.

This is because this MetaPropertyPage was associated with a MetaAttributeGoup; this means that all the attributes it contains (in your case the default content of the "Characteristics" group) are automatically inserted in this page, without it being necessary to specify them in the page [Template]. You have therefore correctly defined an extension to the **Characteristics** page, without prejudging its prior content. You will be better able to appreciate the advantage of such a device later in this exercise.

To do this, take into account the fact that your client want to be able to define a *Site Web'* property for each of his *Firms*. This extension can take the form of a TaggedValue which you shall create for this MetaClass:



After refresh, this extension appears in the **Characteristics** page, without any other specification being required.


Properties of Firm-1
General
Characteristics
Comment
Name:
Firm-1
Site Web:
Department:
Local name
Name
OK
Cancel
Apply
Help

Adding non-standard elements in a page

The previous exercise showed you that you could include a list (ListView) in a page by browsing a MetaAssociationEnd defined on a MetaClass of the object.
The main elements you can feature in a Properties page are listed below.

ListView: view as list

A ListView enables display of a list obtained by browsing a MetaAssociationEnd or a Query from an object in the Properties page. Queries that can be used in this context are either queries without parameters, or queries that have an occurrence as their only parameter.
You can:

- configure the list of columns displayed (which are generally attributes of browsed objects), as well as the list toolbar.
- create a list displaying several distinct collections, by selecting the active collection from the toolbar.

TreeView: view as tree

A treeView enables display of a tree, built from a list of MetaAssociationEnds and Selectors, the root of which is the object in the Properties page. This list can be deduced from an _Operator.
You can:

- define if a path can be of several levels or a single level.
- display folders or not.
- configure the tree toolbar.

Example: the following element displays the default navigation tree of an object. Here, *Navigate* is the _operator used to define the path.

```
Tree=Item(~laCnbKSWt000[Navigate]),Control(TreeView),Param(IsDeep)
```

Viewer: HTML formatter display

You can include an HTML formatter in a Properties page. This formatter will be generated from the object in the Properties page. In this way you can display any content in a Properties page, on condition that it is not intended for update.
You can trigger an action in this HTML document when you execute 'Apply' on the Properties sheet.

HelpComment: help comment display

You can include help text in a Properties page. This text can be text from system data or from modeled data.

SubPage: page in page display

You can display a Properties page in another page, for example to reuse an implementation which can be defined in a generic page.

Displaying elements from another object

So far, you have limited yourselves to display of data obtained directly from the object of the Properties sheet, the display of other objects only being possible by means of:

- TreeView, which allows display of only the name
- ListView, which limits display to columns of attributes, excluding for example the representation of comments.

You can however display in a page elements from other objects, on condition that there is a MetaAssociationEnd or a query allowing you to reach these objects from the main object.

Two possibilities are available:

- The MetaAssociationEnd (or Query) has maximum cardinality 1.
  In this case only one object is visible, and it is possible to include attributes of this object in the same way as for attributes of the main object.
- The MetaAssociationEnd (or Query) accesses a collection of objects. In this case we must have available in the page an element enabling selection of an object from this collection. This object having been selected, elements relating to the attributes of this object can be supplied.

In both cases, you must indicate in page configuration that the element does not relate to the main object, but to another object. To do this, use the Map concept, which will include access to this other object.

Direct display of an associated object

You will first configure a page on the *Department* MetaClass in which you will include the name and comment of the *Firm* of the department.

The *Firm* being unique for a department, you will not require a selector element. To do this, you must:

- declare a Map specifying the browsed element, here the *Firm* MetaAssociationEnd

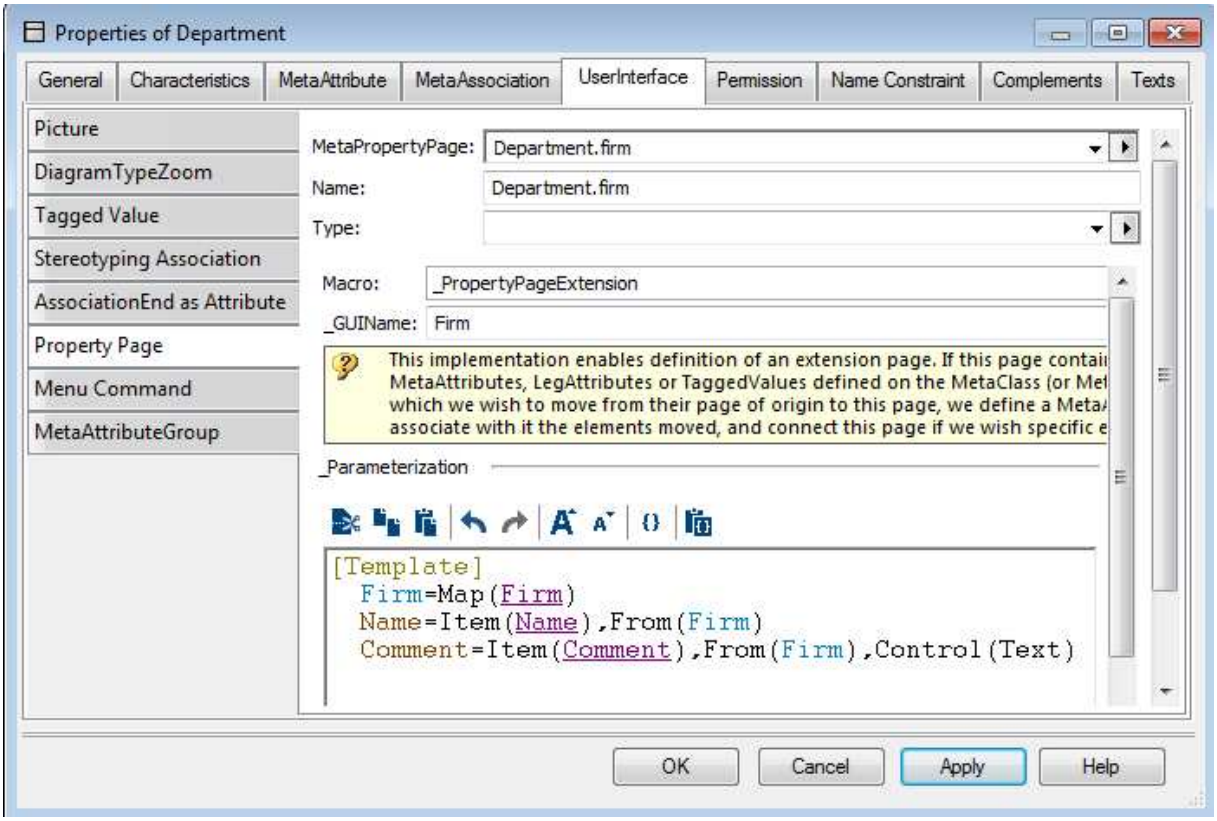
```
Firm=Map(~(hmSXaOYEryC[Firm])
```

- declare the two elements.

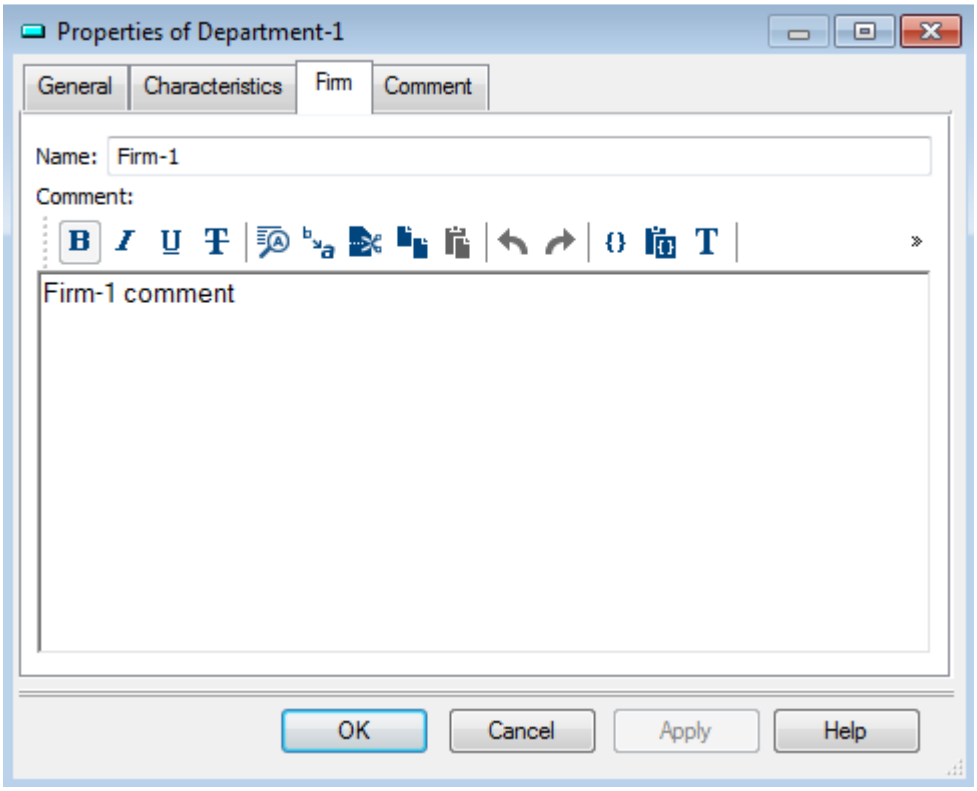
  The keyword From(*map*) indicates which object will use the element.

  ```
  Name=Item(~2100000000900[Nom]),From(Firm)
  Comment=Item(~f10000000b20[Comment]),From(Firm),Control(Text)
  ```

  When the page has been created, you obtain:



When you display the Properties sheet of a *Department*, you obtain:



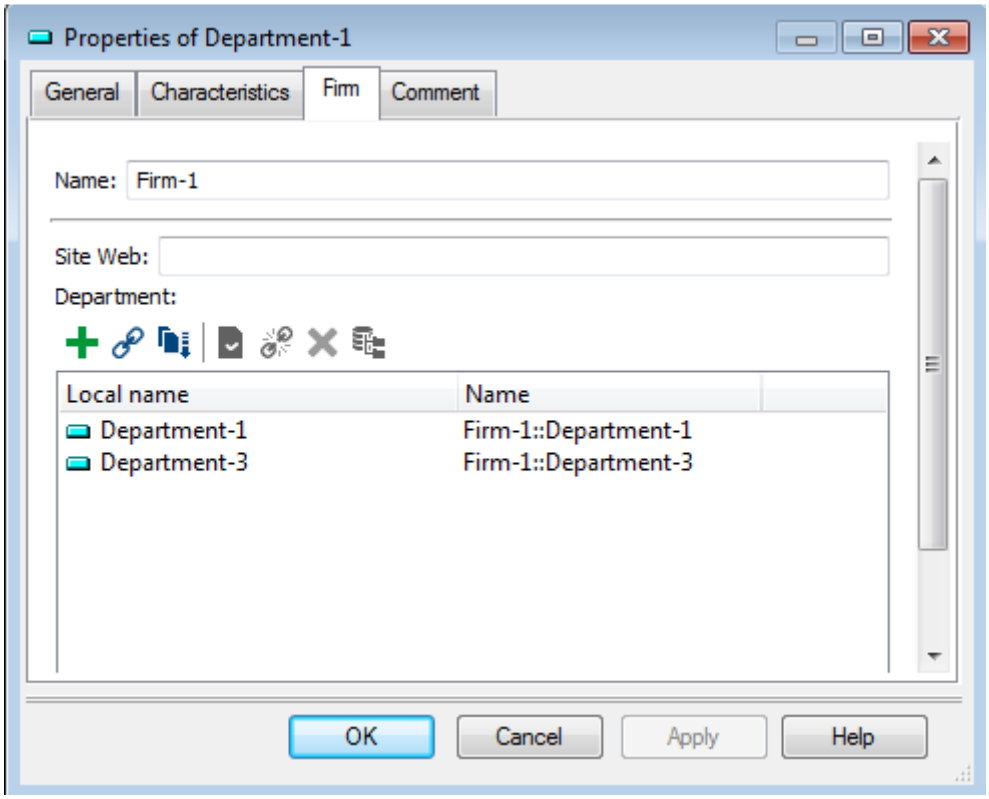
Instead of displaying elements of the *Firm*, you can directly display a Properties page of *Firm*, for example the Characteristics page.
To do this, you must use an element of SubPage type. The Template becomes:

```
[Template]
Firm=Map(~(hmSXaOYEryC[Firm])
Page=Item(~ZPhytJ)cE5k1[Firm.Characteristics]),From(Firm),Control(SubPage)
```

Warning:
In the above example, Firm.Characteristics has an IDabs different from that which you have created. To copy the correct IDAbs:

1. Open the Properties page of the **Firm** MetaClass.
2. Select the **User Interface** tab, then the **MetaAttributeGroup** subtab.
3. Under the **MetaPropertyPage** folder, copy Firm.Characteristics and paste it in the above Template.

The "Firm" tab now displays the **Characteristics** page of the *Firm*.



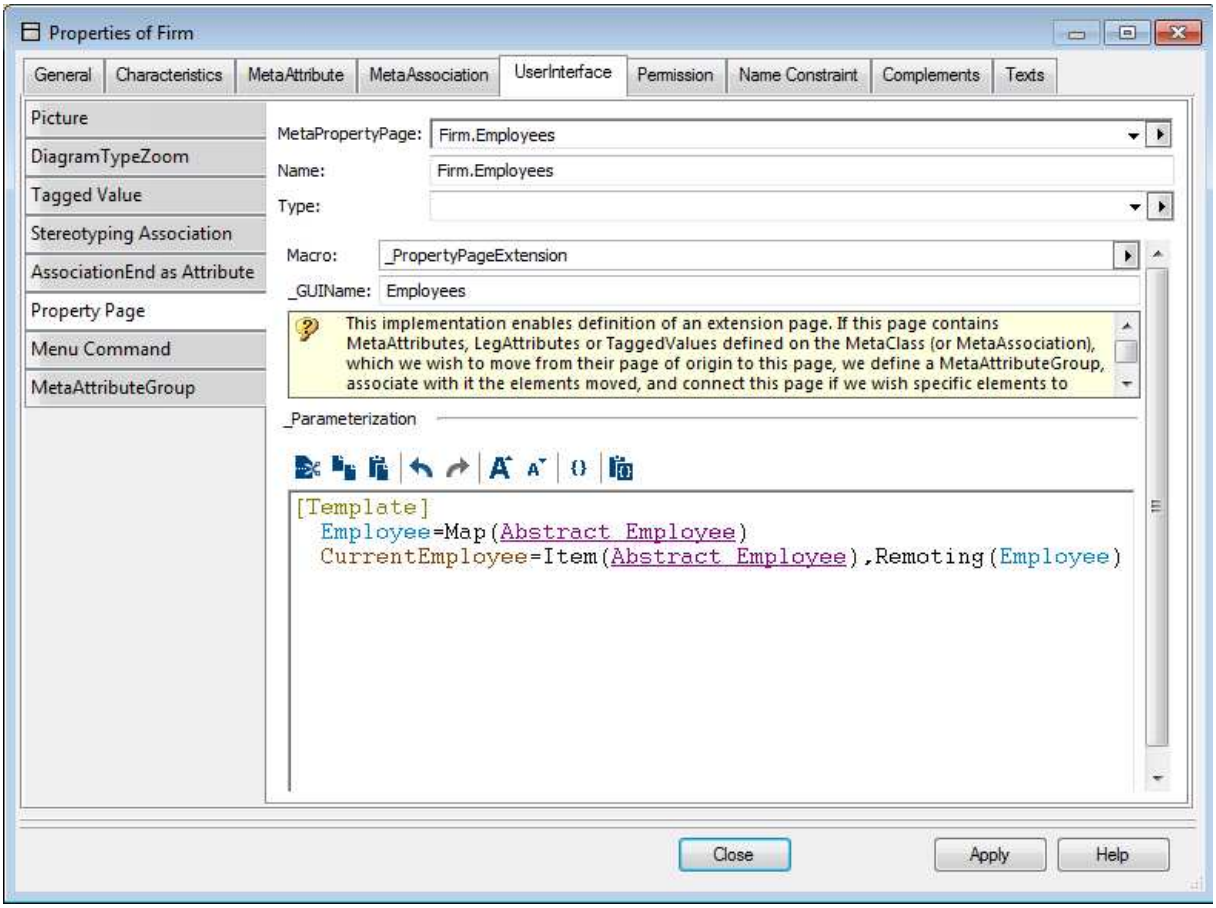
Displaying an object from a selection

We will now consider the case of a multiple collection; in this case you must be able to select an element from the list. This selection will be made by means of an element.
In this context, the identifier of the collection used to define the Map is not used: the Map is 'passive' and only knows its current object through the selection. However, by convention we associate with it, if possible, the browsed collection.
A dedicated element enables achievement of this objective: it takes the form of a drop-down list. To declare it, associate it with the browsed collection and the map concerned.

To illustrate this configuration, you will create an **Employee** page on the ***Firm*** MetaClass, enabling display of the Characteristics page of an Employee:

1. You will configure the element which enables selection of the Employee, by means of the Abstract Employee MetaAssociationEnd. In the configuration considered, declare a Map named Employee and its controller:
   `[Template]`

```
Employee=Map(~f7mZ(UQYEHTI[Abstract Employee])
CurrentEmployee=Item(~f7mZ(UQYEHTI[Abstract Employee]),Remoting(Employee)
```



- The keyword 'Remoting' defines the dependency relationship between the 'Employee' Map and the "CurrentEmployee" element.
- The browsed collection is defined in the CurrentEmployee item; we define the Map by means of the same collection, but this parameter is not used in the Map.
  Thus defined, CurrentEmployee is displayed in the following way:

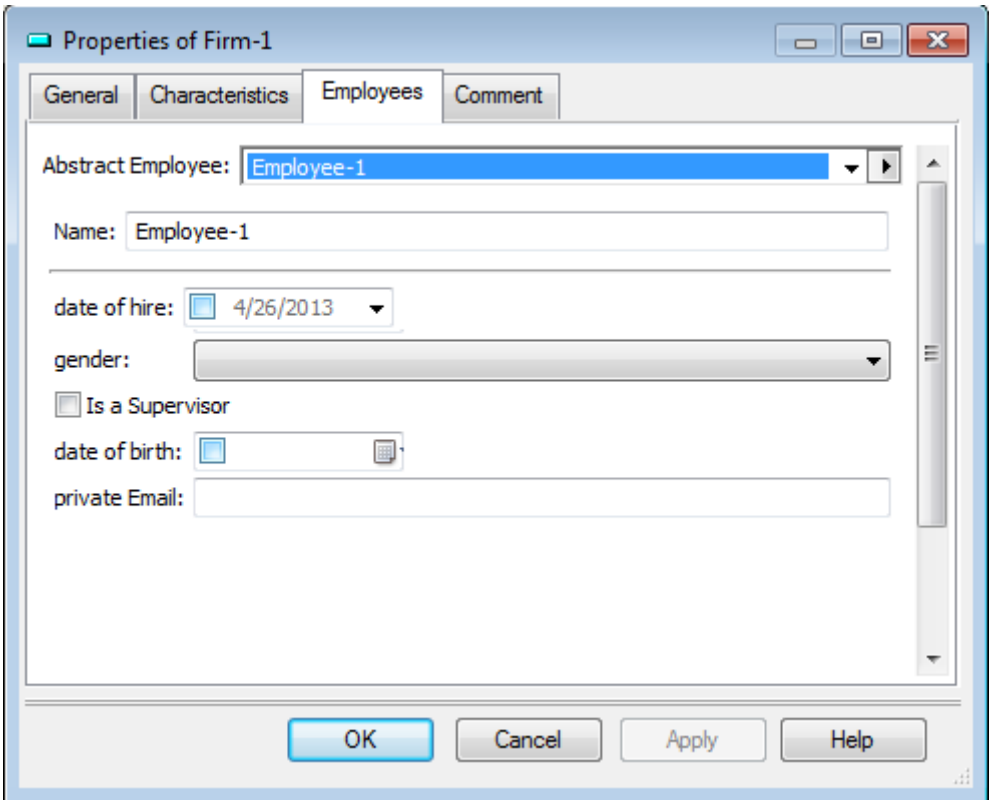
2. You will define the subpage which describes the characteristic page.
   There is an additional difficulty here, since you cannot directly cite the absolute identifier of the page to be displayed. In fact:
   - the Characteristics page does not physically exist

- above all, the association used being generic, the objects listed can be either *Employee* or *Manager*, and the page to be displayed therefore differs according to the object selected

You will use the following specific configuration for the SubPage:

```
EmployeePage=Item(~UZlkzjxjtO00[Characteristics]),From(Employee),Control(SubPage),Param(Computed,Type)
```

- The keyword 'Computed' indicates that the page must be calculated, and therefore that the identifier of the *Item*() is not a MetaPropertyPage.
- The keyword 'Type' indicates the calculation mode. In this case, the identifier of *Item*() is the type to be used: here we request page display of 'Characteristics' type of the selected object.



Modifying customized page appearance

In previous chapters, we covered the problem of populating Properties pages and specification of their content, leaving the responsibility of organization and appearance of elements to be displayed to page implementation. It is possible that the resulting appearance is not satisfactory in terms of:

- element title
- grouping
- order in which elements appear
- size and position
- displayed control not suitable

These parameters can be redefined to achieve the required appearance.

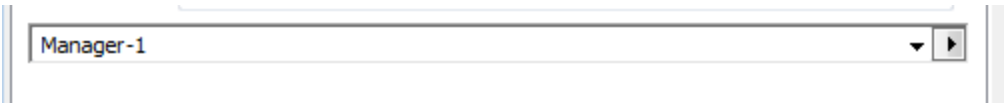
Modifying element title

For a given element, you can define the presence, position and name of its title, by means of keywords *Title* and *Name*.

- The keyword *Title* enables definition of the presence or position of an element title using parameters Up, Left or No.
- The keyword *Name* enables definition of the title name. You can directly cite the name to be used, or a system repository occurrence (MetaAttribute, CodeTemplate for example): in this case you can manage multilingualism.

You will apply these configurations to the "Current Employee" element of your previous example, of which the name is not totally satisfactory.

- Deletion of title:
  `CurrentEmployee=Item(~f7mZ(UQYEHTI[Abstract Employee]),Remoting(Employee),`**`Title(No)`**



- Renaming 'static':

```
CurrentEmployee=Item(~f7mZ(UQYEHTI[Abstract
Employee]),Remoting(Employee),Name(Employee)
```

- Renaming by occurrence:

```
CurrentEmployee=Item(~f7mZ(UQYEHTI[Abstract
Employee]),Remoting(Employee),Name(~ThmSy)NYEzzB[Employee])
```

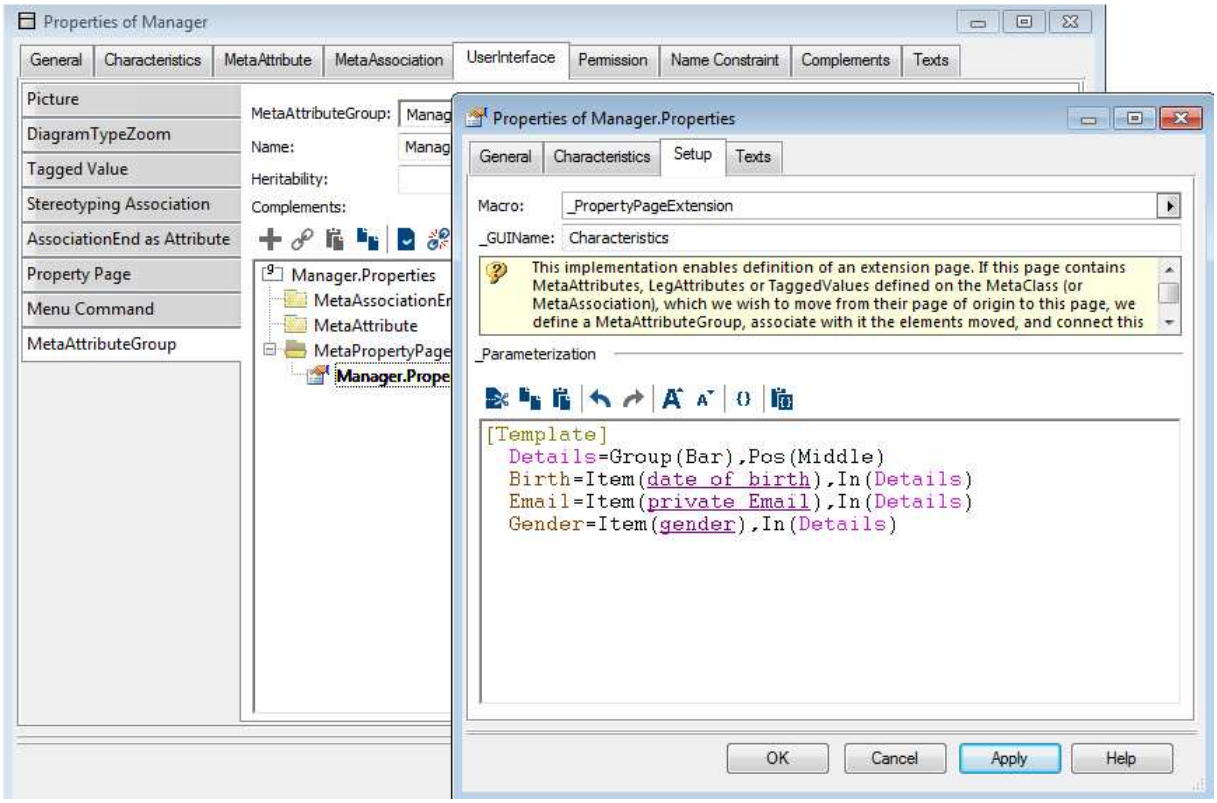
Here the title 'Employee' is calculated from the name of the `~ThmSy)NYEzzB[Employee]` - system object, in this case the *Employee* MetaClass.

- Title positioned at top (by default for this type of control, the title is positioned at left: Title(Left)

```
CurrentEmployee=Item(~f7mZ(UQYEHTI[Abstract
Employee]),Remoting(Employee),Name(~ThmSy)NYEzzB[Employee]),Title(Up)
```

Regrouping elements

To group elements of a Properties page. use the Group concept.
The following exercise will allow you to use this concept in the Characteristics page of a *Manager*, in which you will group personal elements (date of birth, gender, email).
To do this, you will overload the Characteristics page by means of a Characteristics MetaAttributeGoup associated with a MetaPropertyPage.
In the Template paragraph of this page, you will cite the elements you wish to group.

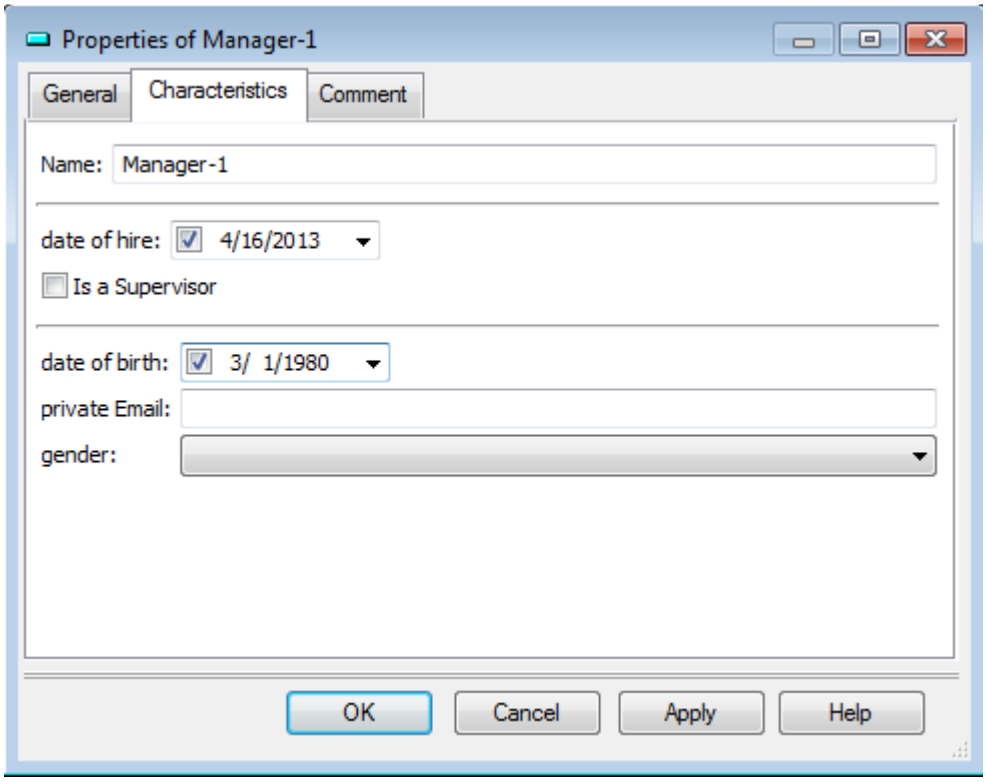



```
[Template]
  Details=Group(Bar),Pos(Middle)
  Birth=Item(~XFIiLwhZE5PJ[date of birth]),In(Details)
  Email=Item(~8DIipwhZEfTJ[private Email]),In(Details)
  Gender=Item(~6emSkyNYETjB[gender]),In(Details)
```

This template defines a group represented by a bar (‘Bar’) and positioned in the "middle" of the page.

Note that it is possible in any case to include in the [template] of a page the attributes it contains; in this case the element included in the template simply overloads standard behavior of the attribute.

In the resulting page, elements concerned are grouped and separated from other elements by a bar. They appear at bottom of the page since no other group has been defined (therefore ‘middle’ is the only available position for ‘bottom’).



Replace Group(Bar) by Group(Hidden) to regroup the elements, but without separator.

Replace Group(Bar) by Group(Frame) to surround group elements by a frame.

To name the group similarly as for an element, add Name(<Group Name>):

```
Details=Group(Bar),Pos(Middle),Name(Details)
```

Note that a certain number of implicit groups are available, in particular the "Name" group you can see in the Characteristics page, and that you can insert elements in these groups.

Modifying element order

It is possible to modify the order of elements within the same group.

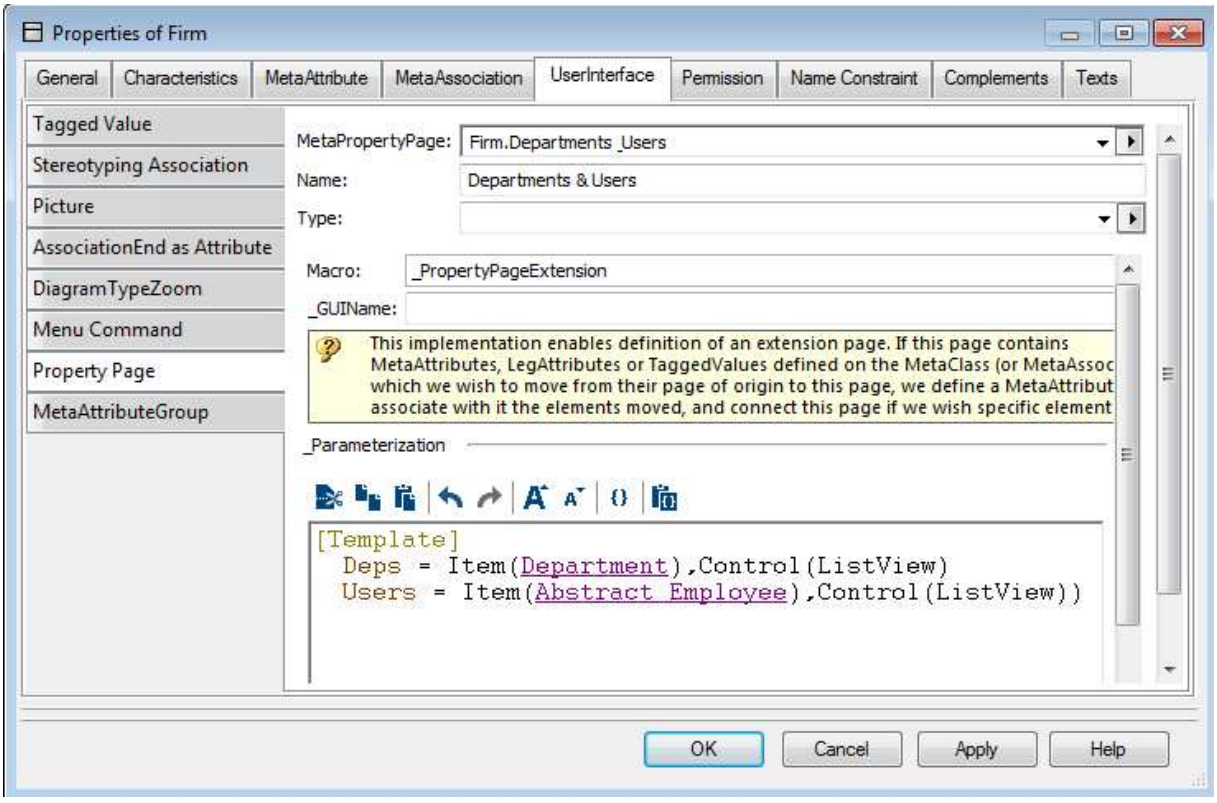
When elements are explicitly defined in a group, the order of their definition in the Template corresponds to display order (from MEGA 2009 SP4). To redefine order of elements, it may only be sufficient to define a Group.

For implicit elements, we use the order of attributes in the link with the MetaClass, that is (MetaClass/MetaAttribute), (MetaClass/TaggedValue), (MetaClass/_LegAttribute) according to the attribute type.
Complete classification of attributes is calculated by consolidating values of the Order attribute on each of these links. For example, a TaggedValue connected to a MetaClass by order number 25 will appear between the MetaAttributes connected to this same MetaClass (or Abstract class if the MetaAttribute is inherited) which have numbers 20 and 30.
To sort elements, it may only be sufficient to reorder them relating to their respective MetaClasses; this sort order will then be valid in all entries.

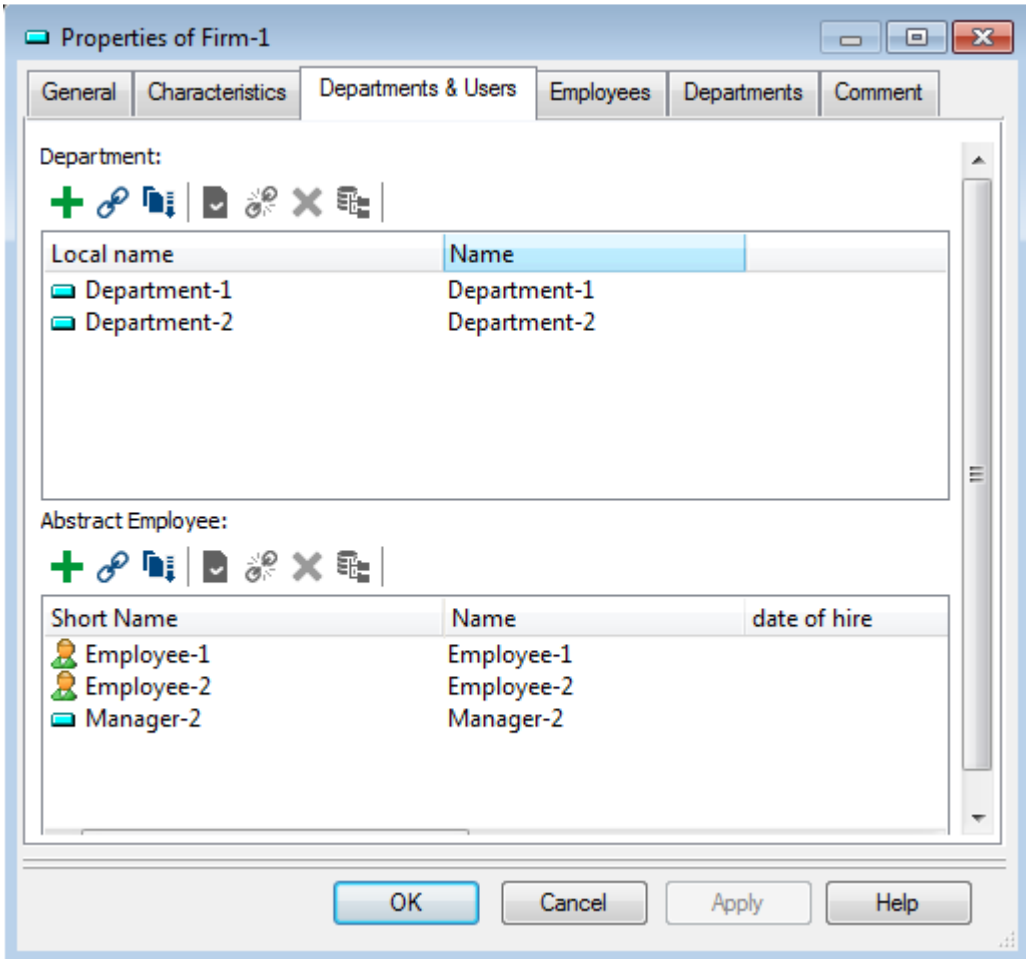
In no case is sort according to MetaAttributeGroup used.

Modifying element size

You can redefine size of an element; this is particularly useful for elements of ListView or TreeView type. For information, all 'multiline' elements (ListView, TreeView, Text, Viewer, HelpComment, SubPage…) use all space remaining in the page when they appear last. Otherwise they have a default height which you can modify.
To illustrate this possibility, you will define on Firm a Properties page displaying two lists.



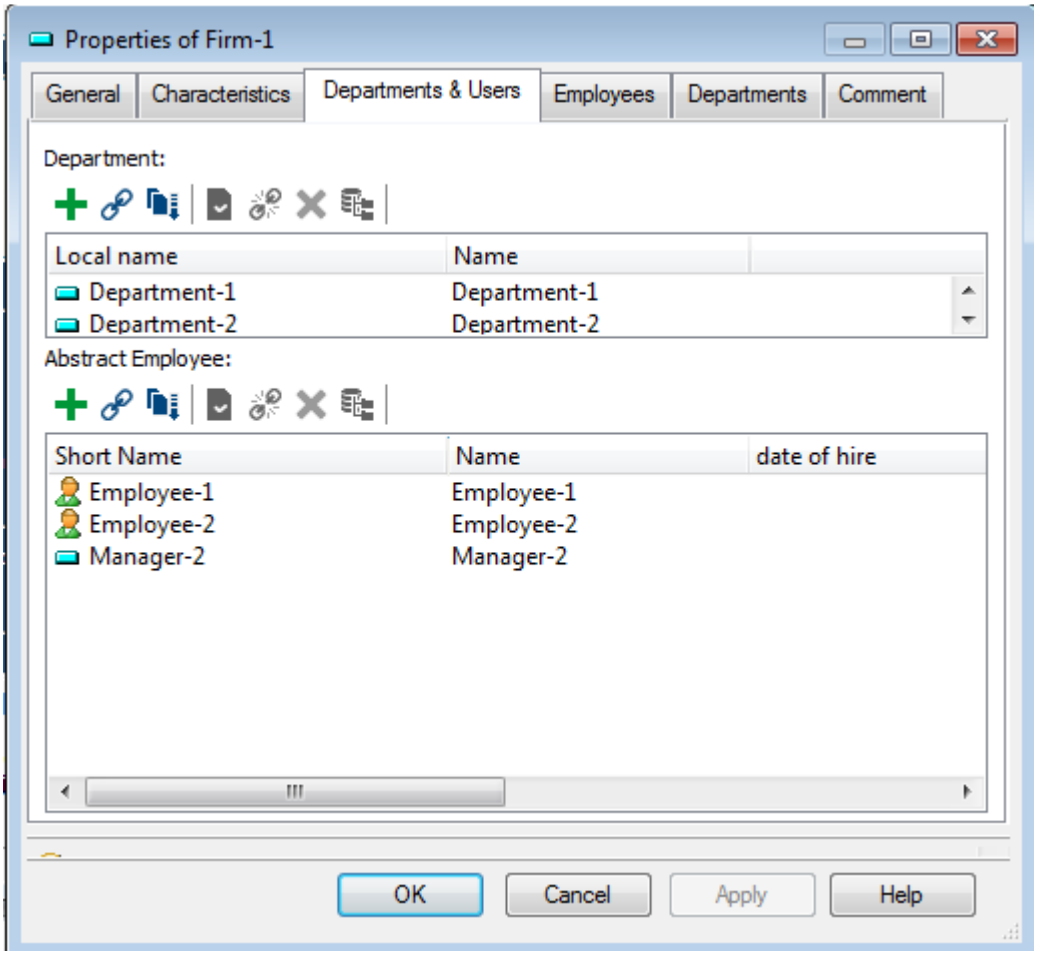
A default height is assigned to the "Deps" list, while the "Users" list, positioned at bottom of the page, will occupy the remaining space:



In redefining a size for the Department area (keyword**Size()**)

```
Deps=Item(~)hmSXaOYEvyC[Department]),Control(ListView),Size(2000,50)
```

You obtain the following result:



Width 2000 is deliberately exaggerated to indicate that the ListView should occupy all available horizontal space; height 50 includes the complete listview, that is its title, toolbar, header and the list itself.

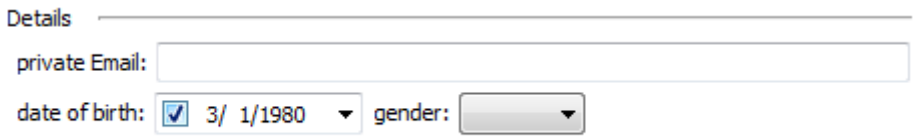
Sizes are expressed in "dialog units", a unit in which a medium character of the font used to draw the page measures 5x8. 50 therefore represents approximately 5 lines (a line can equal 10 dialog units including two separation units)

Modifying element positioning and resizing

For even more detailed configurations, you can specify the exact size and position of elements in the same group. Position is expressed in dialog units and is relative to the Group. Elements without coordinates are added at bottom of the group.

Returning to the previous example, you can position elements so that 'gender' is positioned alongside 'date of birth'. The keyword **At()** enables definition of element position.

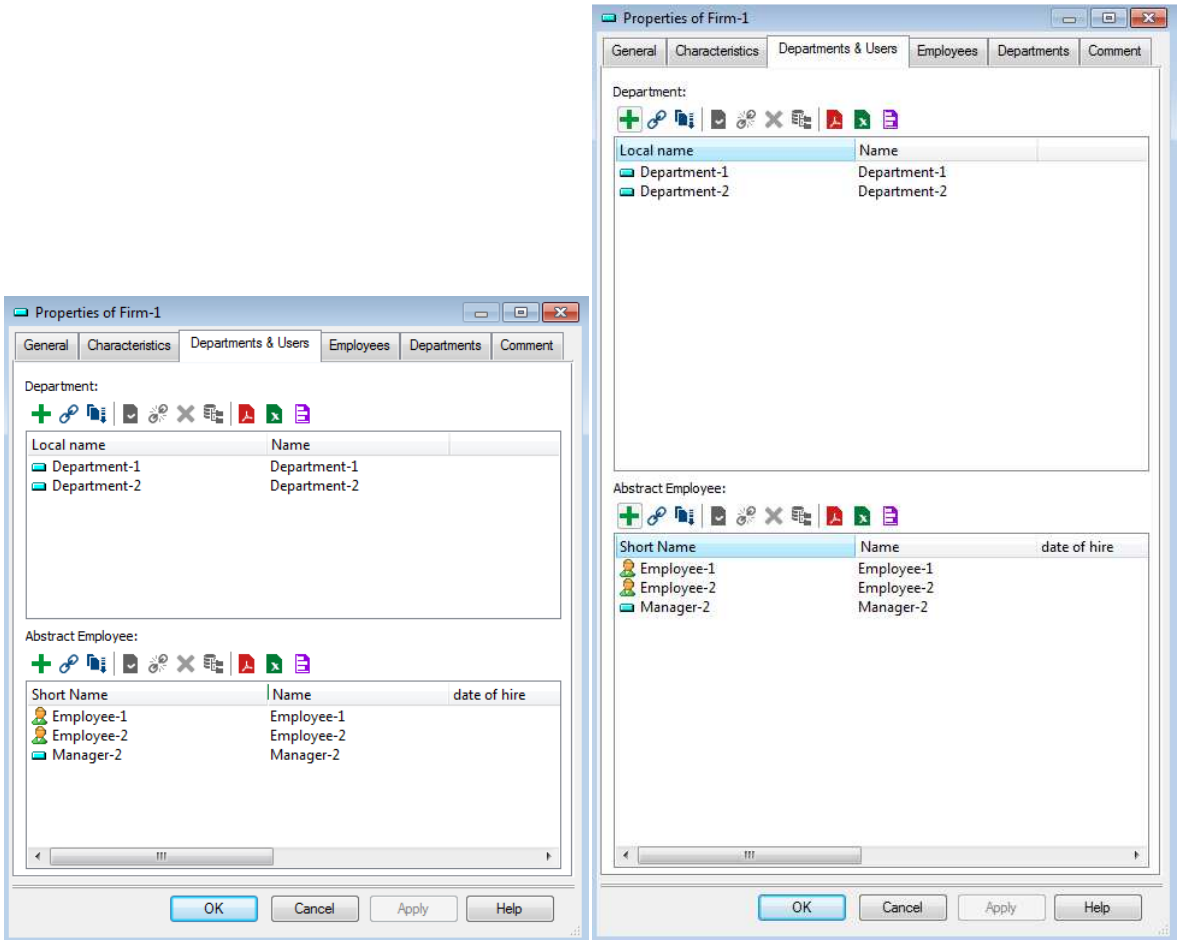
```
[Template]
  Details=Group(Bar),Pos(Middle),Name(Details)
  Email=Item(~8DIipwhZEfTJ[private Email]),In(Details),At(1,10)
  Birth=Item(~XFIiLwhZE5PJ[date of birth]),In(Details),At(1,25),Size(115,13)
  Gender=Item(~6emSkyNYETjB[gender]),In(Details),At(120,25),Size(70,13)
```



Finally you can define resizing laws for a Properties page when its size is modified.
To do this, you must define basic dimensions for the page to define initial docking. It is on this basic dimension that areas must be positioned in the [Template] paragraph.
To apply this principle, you will customize the Department and User page so that both lists can be extended, and not just the second.

```
[Template]
Deps=Item(~)hmSXaOYEvyC[Department]),Control(ListView),At(1,1),Size(2000,100),VClip
(TopToCenter)
  Users=Item(~f7mZ(UQYEHTI[Abstract
Employee]),Control(ListView),At(1,115),Size(2000,100),VClip(CenterToBottom)
[Page]
MinHeight = 225
```

The keyword **VClip()** enables definition of movement of the element, attaching it at (top), (bottom), or (center) of the page. An element attached at two points (for example TopToCenter) will therefore be distorted. In our example, the two lists will be enlarged each to half of page enlargement.



Horizontal clipping uses keyword **HClip(),** and requires definition of page width ([Page] MinWidth)

Deactivating an element

The following will enable modification of the type, or of the ability to update a control displayed for an attribute.

To do this, you will take the first version of the *Department.Firm* MetaPropertyPage on the *Department* MetaClass. In this example, the page Template is:

```
[Template]
  Firm=Map(~(hmSXaOYEryC[Firm])
  Name=Item(~2100000000900[Nom]),From(Firm)
  Comment=Item(~f10000000b20[Comment]),From(Firm),Control(Text)
```

In this page, you do not wish the name of the *Firm* to be modifiable.
To do this, deactivate entry of the element:

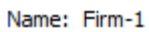
```
Name=Item(~2100000000900[Nom]),From(Firm),DisabledOn(Always)
```

➔ The effect of this is to gray the area and prohibit its modification:

You can also replace the 'Edit' control used by default for the name by a 'Static' control:

```
Name=Item(~2100000000900[Nom]),From(Firm),Control(Static)
```

➔ The appearance of the area changes:



Forcing an entry

To make it mandatory to enter *gender* of a *Manager* in the Manager.Properties customized Properties page define previously, use the keyword Mandatory.

```
Gender=Item(~6emSkyNYETjB[gender]),In(Details),At(120,25),Size(70,13),Mandatory(Yes)
```

With this configuration, you cannot assign value (None) to *gender*.

Hiding a standard element

In this same page, you now want to hide gender. This attribute being derived from MetaAttributeGroup, we must hide it explicitly. To do this, we use the keyword Visibility.

```
Gender=Item(~6emSkyNYETjB[gender]),In(Details),At(120,25),Size(70,13),Visibility(Hidden)
```

With:

- Visibility(Hidden): the attribute is never visible.
- Visibility(Admin): the attribute is hidden to users who do not have 'Expert' metamodel access.
- Visibility(Always): the attribute normally restricted to experts or advanced users can be made visible to all users.

Modifying page behavior

Hiding a page

To condition appearance of a MetaPropertyPage, you can define a condition in the [Filter] paragraph of page configuration.

```
[Filter]
Condition = <condition>
```

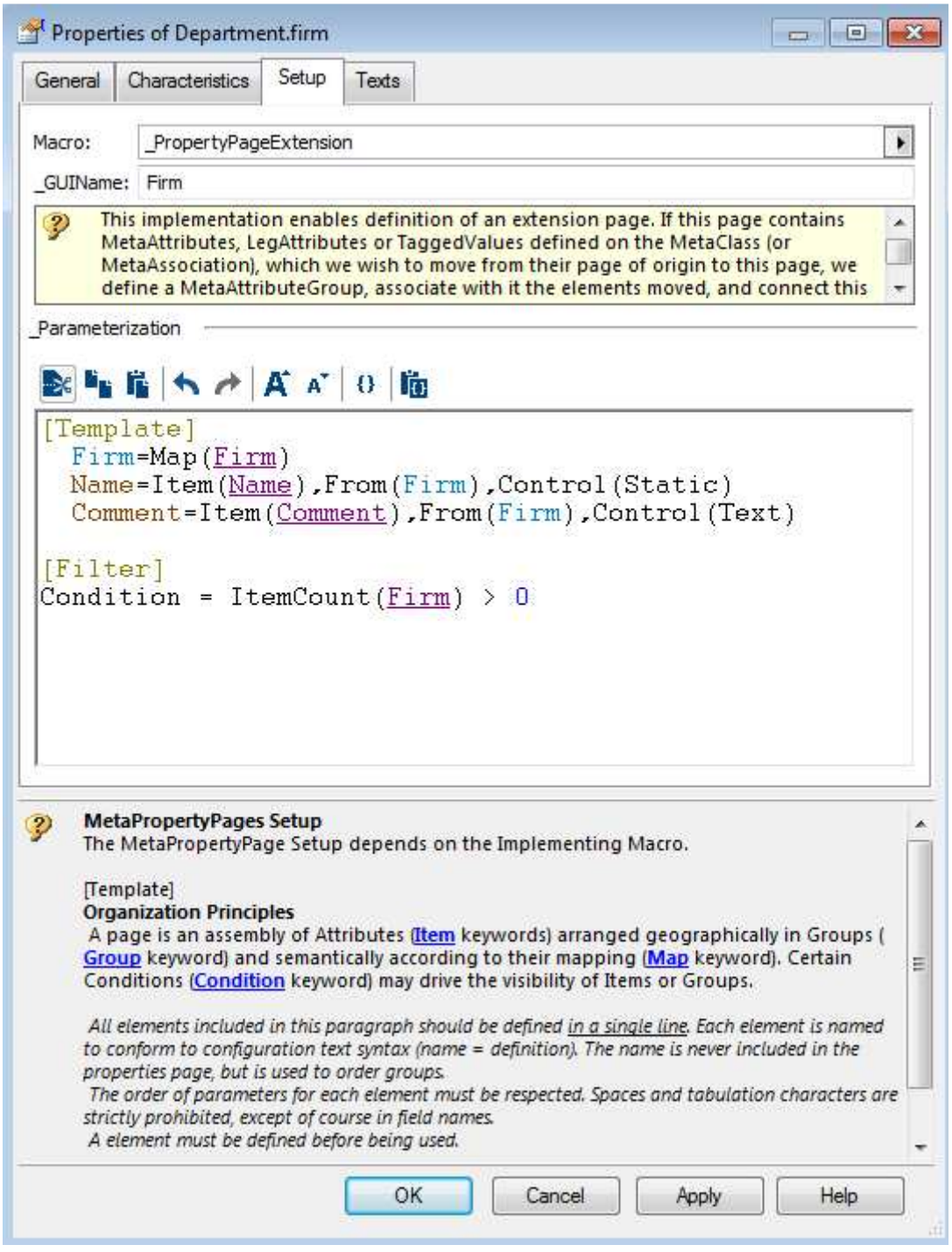
The following example will hide the "*Firm*" page previously defined on a *Department* when the Department is not associated with a *Firm*.
The condition envisaged will relate to MetaAssociationEnd ~(hmSXaOYEryC[Firm]
You can relate the condition to the number of associated *Firms* by means of test *ItemCount:*

```
ItemCount(~(hmSXaOYEryC[Firm]) > 0
```

In the [Filter] section, you must specify:

```
[Filter]
Condition = ItemCount(~(hmSXaOYEryC[Firm]) > 0
```

Conditions can relate to attributes of the object, to attributes of connected objects, or to context data (for example user metamodel access).
To determine conditioning possibilities, you can be guided by _Parameterization text entry help (called by pressing keys <Ctrl>+<Space>, requesting online help (F1 on condition keyword in _Parameterization text) or by consulting the help section available at bottom of the page. A chapter in "*HOPEX Forms*" document is also dedicated to this subject.


Properties of Department.firm
General
Characteristics
Setup
Texts
Macro:
_PropertyPageExtension
_GUIName:
Firm
This implementation enables definition of an extension page. If this page contains MetaAttributes, LegAttributes or TaggedValues defined on the MetaClass (or MetaAssociation), which we wish to move from their page of origin to this page, we define a MetaAttributeGroup, associate with it the elements moved, and connect this
_Parameterization
[Template]
Firm=Map(Firm)
Name=Item(Name),From(Firm),Control(Static)
Comment=Item(Comment),From(Firm),Control(Text)
[Filter]
Condition = ItemCount(Firm) > 0
MetaPropertyPages Setup
The MetaPropertyPage Setup depends on the Implementing Macro.
[Template]
Organization Principles
A page is an assembly of Attributes (Item keywords) arranged geographically in Groups (Group keyword) and semantically according to their mapping (Map keyword). Certain Conditions (Condition keyword) may drive the visibility of Items or Groups.
All elements included in this paragraph should be defined in a single line. Each element is named to conform to configuration text syntax (name = definition). The name is never included in the properties page, but is used to order groups.
The order of parameters for each element must be respected. Spaces and tabulation characters are strictly prohibited, except of course in field names.
A element must be defined before being used.
OK
Cancel
Apply
Help

Conditions on page elements

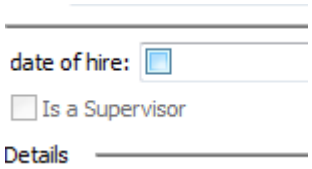
In the previous chapter we saw that it was possible to deactivate or hide an element. In particular, the keyword Visibility(Admin) enabled restriction of visibility to expert users: appearance of the page can therefore differ according to context.
This principle can be generalized: you can deactivate or hide elements and groups according to conditions explicitly defined in the page Template.
To do this, use keywords DisabledOn(*condition*) and HiddenOn(*condition*) after having defined a condition.
To illustrate conditioning, you shall specify that the "Is a Supervisor" attribute, visible in the Characteristics page of a Manager, will be activated when the Manager has *Employees*. The corresponding condition is declared in the page Template as follows:

```
HasEmployee=Condition(ItemCount(~demSW1OYEDCC[Employee]) > 0)
```

Simply cite this condition when redefining the 'Is a Supervisor' element.

```
Supervisor=Item(~6emSr2OYEnKC[Is a Supervisor]),DisabledOn(HasEmployee)
```

If the condition is false (if the Manager has no employees), the element is deactivated.



Similarly, the element would be hidden if you apply the keyword HiddenOn(HasEmployee).

You can directly deactivate or hide complete content of a group by applying these keywords to a Group in the Template. We can test this possibility on the Details group previously defined in the 'Manager' page.

```
Details=Group(Bar),Pos(Middle),Name(Details),HiddenOn(HasEmployee)
```

Redefining page template content

When content of a Properties page is highly dependent on specific characteristics of the object, definition of conditions can be complicated. To simplify this type of configuration, it is possible to extend Template content by means of an IncludeTemplate directive.
This directive initially enables inclusion in a Template paragraph of lines obtained from a text located on another object.

```
Extra=IncludeProfile(<objectId>),Text(<textid>),Paragraph(prgName)
```

If the keyword Text is not specified, the default text is _Parameterization.
If the keyword Paragraph is not specified, the default paragraph is 'Template'

In this first use mode, <objectId> represents an explicit system object; this mode enables reuse of a configuration in several templates.

In more highly developed use modes, it is possible to calculate the system object to which the customization will relate.

Configuration depending on a system object connected to the current object

This object can for example be obtained by browsing a MetaAssociationEnd from the current object, which is indicated by the keyword Origin(Service)

```
ExtraParam=IncludeProfile(<AssociationEndId),Origin(Service),Paragraph(prgN
ame:Template)
```

In particular, this device enables extension of an object page according to the _type of the object (_type being a system repository object).

Configuration depending on a MetaAttributeValue

Another possibility is to extend a page according to a particular value of MetaAttributeValue. In this case you must cite the MetaAttribute of the object and the keyword Origin(Value)
You can test this kind of extension on the Manager MetaClass.

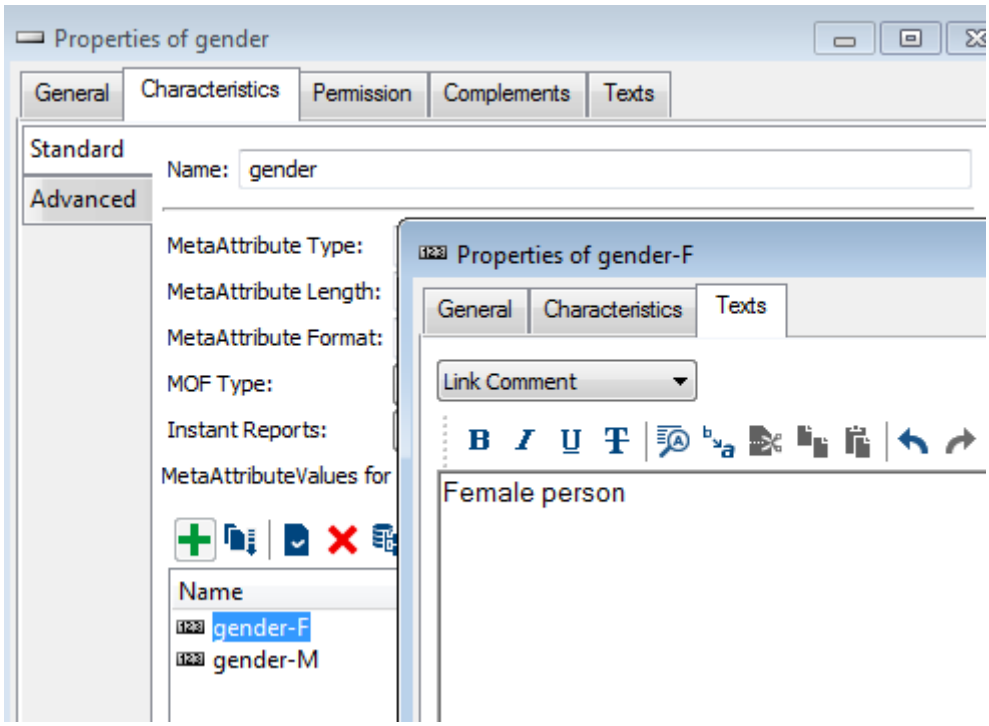
You will first comment the MetaAttributeValues of the gender MetaAttribute, then define a template extension specific to the MetaAttributeValue in the _Parameterization text of the MetaAttributeValue, enabling display of its comment:

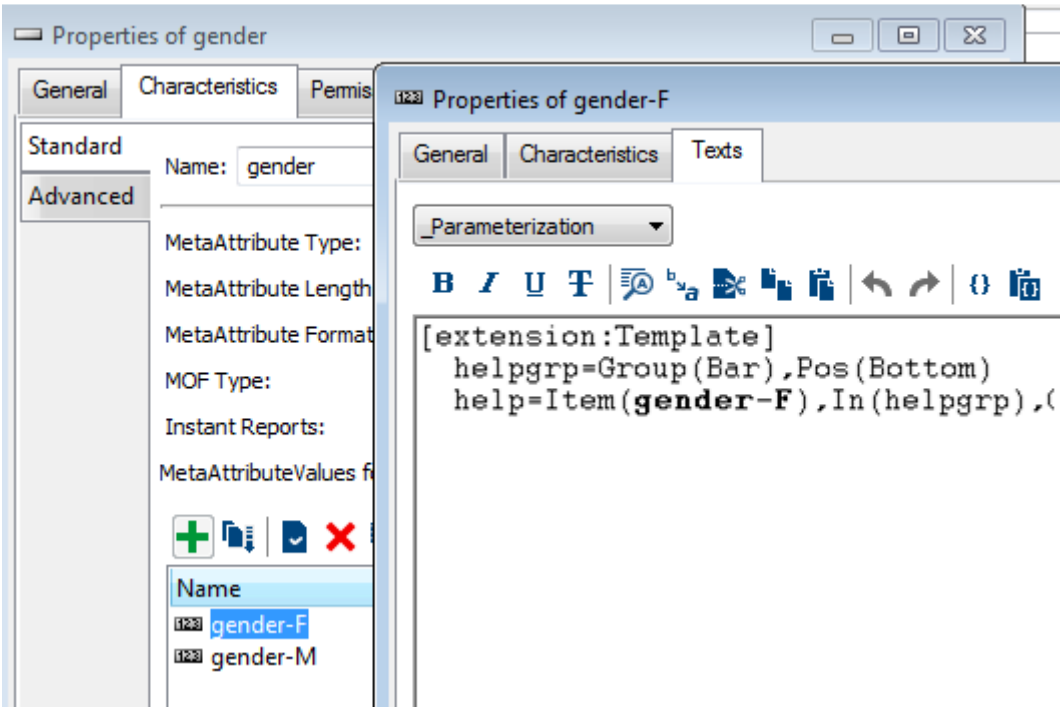
- For gender-F

```
[extension:Template]
   helpgrp=Group(Bar),Pos(Bottom)
   help=Item(~yhmSszNYEHuB[gender-F]),In(helpgrp),Control(HelpComment)
```

- For gender-M

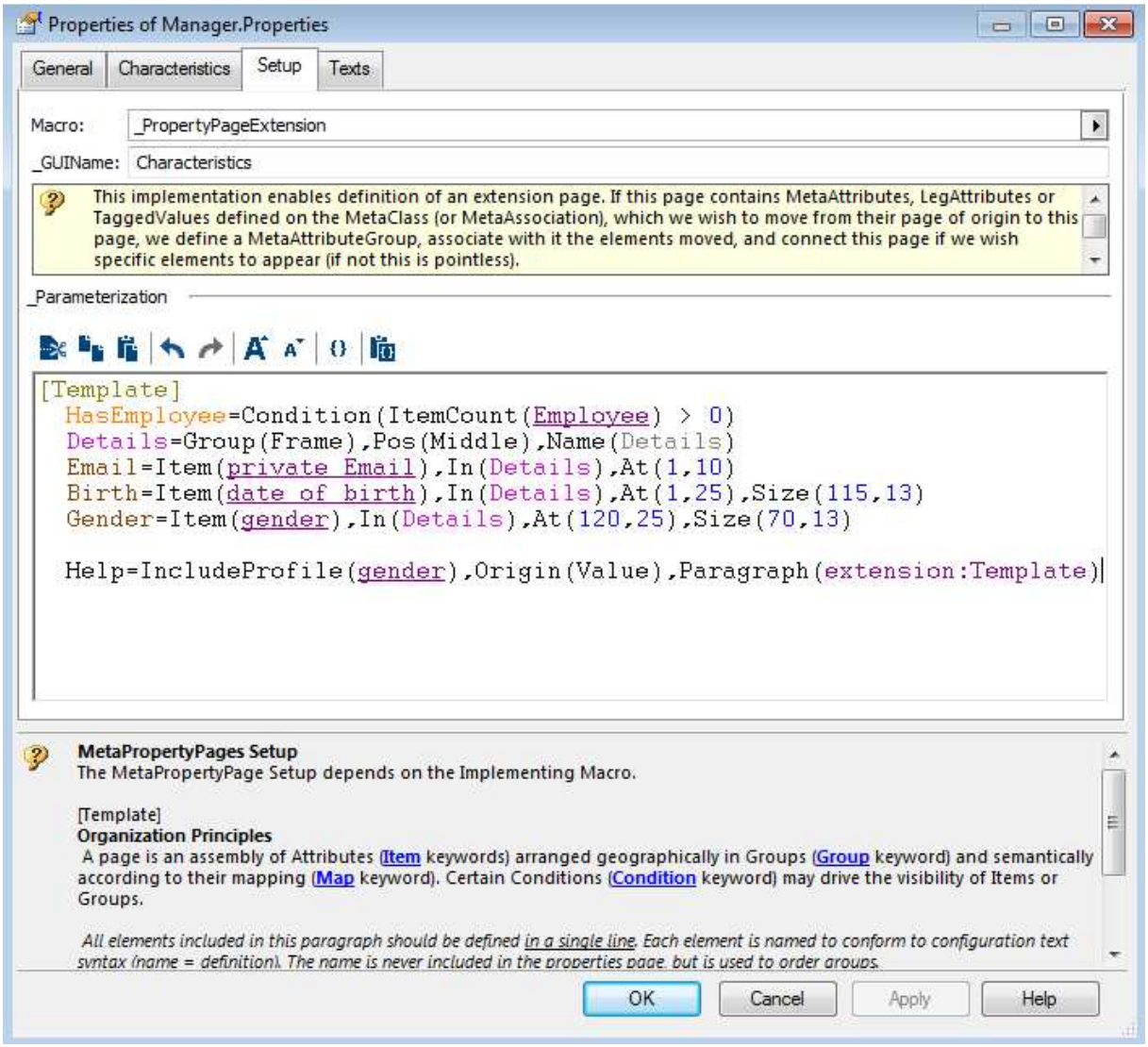
```
[extension:Template]
     helpgrp=Group(Bar),Pos(Bottom)
     help=Item(~IgmSXzNYEzoB[gender-M]),In(helpgrp),Control(HelpComment)
```





Finally, we add the inclusion directive in the Manager.Properties MetaPropertyPage :

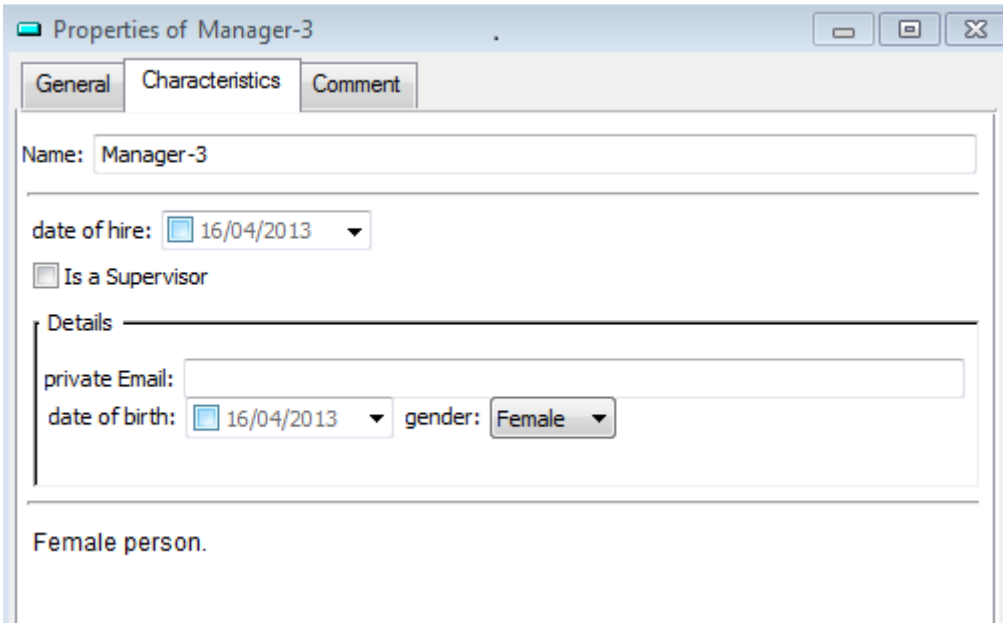
```
Help=IncludeProfile(~6emSkyNYETjB[gender]),Origin(Value),Paragraph(extensio
n:Template)
```

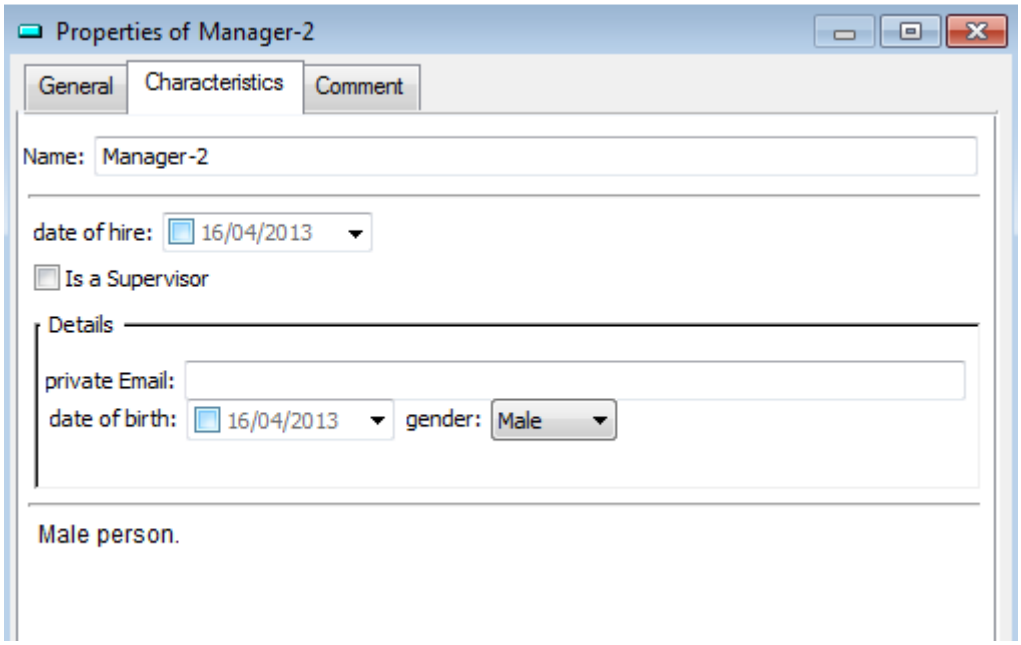


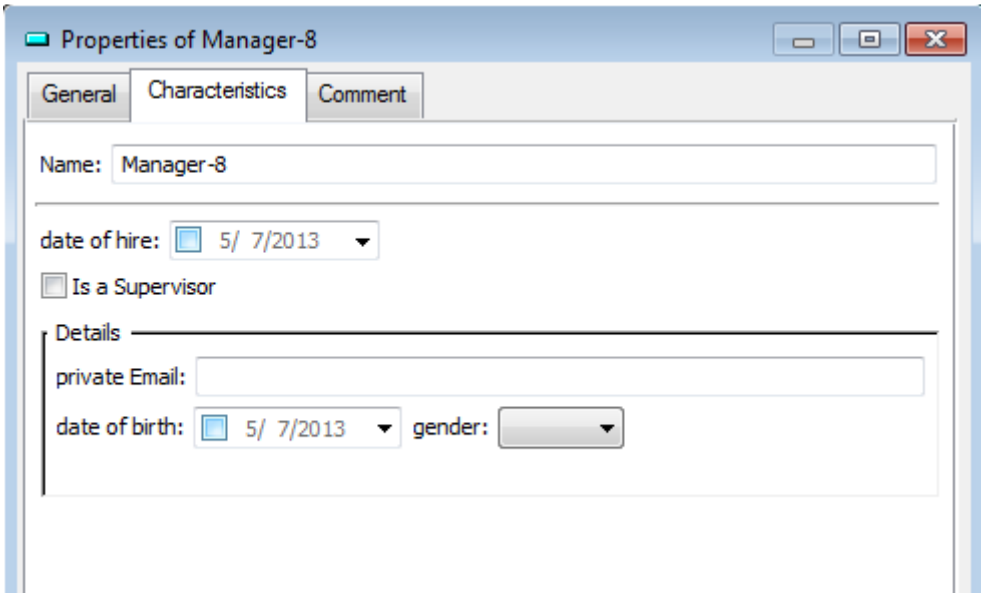
The help section will appear at bottom of the page (since it is in a Pos(Bottom group)) and will depend on the *gender* MetaAttribute.

If the value is not defined, nothing will be inserted.

Note that this configuration is calculated from an effective value: to note the change, click **Apply** to validate gender modification.


Properties of Manager-3
General
Characteristics
Comment
Name: Manager-3
date of hire: 16/04/2013
Is a Supervisor
Details
private Email:
date of birth: 16/04/2013
gender: Female
Female person.


Properties of Manager-2
General
Characteristics
Comment
Name: Manager-2
date of hire: 16/04/2013
Is a Supervisor
Details
private Email:
date of birth: 16/04/2013
gender: Male
Male person.



Configuration calculated by macro

You can dynamically generate page content by means of a Macro.

```
ExtraParam=IncludeProfile(<MacroId>),Origin(Macro)
```

The macro should implement the InvokeOnObject function which should return a character string representing a "Template" paragraph.

```
Function InvokeOnObject(object,idText,Page)
InvokeOnObject = "[Template]" & VbCrLf
End Function
```

In this way you can freely customize a Properties page, by making dependent the Template generated from the object supplied at input.

From knowledge to performance

HOPEX Forms - Mega Wizard Implementation - Tutorial

Objective

MEGA MetaStudio enables definition of wizards by assembling properties pages and non-interactive code, which can be written in Script.

These wizards have been designed to:

redefine MEGA object creation dialog boxes, covered in the first part of this document.

develop basic user interfaces in the form of dialog boxes or wizards, covered in the second part of the document.

Initializing the courseware

In order to be independent of evolution of the **MEGA** metamodel, this courseware is based on a specifically designed metamodel extension.

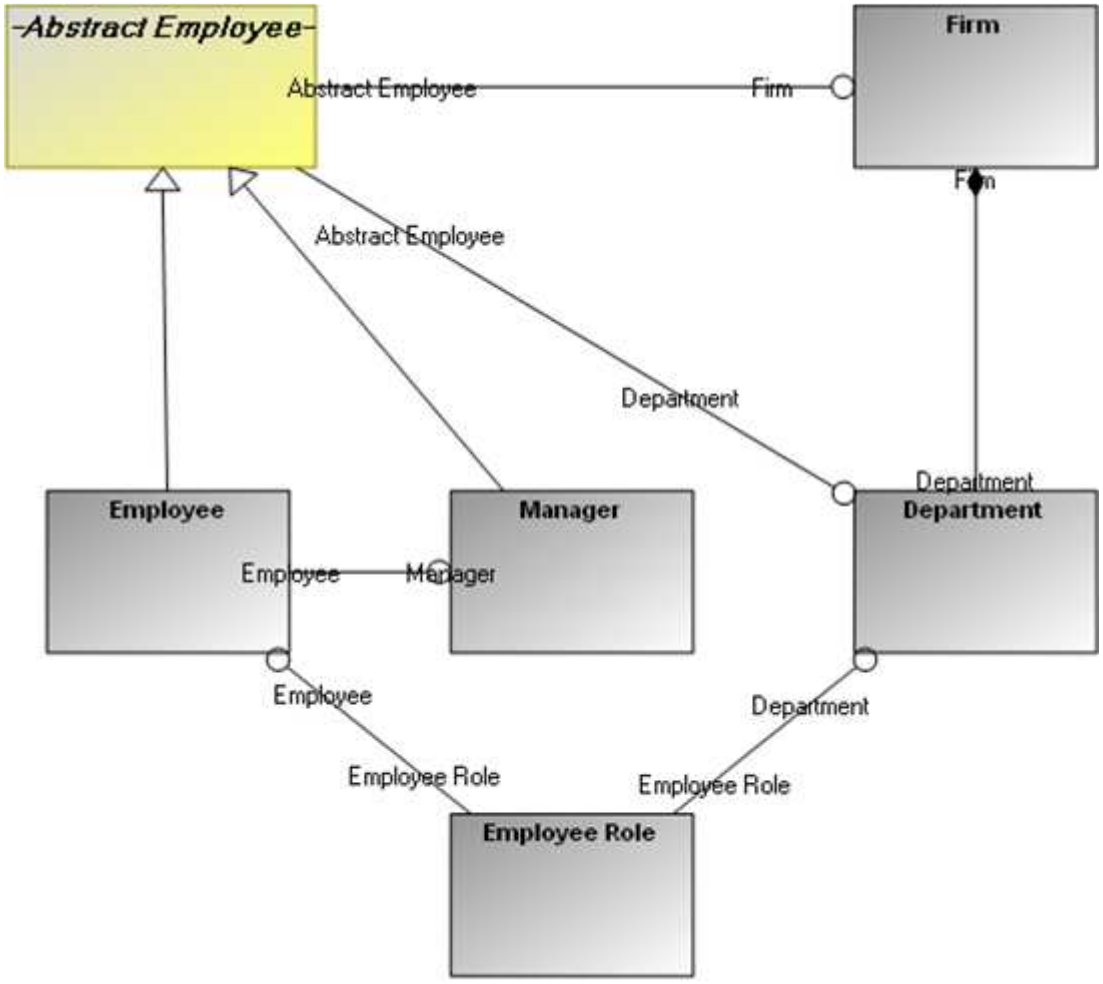
Before starting work, you must initialize your environment:

1. From MEGA, import into the system repository the command file below (this file is attached to PDF).

MetaModel Extensions for training courses.mgr

2. From the MEGA Administration module, recompile the metamodel.

Content of the specific metamodel is the following:



Customizing a creation wizard

From version 2007, but particularly in version 2009, the MEGA platform has greatly simplified redefinition of creation wizards for a MetaClass.

This customization can now be carried out at three definition levels. You can:

- if this satisfies the objective of the wizard, simply define the MetaAttributes, or Role Attributes that you want to use in the standard page of the creation wizard.
- insert initialization code for the object, or redefine the standard creation page according to context.
- add pages to the wizard, and if necessary modify their sequencing.

➔ The last two levels require creation of a Meta Wizard associated with the MetaClass. The Wizard processing code is implemented in one or in several Macros specified as Wizard Trigger. Pages defined for the wizard are MetaPropertyPages.

Prerequisites

Before customizing a creation wizard, you must define the following **environment options** values:

- **Metamodel Access**: "Expert"
- **Authorize Dispatched Objects Deletion** : "Authorize"

This customization example is based on the **Employee** MetaClass.

Adding properties in the standard creation page

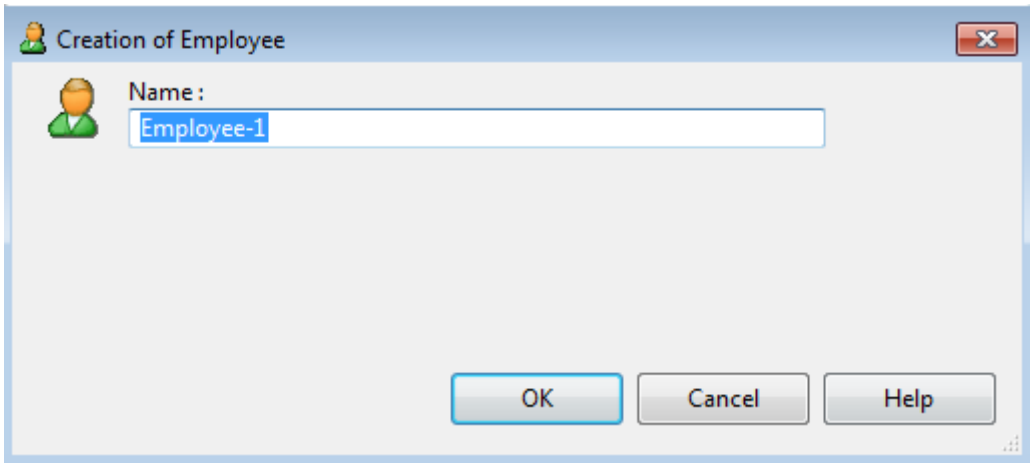
This first customization level is available in version 2007 of MEGA. It does not require creation of a Meta Wizard, since this customization is directly configured on the appropriate concepts of the metamodel by means of the 'Detailed Behavior' attribute. This attribute is present on:

- MetaAssociation (MetaClass/MetaAttribute)
- MetaAssociationEndAttribute (via the MetaAssociationEnd _LegAttribute, which indicates that the role wishes to be seen as an attribute).

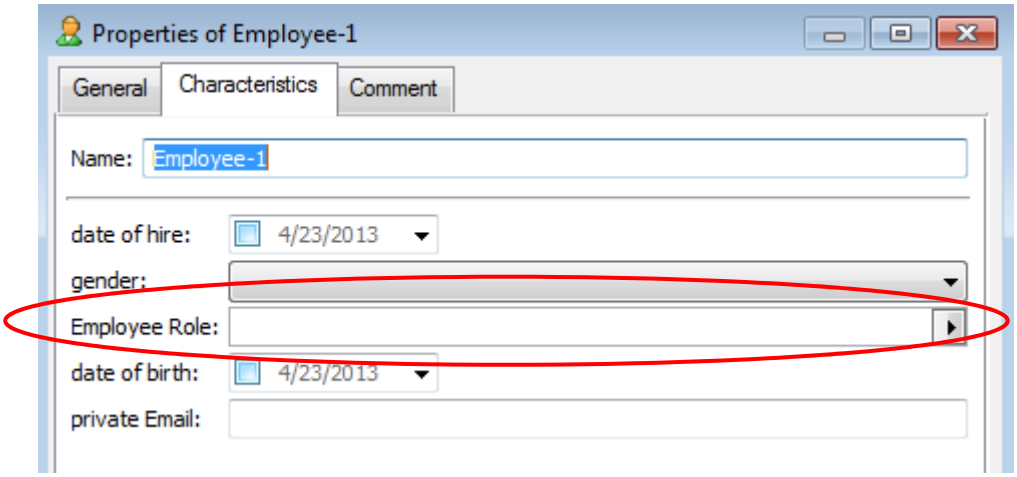
The 'Detailed Behavior' attribute is modified via a multiple choice list:

- '**Creation**' indicates that the element must appear in the object creation page
- '**Mandatory**' indicates that the element is required
- '**Immutable**' indicates that the element can only be modified at the time of creation
- '**Category**' indicates that the element defines a category for the object. This category can be used in the creation wizard, which can modify itself according to the category when the latter is initially specified: the name of the category and the associated image replace that of the MetaClass in the wizard.

Example: the creation wizard of the "**Employee**" MetaClass has not been customized. It is presented in the following form:

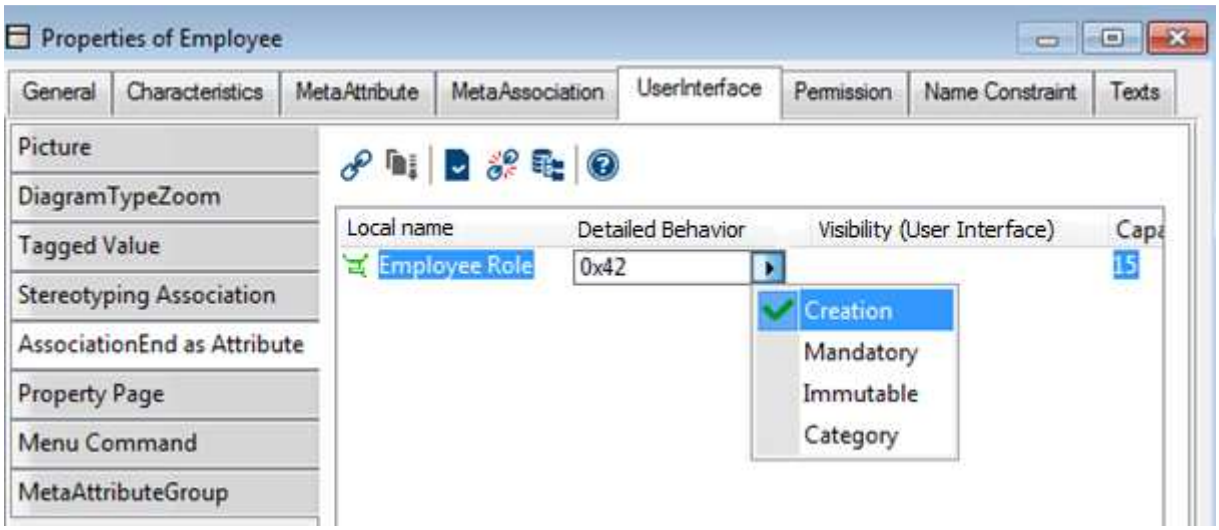


In the properties page of an **Employee**, there is the "**Employee Role**" role which we want to be able to specify at creation of the object.

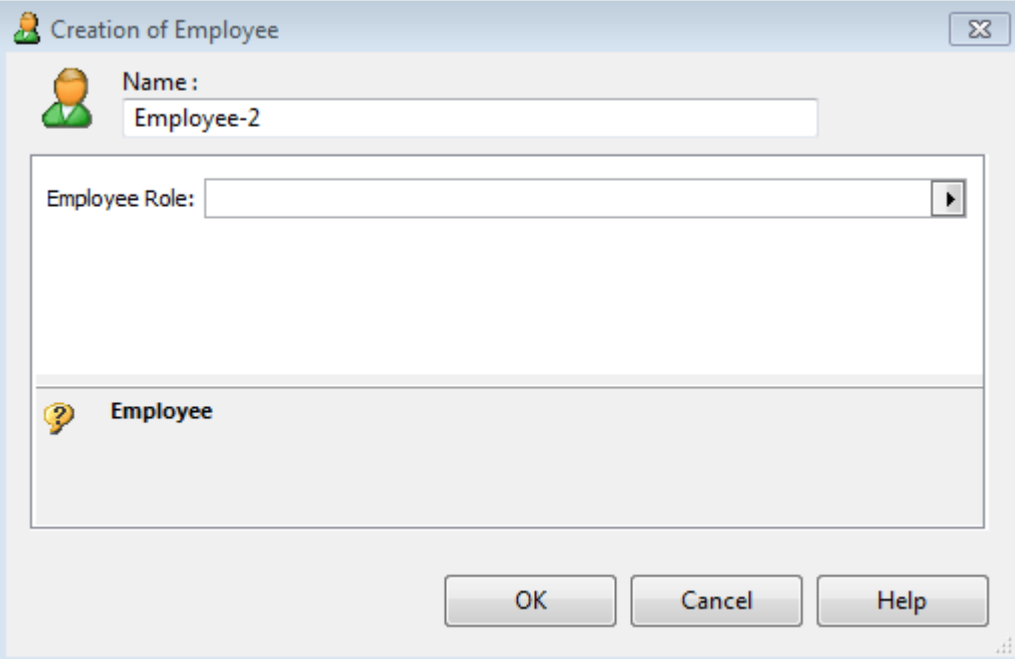


To do this:

1. From the **Employee** MetaClass Properties page, in the **UserInterface** tab, select the **AssociationEnd as Attribute** subtab.
2. Update the '**Detailed Behavior**' attribute of the link _LegAttribute **Employee Role**.



- On completion of this configuration, the **Employee** MetaClass creation wizard becomes:



Adding processing code to a creation wizard

In this chapter we shall consider the case of a configuration of the wizard not requiring page definition, but restricting itself to modifying behavior of the standard wizard by means of a trigger.

To do this, you shall consider the example of the **Employee** creation wizard, modifying its specification:

- The **Employee Role** role should not appear until the **Employee** is created from a **Manager**.
- The comment of the **Employee** should be initialized with the comment of the **Employee Role** when the latter is defined.

To implement the first requirement, it is not possible to configure the MetaClass, since this configuration is not conditional.

- Remove the "Creation" **Detailed Behavior** on the "Employee Role" role of the **Employee** MetaClass.

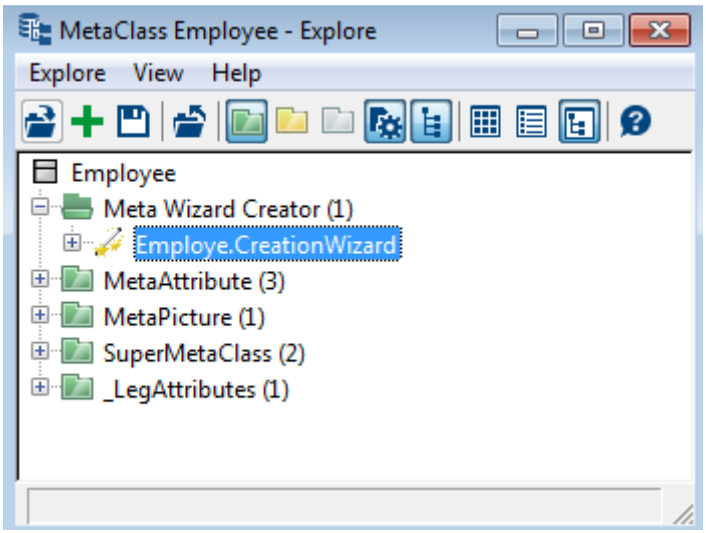
As for the second requirement, you must be able to insert code executed at object creation.

- To carry out this customization, you must implement a Meta Wizard.

Creating a wizard and its trigger

To create this Meta Wizard:

1. In the **MetaStudio** tab, expand the **Employee** MetaClass.
2. Right-click the **Meta Wizard Creator** folder and create a new Meta Wizard.
3. Name this Meta Wizard : Employee.CreationWizard.

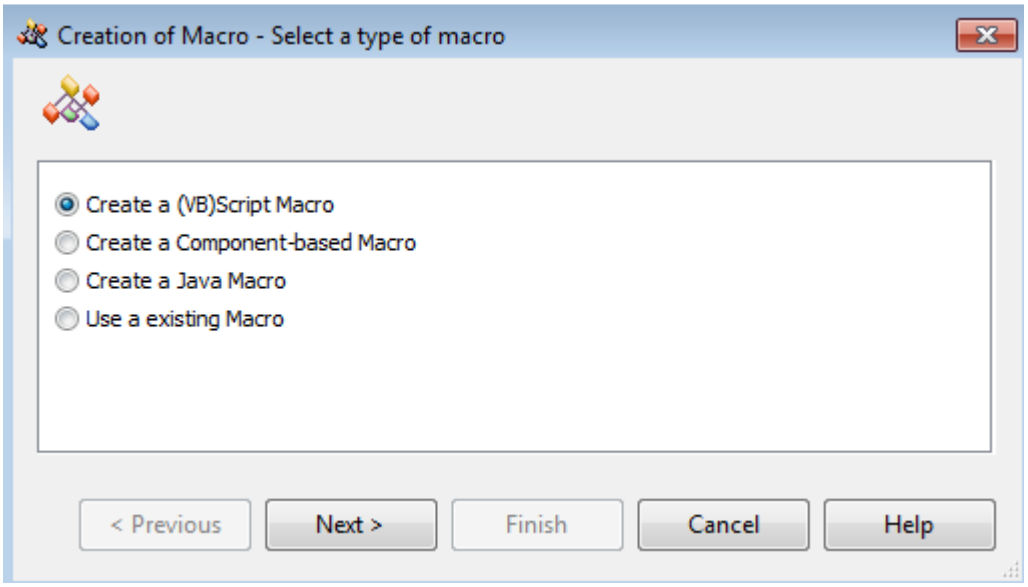


To create a trigger for this wizard:

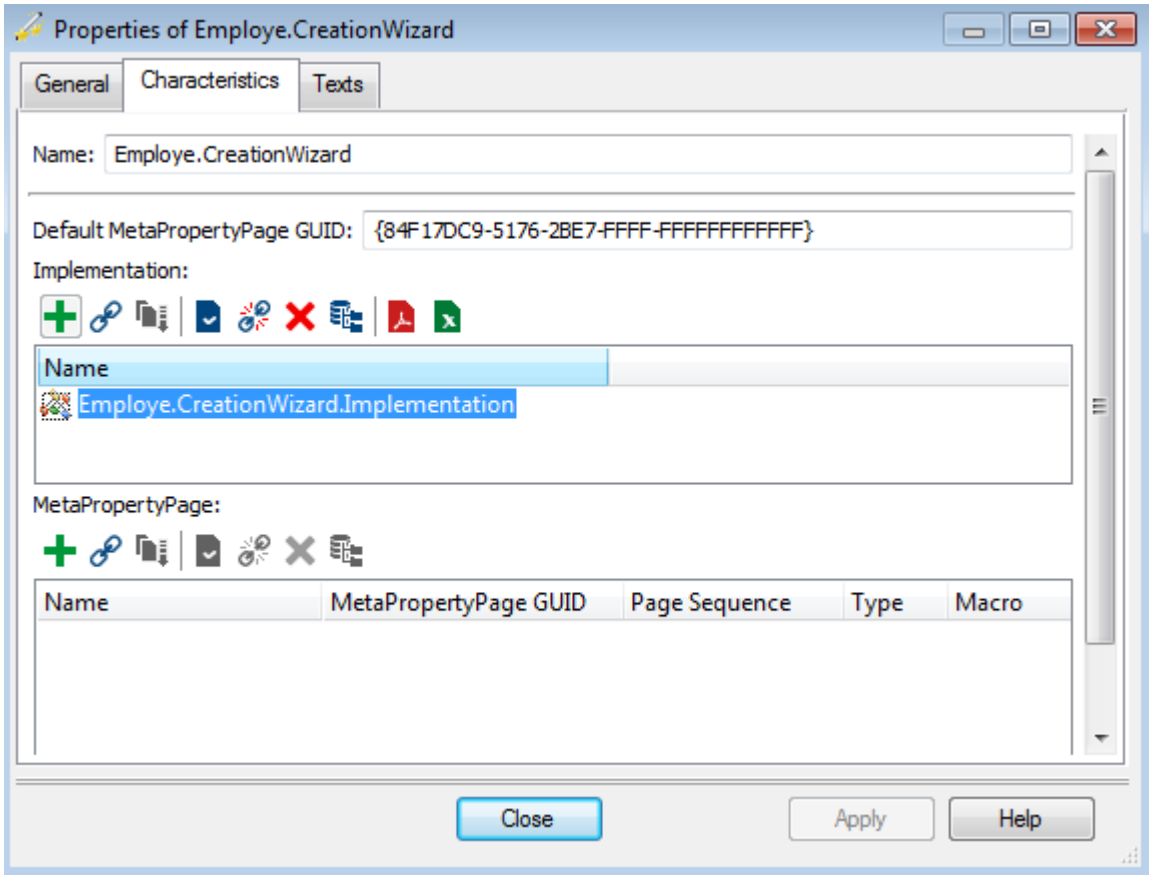
In this example, you can implement the trigger in JAVA or in (VB)Script.

In version 2009 SP1, it must be created from the explorer

1. From the **Meta Wizard Creator** folder, open the Properties of the "Employee.CreationWizard" Meta Wizard.
2. In the **Implementation** frame, click **New**.
3. Select **Create a (VB)Script Macro** (or **Create a Java Macro**).



4. Click **Next**, then **Finish**.



5. Open the **Properties** dialog box of Employee.CreationWizard.Implementation. In the **VB Script** tab, copy the code given in example P. 13.

Trigger interface description

A wizard trigger enables modification of wizard behavior. In principle, functions implemented in the trigger are called on each transition of the wizard. The list of functions which can be implemented is as follows:

```
void OnWizInitialize(MegaWizardContext mwctx);
void OnWizPageInitialize(MegaWizardContext mwctx,MegaPropertyPage mpage);
void WizPageCheck(MegaWizardContext mwctx, MegaPropertyPage mpage,Integer[]
result);
void OnWizNext(MegaWizardContext mwctx, MegaPropertyPage mFrom,
MegaPropertyPage mTo);
void OnWizPrevious(MegaWizardContext mwctx, MegaPropertyPage mFrom,
MegaPropertyPage mTo);
void OnWizRealize(MegaWizardContext mwctx);
void OnWizBeforeTerminate(MegaWizardContext mwctx, MegaPropertyPage
mpage,Integer[] result);
void OnWizTerminate(MegaWizardContext mwctx);
void OnWizCancel(MegaWizardContext mwctx);
void WizPageNextChange(MegaWizardContext mwctx, MegaPropertyPage
mpage,StringBuffer nextPageId);
void WizPagePreviousChange(MegaWizardContext mwctx, MegaPropertyPage
mpage,StringBuffer PreviousPageId);
```

Function **OnWizInitialize** is called at initialization of the wizard. It is called before creation of the user interface. It is in this function that you can:

- add or remove properties in the default page
- modify wizard display mode (for example force complete mode when called in "in place" mode.

Function **OnWizPageInitialize** is called at initialization of a wizard page. It is called only once per page, when the wizard needs to collect information about it; it can thus be at any moment in the wizard lifetime, and not necessarily at the moment (but obviously before) the page is shown.

Function **WizPageCheck** is called at least each time the wizard prepares to display a next or previous page; it enables dynamic masking of a page and redefinition of the state of the previous, next and finish buttons. It can be called any number of times.
Functions **OnWizNext** and **OnWizPrevious** are called on each transition between two pages of the wizard.
Function **OnWizRealize** is called at object creation. We shall see later that this is not necessarily at the end of the wizard.
Function **OnWizBeforeTerminate** is called before the termination action; it enables a final check before the code terminating the wizard. It is particularly useful on wizards which have only one page, and therefore no transition.
Function **OnWizTerminate** is called when pressing the finish button (or OK if only one page is displayed).
Function **OnWizCancel** is called at cancellation of the wizard (Cancel button or explicit cancel request during processing). In this case, any MEGA updates carried out during wizard processing are automatically cancelled.
It is possible to force display of a next or previous page using functions **WizPageNextChange** and **WizPagePreviousChange**.

Wizard default page

Note that for the moment, no specific page has been defined for the wizard we are currently creating. A wizard in this state will continue to use the default page, a page defined for non-customized wizards. It should be noted that this page will remain visible in the wizard as long as the wizard considers it useful, or as long as customization does not explicitly mask it. By default, this page contains, in addition to the name and the owner, the properties which have been declared visible at object creation.

Wizard context: MegaWizardContext interface

Customizations mentioned in our example require knowing the creation wizard call context, since it is specified that behavior differs depending on whether the **Employee** is created from a **Manager** or not.
All information relating to context of the wizard and its state are accessible in the MegaWizardContext object, which is transmitted to each of the trigger functions.

This interface is quite complex. We will cover its different aspects in the course of this document. Note however the presence of functions enabling access to the following properties:

mode

Determines wizard start mode. There are at present 3 main wizard start modes.

- mode(1): complete mode, as used particularly in navigators
- mode(0): 'inPlace' mode. This mode is used in ListViews on the basis that it is not necessary to enter the name of the object to be created, either because this name is already fixed and we do not consider it necessary to offer the possibility of modification, or because it will be modifiable later (in the listview, the name of the newly-created object is modifiable 'inPlace' in the list). It is of course possible to customize the wizard by forcing complete mode.
- mode(2): we wish to call the creation wizard from a non-interactive tool. This mode is required when we consider that only the wizard is capable of creating a 'valid' object, and we wish to ensure that the code executed in the triggers and used to initialize the object will be called. In no event can this mode present a user interface, and creation will fail if all prerequisites of creation are not satisfied; if for example a property is declared as mandatory, and if its value is not known or cannot be deduced in the wizard, the wizard will fail.

It is possible to modify the mode in function **OnWizInitialize**. This is not possible later, the wizard having already started.

Name

Contains the name we wish to give the object during creation. It can be modified in the trigger code.

parentTypeID and parentID

When the object is created from another object (create-connect) these properties indicate the origin object of creation. *ParentID* contains the absolute identifier of this origin object, and *parentTypeID* that of the MetaAssociationEnd (viewed from source) via which we wish to connect the object after its creation.

sourceID and targetID

When the object is created from a link action in a diagram, these two properties contain the source and target absolute identifiers of the action.

kindID

When category of an object is already known, this property contains its absolute identifier. When an image has been associated with the category, this image replaces that of the MetaClass in the wizard; similarly, the name of the category replaces the name of the MetaClass.

template

Represents the object being created, in the form of a MegaObject. In most cases, we can use this as an existing object, but a certain number of functionalities are not accessible to such an object due to its non-existence. In particular, virtual attributes and associations are not suitably managed by this object. Its absolute identifier can however be modified.

property and propertyFlag

These functions enable definition and consultation of properties associated with the context of the wizard. These properties are addressed by an absolute identifier.

These properties represent either:

- Properties of the object being created: this case is automatically detected when the identifier provided corresponds to the absolute identifier of a property (MetaAttribute, TaggedValue or LegAttribute) of the object. In this case the property of the context enables fixing of the initial value of this property (*property*), as well as its display mode in the wizard (*propertyFlag*. See details of these *flags*) below.
- Data specific to the wizard, which we shall call 'Cookies', use of which will be covered later in the document.

Dynamic addition of a property in the default page

Here we shall consider the first customization, which consists of adding the '**Employee Role**' role in the creation page when the **Employee** is created from a **Manager**.
This customization must be implemented in the **OnWizInitialize** function; it is the only function which is always called before creation of the default page, and therefore enables modification of the content of this page.

The following is an example of Java code implementing this customization:

```
import com.mega.modeling.api.*;
import com.mega.modeling.api.util.MegaWizardContext;

public class EmployeeWizardTrigger {

  public void OnWizInitialize(MegaWizardContext wctx) {
    MegaToolkit mToolkit = wctx.template().getRoot().currentEnvironment().toolkit();
    if(mToolkit.sameID(wctx.parentTypeID(),"~demSW1OYEDCC[Employee]")) {
      wctx.propertyFlag("~y4mZK)OYErIH[Employee Role]", "Visible", true);
      wctx.propertyFlag("~y4mZK)OYErIH[Employee Role]", "Updatable", true);
    }
  }
}
```

In (VB)Script this example is as follows:

```
'MegaContext(Fields,Types)
'Uses(Components)
Option Explicit

Sub OnWizInitialize(wctx As MegaWizardContext)
 Dim mToolkit As MegaToolkit
 Set mToolkit = wctx.template.getRoot.currentEnvironment.Toolkit
 If mToolkit.sameID(wctx.parentTypeID, "~demSW1OYEDCC[Employee]") Then
   wctx.property("~y4mZK)OYErIH[Employee Role]", "Visible") = True
   wctx.property("~y4mZK)OYErIH[Employee Role]", "Updatable") = True
  End If
End Sub
```

We first determine whether the object is created as a **Manager** by comparing the type of the parent with the MetaAssociationEnd concerned (to do this, we use the sameID method available in the MEGA toolkit).
In this case, we inform the wizard that the property concerned (here the '**Employee Role**' role) is visible and can be entered. The propertyFlag function previously evoked enables fixing of flags "Visible" and "Updatable" and therefore make the property visible in the wizard.

The available flags are:

Visible = True makes property visible. It is read-only by default. Visible = False does not mask the property, but only indicates that we wish standard behavior on this property.

Updatable = True authorizes modification of the property (but does not make it visible).

Hidden = True masks the property.

Mandatory = True makes property mandatory.

In Script, the propertyFlag function does not exist; we use the property function with two parameters to access the flag, the second being the name of the flag.

Adding a processing in the wizard

The second customization in the proposed example consists of adding a processing on a newly-created object: the comment of the **Employee** must be initialized with the comment of the **Employee Role** when the latter is defined.

A processing of this type can be inserted in any transition; however in the case of a wizard without a specific page, the only possible transitions are in the function:

- OnWizRealize, immediately after object creation,
- OnWizTerminate at the end of execution of the wizard.

The proposed initialization must take place as late as possible, so as to allow the wizard to specify an **Employee Role** or a comment for the **Employee**.

- it is therefore preferable to include the processing in the final function, that is in **OnWizTerminate**.

The following is an example of code implementing this customization:

In Java:

```
public void OnWizTerminate(MegaWizardContext wctx) {
  MegaObject mEmployee = wctx.template();
  if(mEmployee.getProp("~f10000000b20[Comment]").compareTo("") == 0) {
   if(mEmployee.getCollection("~y4mZK)OYErIH[Employee Role]").count() > 0) {
    MegaObject mRole = mEmployee.getCollection("~y4mZK)OYErIH[Employee Role]").item(1);
    String sComment = mRole.getProp("~f10000000b20[Comment]");
```

```
      mEmployee.setProp("~f10000000b20[Comment]", sComment);
     }
    }
  }
```

En (VB)Script:

```
    Sub OnWizTerminate(wctx As MegaWizardContext)
     Dim mEmployee As MegaObject
   Set mEmployee = wctx.template
  If mEmployee.getProp("~f10000000b20[Comment]") = "" Then
   If mEmployee.getCollection("~y4mZK)OYErIH[Employee Role]").count > 0 Then
     Dim mRole As MegaObject
        Set    mRole    =    mEmployee.getCollection("~y4mZK)OYErIH[Employee
Role]").item(1)
     Dim sComment As String
     sComment = mRole.getProp("~f10000000b20[Comment]")
     mEmployee.setProp "~f10000000b20[Comment]", sComment
 End If
 End If
End Sub
```

The object in course of creation, which is necessarily created at the OnWizTerminate transition is always accessible via the *template* function. The above code updates the comment from that of the **Employee role**. Note here that update only occurs if the object comment is empty. This is not insignificant:

You should try to ensure that a customization disturbs later customizations of the wizard as little as possible. Suppose in the present case that after this customization or at least independently, we decide to make the comment of an **Employee** visible in the creation wizard by modifying the 'Detailed Behavior' attribute. In this case, the comment of the object appears on the creation page of the wizard, and the creator may be led to initialize it. If the customization above did not take account of the fact that the Comment might already be initialized and overwrites the user entry, behavior of the wizard would be difficult to understand.

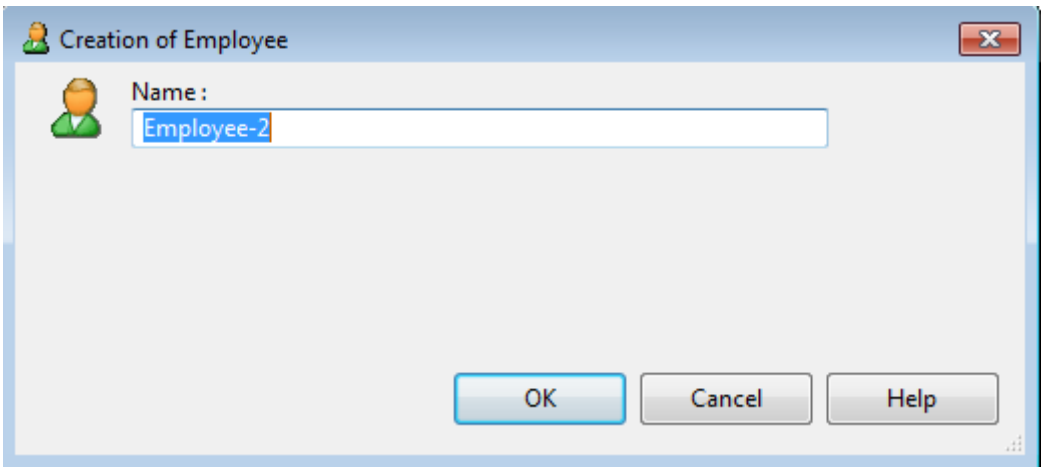
Testing the wizard

To test the wizard, you can:

- Create an **Employee** from the explorer
- Create a **Manager** from the explorer
- Create an **Employee** from a **Manager**
- Connect an **Employee**, already created, to a **Manager**.

To create an **Employee**:

1. From the desktop, run the explorer.
2. Select **Explore > Create**.
3. In the **Choose MetaClass** dialog box, select **Employee**.



4. Click **OK**.

   Employee-2 is created.

   Note that **Employee Role** information does not appear.

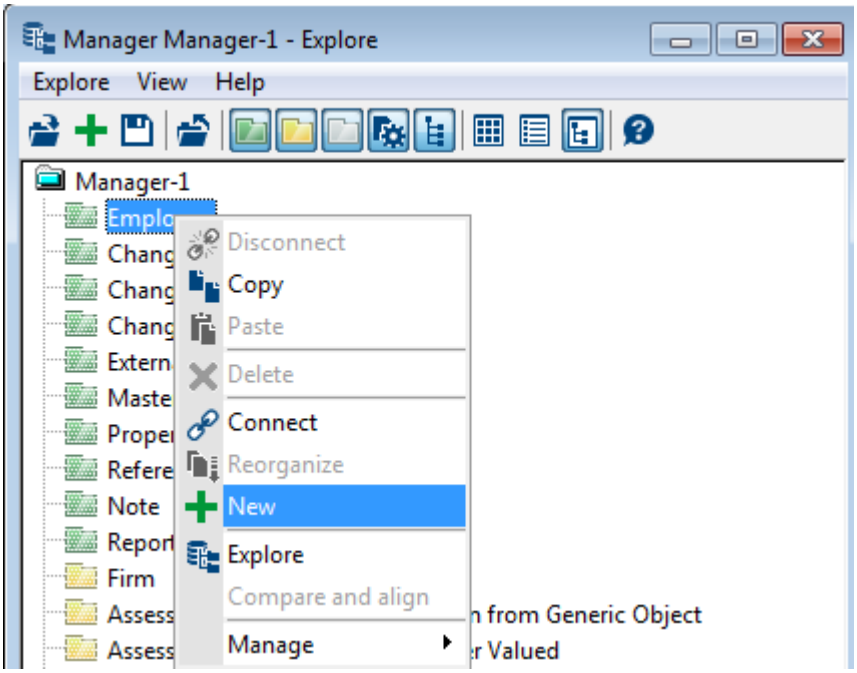
To create a **Manager**:

1. From the desktop, run the explorer.
2. Select **Explore > Create**.
3. In the **Choose MetaClass** dialog box, select **Manager**.
4. Click **OK**.

   Manager-1 is created. Note that conforming to the metamodel diagram, the **Employee** MetaClass is under "Manager-1".

To create an **Employee** from "Manager-1":

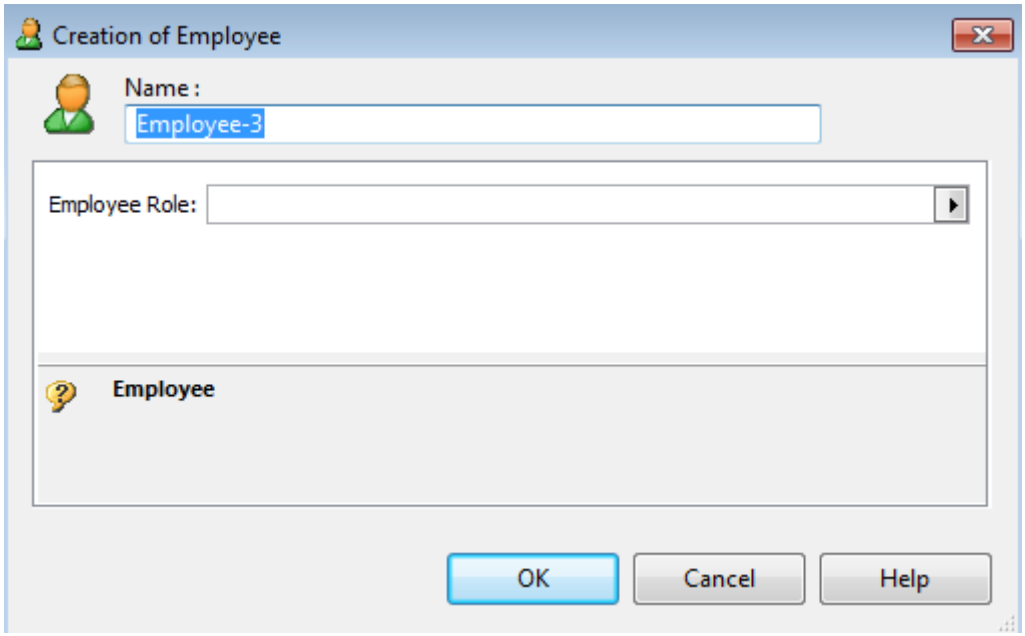
1. From the explorer of "Manager-1", right-click **Employee** and select **New**.



2. In the **Creation of Employee** dialog box, click **OK**.

   "Employee-3" is created.

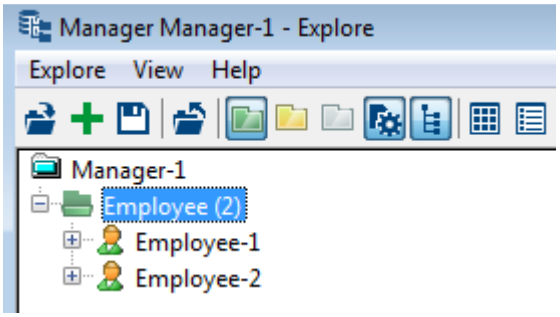
   Note this time that the **Employee Role** field appears in the creation page.



To connect an **Employee** already created, from "Manager-1":

1. From the explorer of "Manager-1", right-click **Employee** and select **Connect**.
2. In the query tool, select **Employee-1** and click **Connect**.

"Employee-1" is connected to "Manager-1".



Adding independent triggers

You can associate several independent triggers with a wizard. The order of the link between the *MetaWizard* and the associated macros determines trigger call order.

Defining new pages in a creation wizard

Another type of creation wizard customization has been envisaged. It consists of adding steps in the creation process, in the form of pages.
In a wizard of this type, the 'OK' button is replaced by 'Previous', 'Next' and 'Finish' buttons, enabling navigation between pages defined in this way.
You can define wizards which have several steps without implementing a trigger, the simple logic of succession of pages defined in the wizard being sufficient to achieve the customization objective.

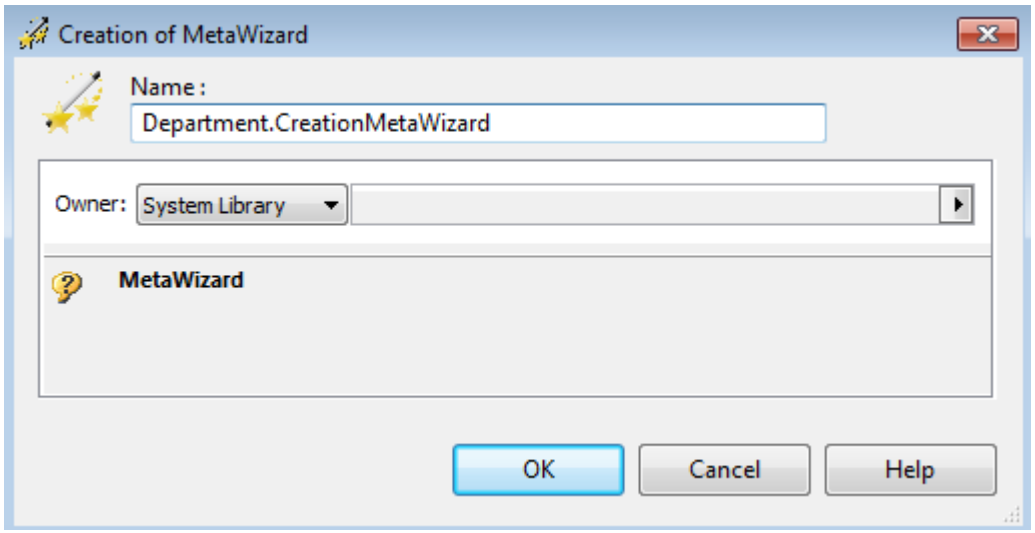
This system also enables presence in the creation wizard of elements that are not properties of the object to be created.

Here you will implement the following customization:

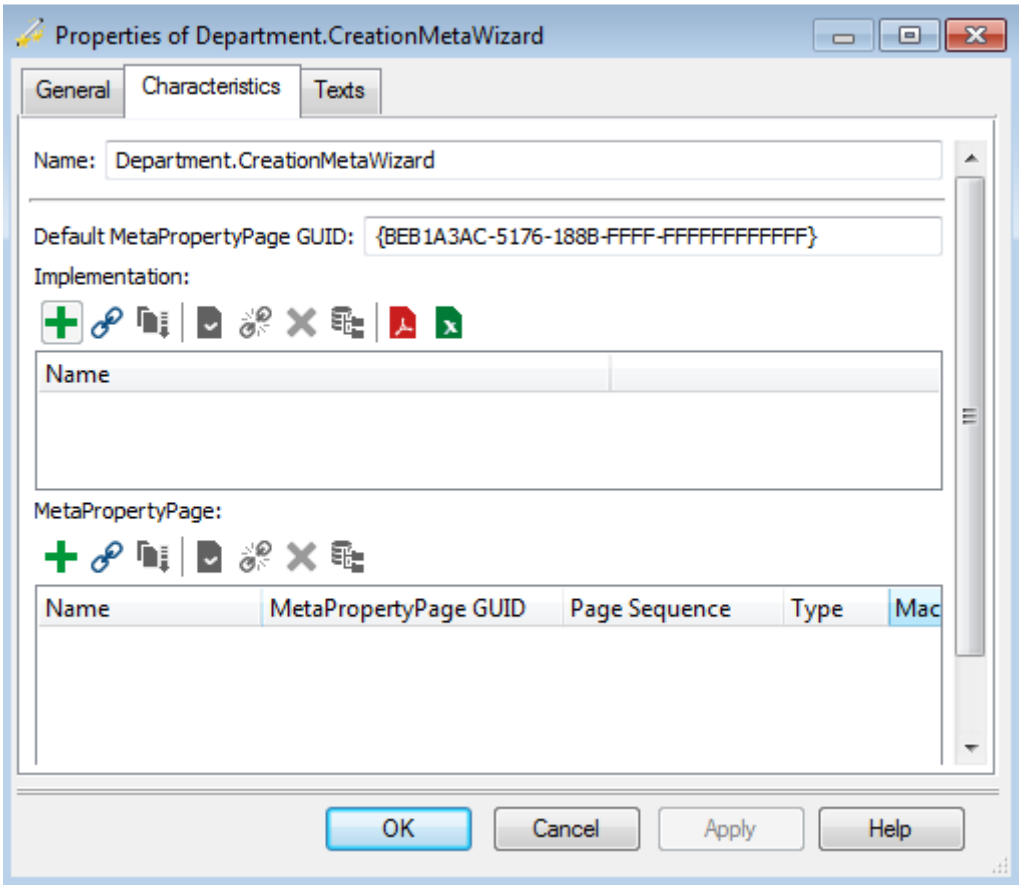
> "The list of **Abstract Employees** (comprising **Employees** and **Managers**) associated with a **Department** must be accessible in the creation wizard".

To do this:

1. Create the creation wizard of the **Department** MetaClass.



2. Open the properties page of the MetaWizard you have just created.

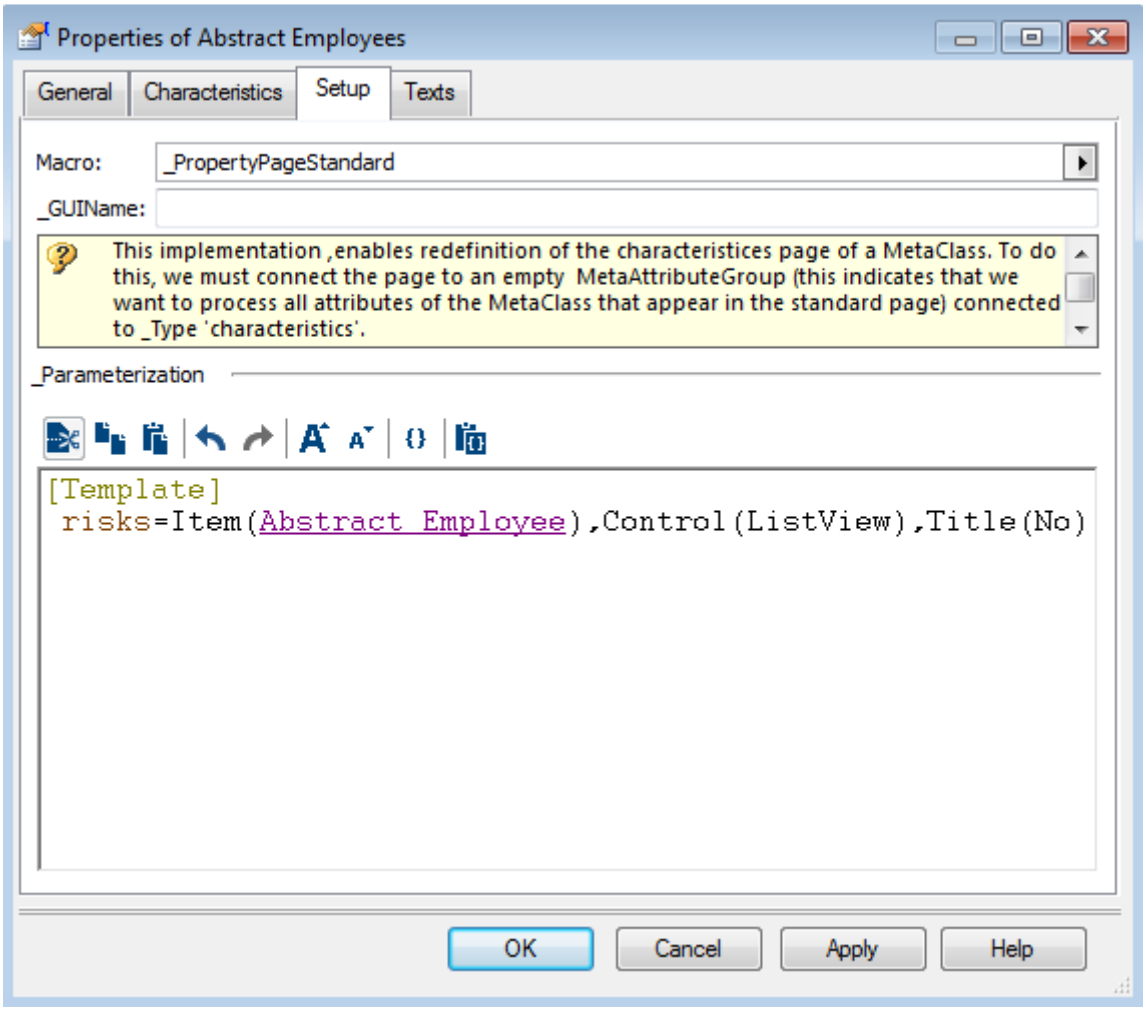


The pages you can integrate in a wizard are similar to object properties pages, and are modeled by **MetaPropertyPage**.

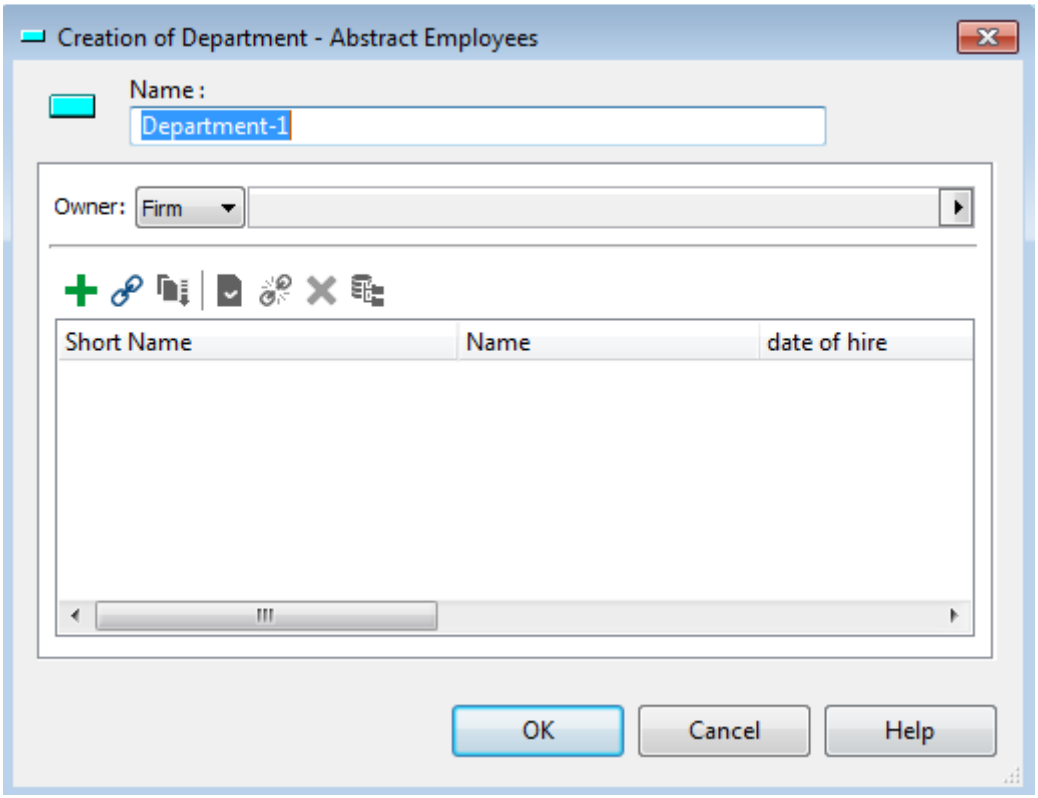
In the above wizard, you will create a page "Abstract Employees", display its properties dialog box and define its configuration:

1. In the **MetaPropertyPage** frame, click **Create**.
2. Name the page "Abstract Employees".
3. In the **Macro** field, click the PropertyPage.Kind menu and select macro **_PropertyPageStandard**.
4. Click **OK** to take this macro into account.
5. To insert the list of Abstract Employees, in the same way as for a properties page (for more information see the document on properties pages configuration), in the « Abstract Employees » Properties window select the **Setup** tab and in the **_Parameterization** frame, enter:

```
[Template]
risks=Item(~i5mZlVQYEHoI[Abstract Employee]),Control(ListView),Title(No)
```



6. Clixk **Apply**.
7. Start the creation wizard.



The wizard generates only a single page**:** each non-preparatory page includes name entry; from this the wizard deduces that this page displays all data necessary for execution.

The implementation '_PropertyPageStandard' corresponds to implementation of the default page, and therefore displays entry of the potential owner of the object.

The only significant difference compared with the default page: the help area is not present. This is an unfortunate anomaly which will be corrected…

The result does not perhaps conform to what we had hoped, and to achieve this we shall specify our customization request. The wizard should:

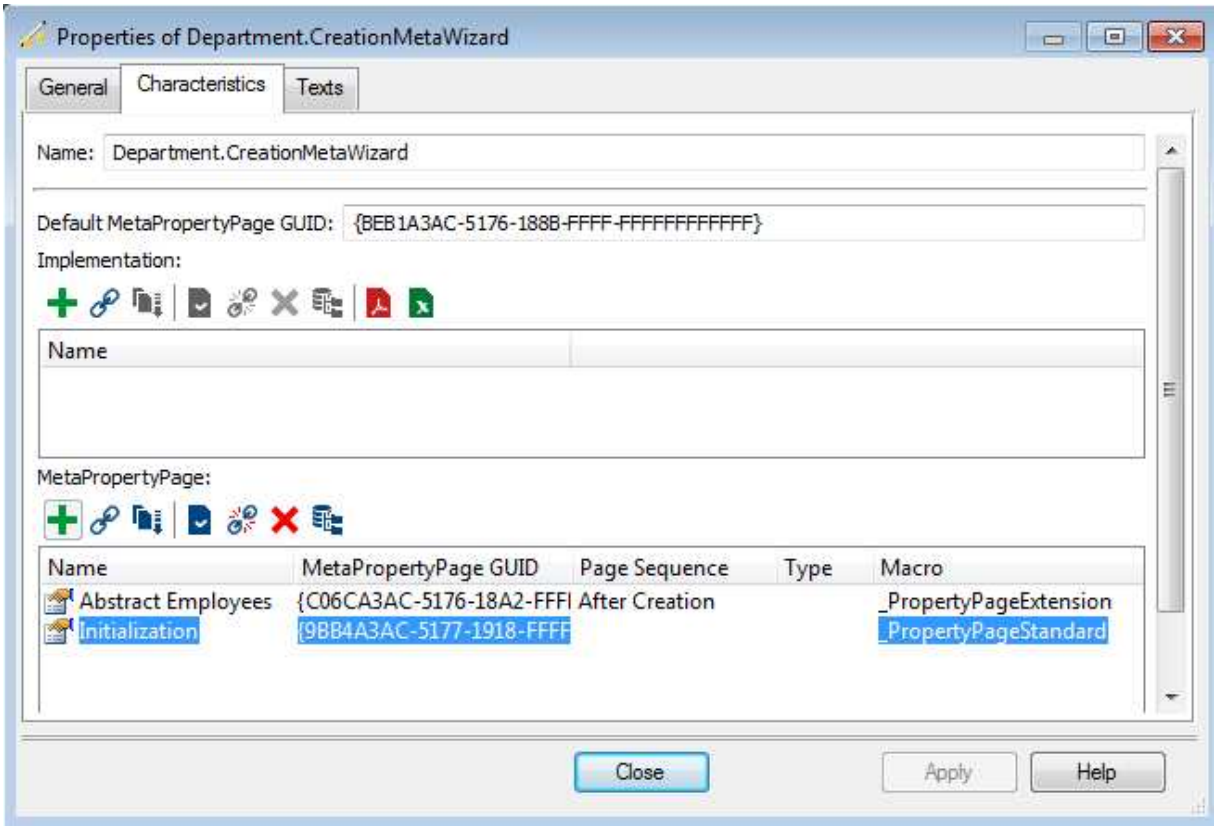
1. Present a page displaying the name (and the owner).
2. Create the occurrence of a Department.
3. Propose the "Abstract Employees" entry page, without the owner.

To do this:

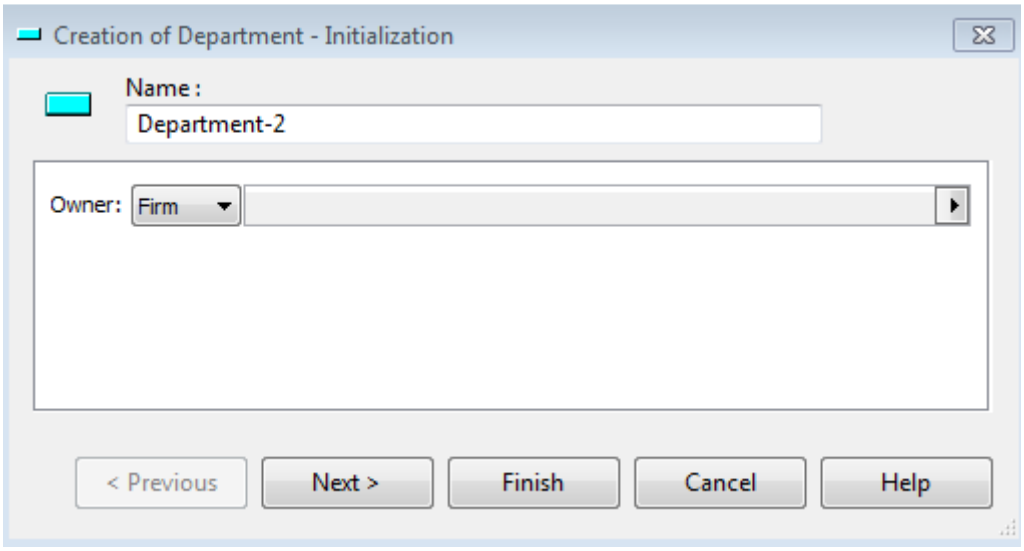
1. Create a new page "Initialization" in the wizard.
2. In the **Macro** field, select "_PropertyPageStandard" to propose the entry of the owner.

3. For the "Abstract Employees" page, in the **Page Sequence** field, select "After Creation" and in the **Macro** field, select "_PropertyPageExtension" so that owner entry is not proposed.

   The result should be as follows:

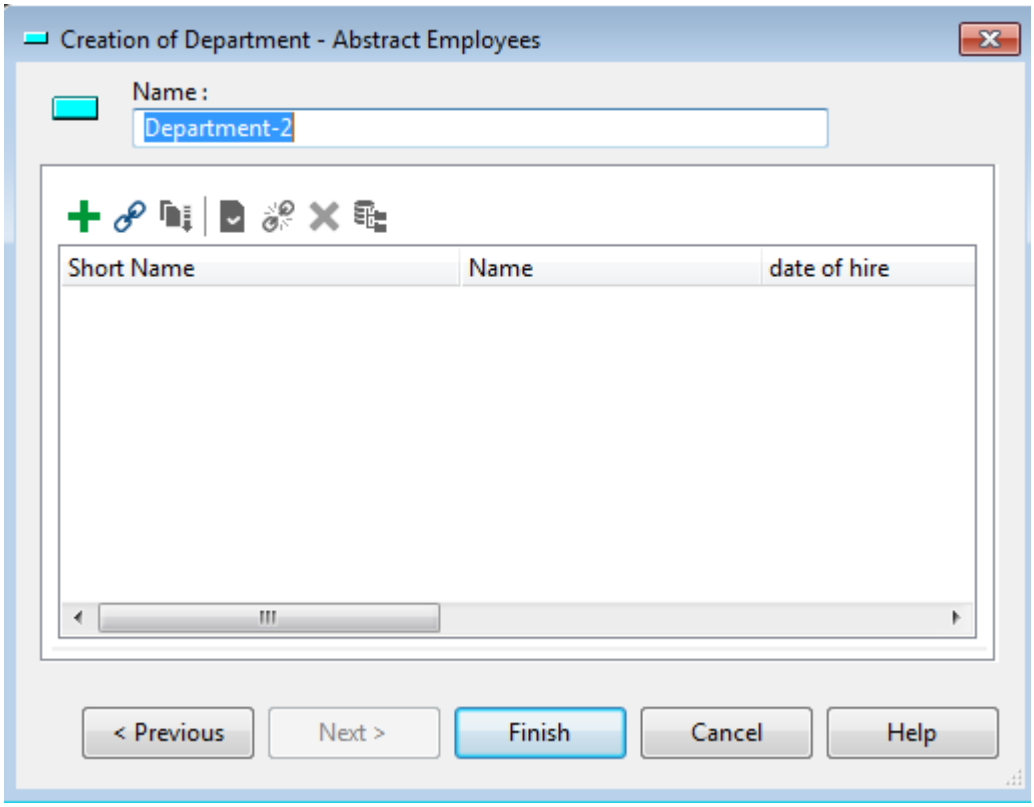


4. Start the the wizard: from the explorer, create a **Department**.

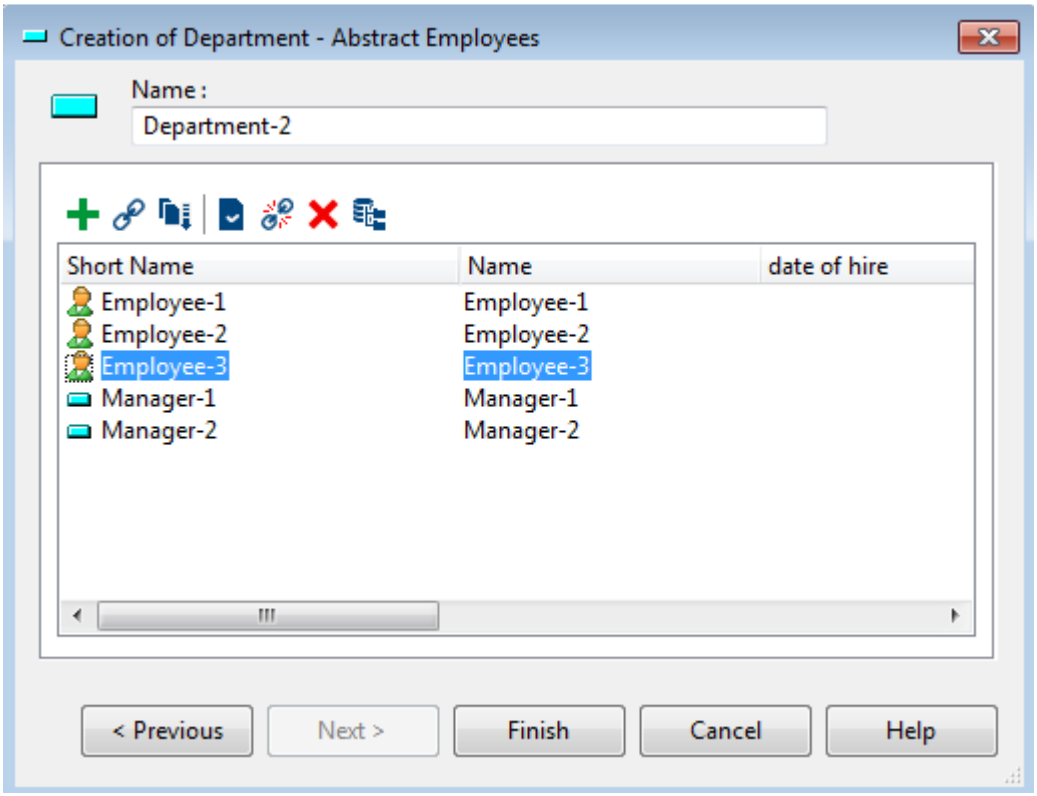


Note that it now presents two pages (the **Next** button is active).

5. Click **Next**.



6. Click **New** or **Connect** to add Employees and/or Managers to "Department-2".



- "Department-2" is created in the transition between the first and second pages.
- Pages appear automatically in the correct order, although you did not order them in the list. The "Abstract Employees" page defined as "After Creation" appears after the "Initialization" page.

Two pages appearing in the same sequence must necessarily be ordered.

This result can be obtained more simply, by configuring the wizard itself, see Calling a wizard by specifying a context p. 26.

Calling a wizard by specifying a context

The context of the wizard, mentioned above, enables extremely powerful external configuration of a wizard. This enables:

- modification of appearance and behavior of a wizard without modifying the wizard.

- design of specific context modes enabling wizards to adapt to use cases.

To do this, we shall consider the functionality of configuring and starting a wizard.

Calling a wizard - configuring properties of the object to be created

Calling a wizard is by means of the *instanceCreator* function accessible from a *MegaCollection*.

The *MegaCollection* used should enable creation of a new object within itself; it therefore consists of collections based on a *MetaClass* or on a *MetaAssociationEnd* seen from a *MetaClass*:

- in the first case, the wizard creates an isolated object,
- in the second, the wizard creates the object seen from the cited association.

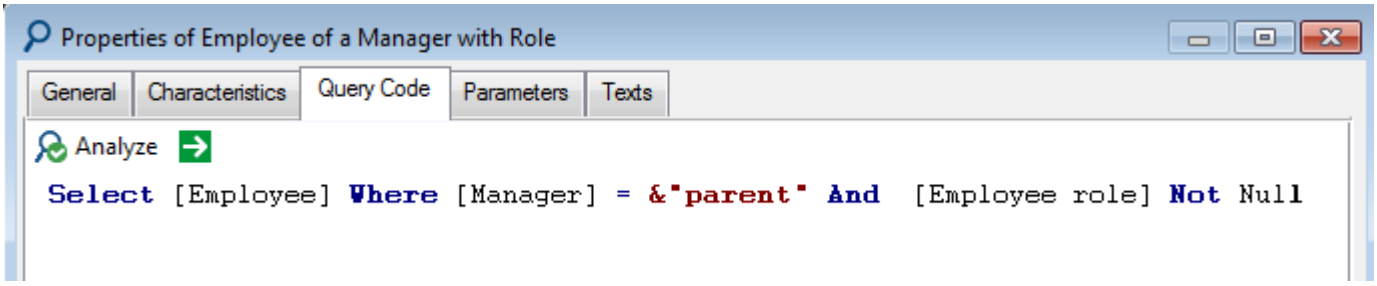
The *instanceCreator* function makes available a *MegaInstanceCreator* component which enables more specific configuration of the wizard and its starting.

The MegaInstanceCreator component implements the functions of MegaWizardContext, and therefore authorizes configuration of the wizard. In addition it implements the *create* function, which enables starting of the wizard.

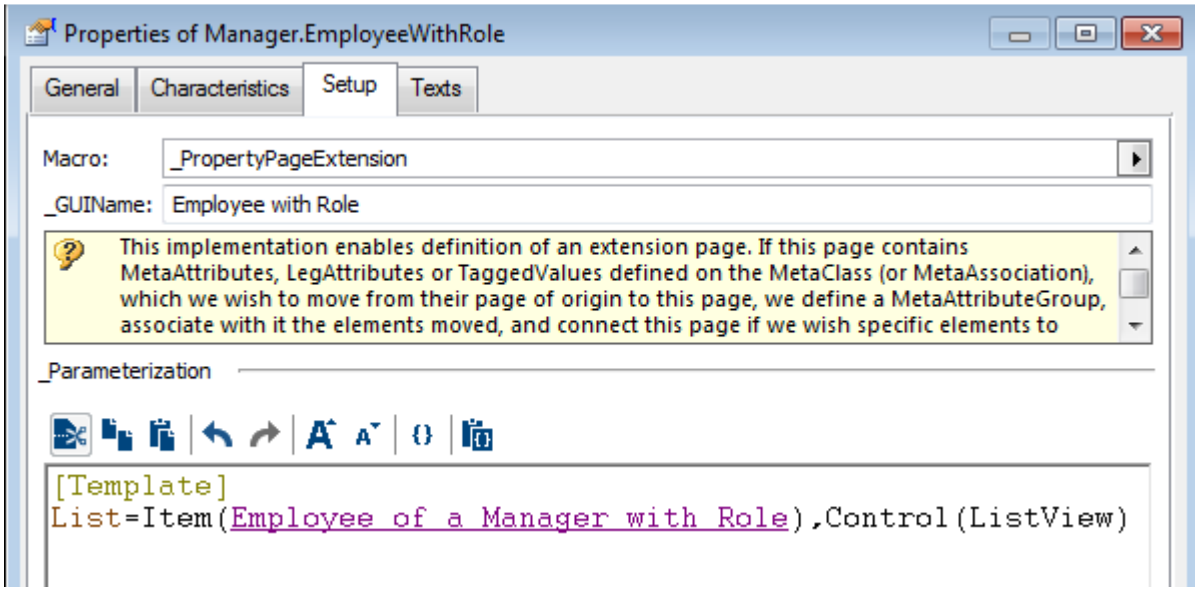
Consider again the example of creation of a **Manager**, now with the following objective:

1. Create a query "Employee of a Manager with Role" defined on the **Manager** MetaClass , listing all **Employees** for which an **Employee Role** has been defined.

```
« Select [Employee] Where [Manager] = &"parent" And  [Employee role] Not Null »
```

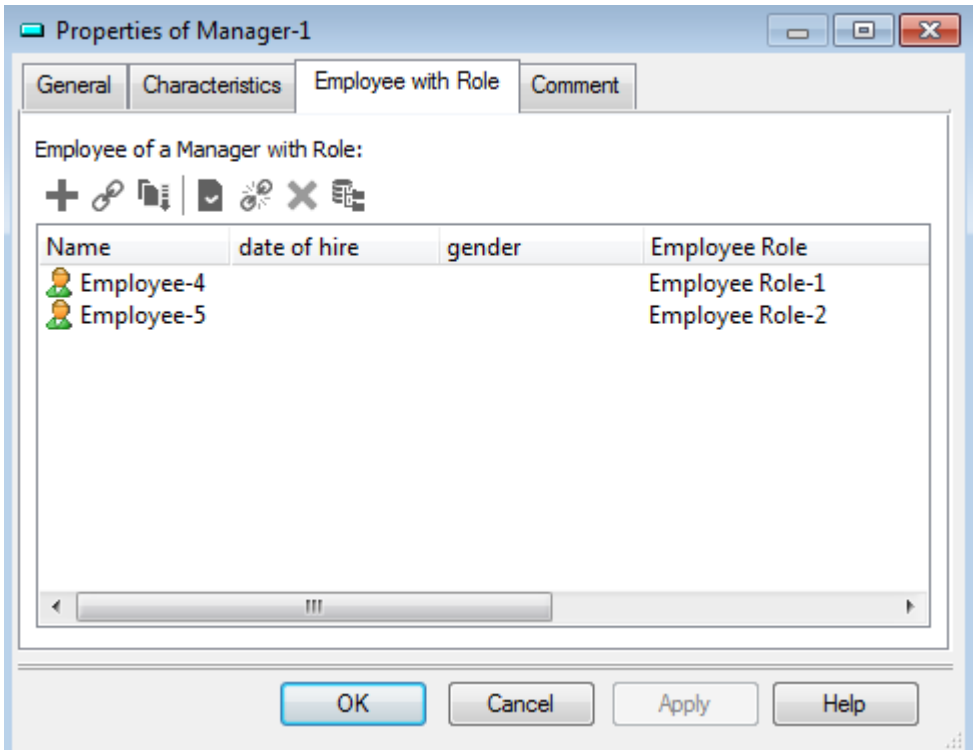


2. From the **Manager** MetaClass, create the PropertyPage "Manager.EmployeeWithRole".



From the properties pages of a Manager, in the "Employee with Role" tab, note that:

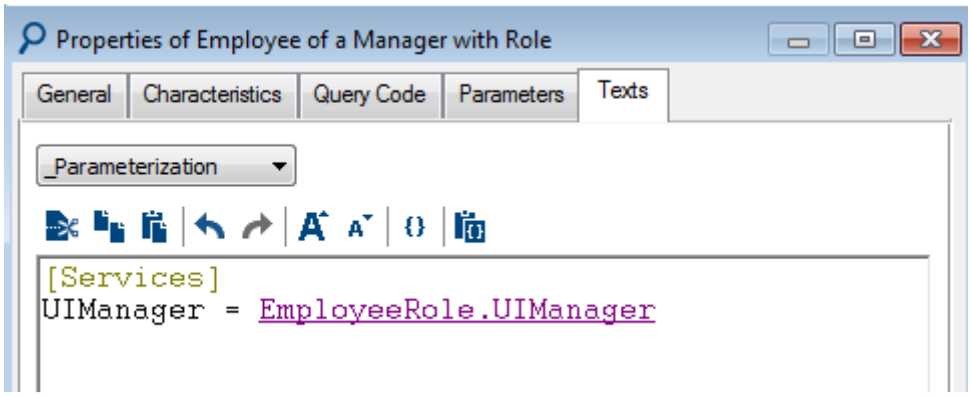
- only the **Employees** for which an **Employee Role** has been defined are listed.
- from this tab, you cannot create an Employee.



- You want to be able to create an **Employee** conforming to this query, by implementing a user interface manager implementing the 'create' method accessible from the query, when for example it is presented in a tree or in a list.

3. The user interface manager is configured in the **Texts** *_Parameterization* tab of the "Employee of a Manager with Role" query in the following form:

```
[Services]
UIManager = EmployeeRole.UIManager
```



Where EmployeeRole.UIManager is the Macro – here in VBScript – implementing the DoCreation function which calls the required creator wizard.

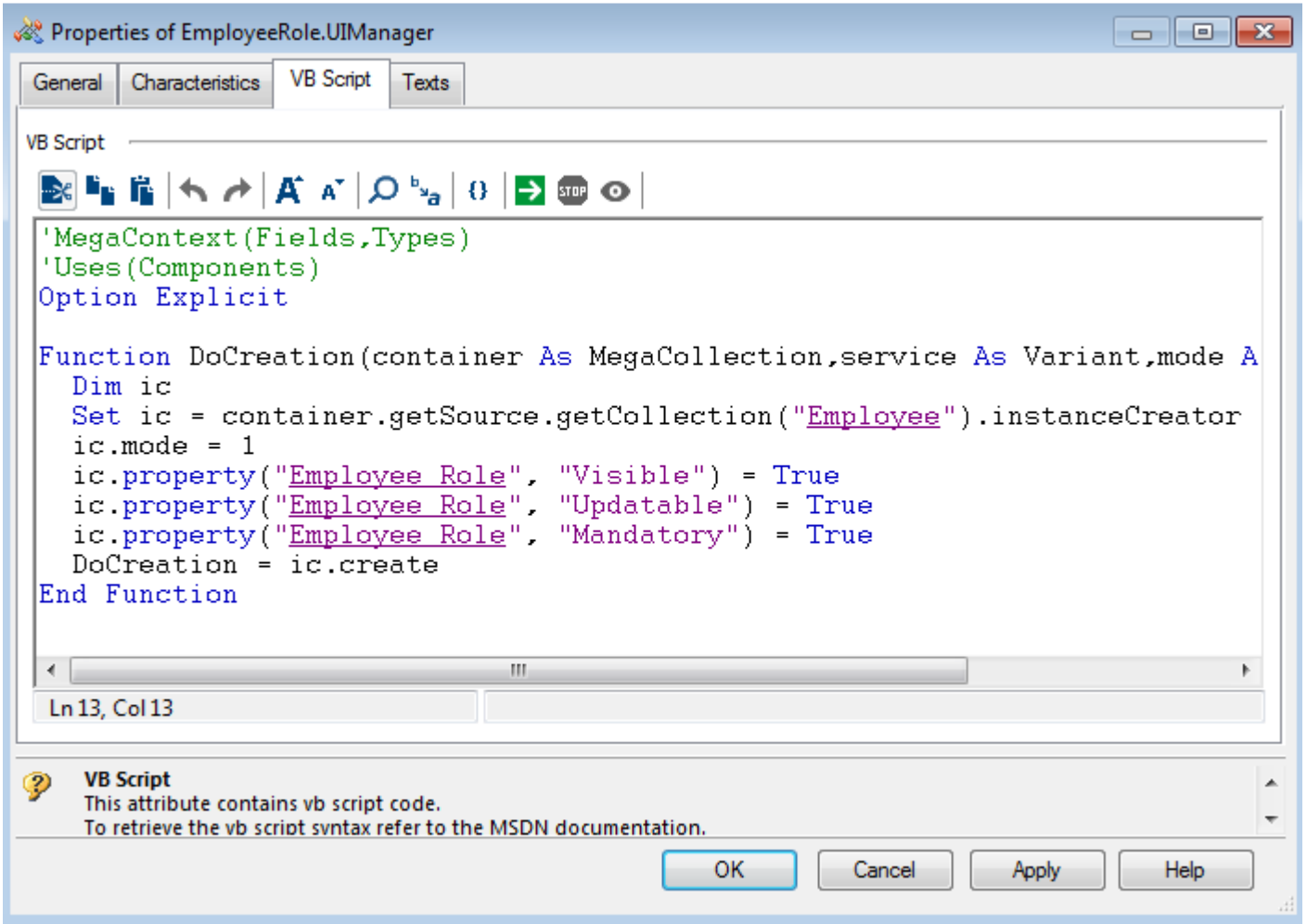
Create the "EmployeeRole.UIManager" macro:

So that the **Employee** created conforms to the query, it must be created from a **Manager** source of the query, and must be associated with an **Employee Role**.
To do this, you have to:

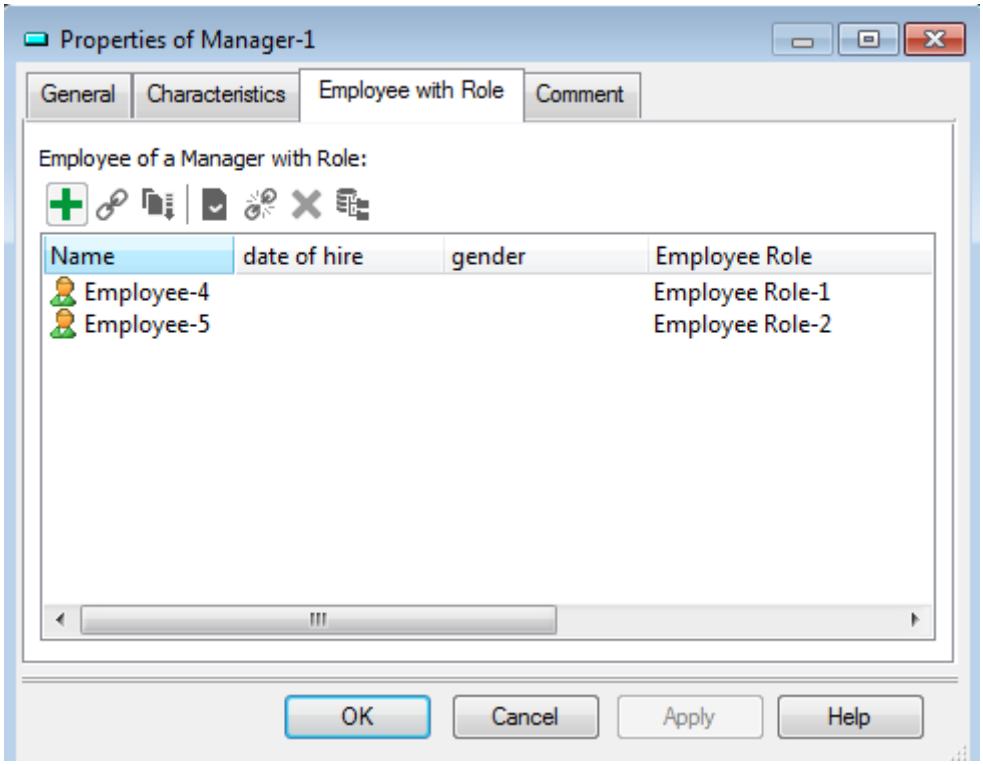
- create an instanceCreator from the collection of **Employees** created from a **Manager** source of the query.
- impose start of the wizard in complete mode by ic.mode = 1
- then configure that the **Employee Role** must be visible and updated, in exactly the same way as in the above customization. The only difference is that here the role is Mandatory.

```
'MegaContext(Fields,Types)
'Uses(Components)
Option Explicit

Function DoCreation(container As MegaCollection,service As Variant,mode As Integer) As Variant
  Dim ic
  Set ic = container.getSource.getCollection("~demSW1OYEDCC[Employee]").instanceCreator
  ic.mode = 1
  ic.property("~y4mZK)OYErIH[Employee Role]", "Visible") = True
  ic.property("~y4mZK)OYErIH[Employee Role]", "Updatable") = True
  ic.property("~y4mZK)OYErIH[Employee Role]", "Mandatory") = True
  DoCreation = ic.create
End Function
```

The function create, and the function DoCreation, return if necessary the identifier of the newly-created object.



From the Properties page of a **Manager**, in the "Employee with Role" tab, note that the **New** button is now accessible.



- You can now from a **Manager** create an **Employee** for which you must specify an **Employee Role**.

This example highlights the fact that it is possible to configure a wizard before calling it. This fact should be noted by the wizard customization designer, who must limit impact of customization to only those elements effectively customized.

It also enables elimination of a possible ambiguity between use of the *Property* function of the context of the wizard, and use of the *GetProp* function of the *template* object of this same wizard. If we wish to initialize a property value for the object to be created, we can consider that the following two functions are possible and therefore redundant:

```
Context.property(propID) = value
Context.template.getProp(propID) = value
```

The *template* object is not available before calling the *create* function: the *property* function is therefore the only one to use if we wish to initialize a property value before starting the wizard. To be more specific:

- Before starting the wizard, the *template* object is not yet created, and use of the *property* function is therefore essential if we wish to consult or update configuration values of the wizard.
- After starting the wizard, and therefore after the *OnWizInitialize* trigger, the *template* object is available; following its creation, property values which may be initialized in the context are assigned to it.
- From the moment the *template* object is available, any modification made from the *property* function is automatically transferred to the template object.
- This system is not however reversible; any direct modifications of the *template* object, for example entries carried out in pages, are not reflected in the corresponding *property* value. During progress of the wizard, this enables determination of whether the current value of *template* has been modified from its initial value.

Specific context elements: cookies

Elements enabling configuration of wizard call context are not necessarily properties of the object.

Certain are sufficiently generic to have been defined directly in the context of creation; in particular this is the case for *sourceID* and *targetID*, which enable indication to the wizard that object creation has been motivated by an action in a diagram (in this case, data contains absolute identifiers of objects to be connected).

However, others that are more specific or less detailed are not anticipated. It is nevertheless possible to carry out configuration of the wizard with such parameters, using cookies.

Like object properties, a configuration cookie is associated with an absolute identifier, and can be consulted or updated by means of the *property* function. A cookie can have any absolute identifier, as long as it does not correspond to the absolute identifier of a property accessible from the object to be created, in which case there is ambiguity.

You can add a cookie to the context of a wizard; to do this, you must define the absolute identifier of this cookie (*cookieID* below), and initialize the property as follows:

```
context.property(cookieID, « Cookie ») = True 'VBScript
context.propertyFlag(cookieID, "Cookie", true);  // Java
```

The "Cookie" flag indicates that the property of the context of the wizard defined does not correspond to a property of the object to be created, but is specific to the context itself.

The context of a wizard can manage two types of cookies; either character strings, or objects. The default type being character strings, we specify that a cookie is of object type by means of the "Object" flag.

The cookies system is used by MEGA tools to enable contextual customization of creation wizards, in particular in diagrams and navigators.

Context cookies of diagrams

When a wizard is called from a diagram, it has the following cookies:

- `~XN(cdzBR8PO0[CookieDiagramNature]` contains the identifier of the *MetaAttributeValue* representing the diagram type.
- `~jL(cb)BR8HP0[CookieDiagramDescribedObject]` contains the identifier of the described object.
- `~bM(cG0CR81Q0[CookieDiagramLegToDescribedObject]` contains the identifier of the *MetaAssociationEnd* to the described object.

Context cookies of navigators

When a wizard is called from a navigator element, it has the following cookies:

- `~aodH6MLoy800[MetaTree]` identifier of *MetaTree* (in the form of a field)
- `~lAhSwEN5)e00[MetaTreeBranch]` identifier of *MetaTreeBranch* (in the form of a field)
- `~u250i2WT(W00[MetaTreeNode]` identifier of *MetaTreeNode* (in the form of a field)

External addition of pages and triggers

You can externally add triggers (in the form of macros) and pages (in the form of *MetaPropertyPages*) to a creation wizard. You can therefore transform a creation wizard into an addition wizard by inserting a suitable preliminary page. These additions should be made before initialization of the wizard, and are therefore available only from the *MegaInstanceCreator* component. This component has the following functions:

- **`addTrigger(TriggerId As Variant,Optional Options As String) :`**
  This method enables insertion of a trigger, of which we define the identifier here. This trigger enables modification of wizard behavior, and in particular the addition of an initialization or processing code. By default, this trigger is called last (that is after the triggers defined for the wizard). However, with the `OnHead` option, it is possible to arrange that this trigger be called first.
- `addPage(PageId As Variant,Optional Options As String):`
  this method enables insertion of an additional page in wizard. PageId is the identifier of the *MetaPropertyPage* that we wish to insert. The option enables specification of the position of the page, and it can take the following values:
  - `Preliminary`: the page is presented in first position. It does not propose entry of name.
  - `Preparatory`: the page is presented after the preliminary pages, but before creation of the object.
  - `Standard`: This is the default; the page is presented after the preliminary and preparatory pages, but before creation of the object. It displays the name of the object if necessary.
  - `AfterCreation`: the page is presented after creation of the object.
  - `Conclusive`: the page is presented in last position.

You can add several pages. If they are of the same type, they will be proposed in the order in which they are added.

Creation wizard static configuration

Certain customization elements of a creation wizard can be specified independently of the triggers and pages which have been associated with it. These elements appear in the *_Parameterization* text of *Meta Wizard*.

Default page configuration

Rather than inserting a specific page, it is often wiser and preferable to configure the wizard default page; in fact this page is likely to appear as soon as properties are added (via the metamodel or external customization), and it is therefore difficult to imagine that it will never appear.

The following configurations have been defined for this:

```
[CreatorWizard]
ForceDefaultPage = 1
```

This configuration forces display of the default page, even if not necessary. In this way, you can for example simplify the activity creation wizard customization example mentioned above, by avoiding creation of the "Initialization" page.

`[Template]`, `[Page]`: these two sections, defined for configuration of a properties page, can appear in wizard configuration and enable definition in complete freedom of the default page of the creation wizard, knowing that properties defined externally will be naturally inserted.

Wizard execution configuration

The following configuration:

```
[Wizard]
CloseAfterTerminate = 1
```

arranges that the wizard window closes after execution of *OnTerminate* triggers; it can be used when particularly lengthy processing is being carried out, associating this with a gauge device.

Wizard execution check

In addition to the possibility of dynamically configuring a wizard and carrying out specific processing at transitions, you can modify behavior of a wizard by means of triggers

- by redefining conditions and if necessary the display order of wizard pages.
- by causing its stop, either by discard or by reuse.

Filtering and conditioning of wizard page sequencing

Wizard pages, like *MetaPropertyPages* of properties pages, can be statically filtered. Page display conditions, and in particular filtering carried out in the *[Filter]* section of its configuration, must be respected for the page to appear in the wizard.

You can however dynamically modify this filtering by implementing the *WizPageCheck* function in a trigger, and by acting particularly on the *result* input/output argument.

This function can be used if you wish to modify appearance of wizard buttons and conditions of page display. It can be called several times according to requirements induced by wizard processing logic; in particular it is called at each transition leading to a page.

The *result* parameter is a bit field enabling definition of button appearance and page visibility; bits used are the following:

- bit 1 (value 1): activated when the 'Finish' button is active
- bit 2 (value 2): activated when the 'Previous' button is active
- bit 3 (value 4): activated when the 'Next' button is active
- bit 5 (value 16): activated when the page must be hidden.

When calling this function, button bits (1, 2, and 3) have a value calculated by wizard logic, which in particular deduces that if we are on the first page, the Previous button is inactive, and if we are on the last page it is the Next button which is grayed. If a page declared as mandatory has not yet been displayed, the Finish button cannot be selected.

The *WizPageCheck* function enables overload of these values and also indication that the page should not be displayed on activating bit 5. Consistency of modifications is not checked, and it is therefore possible to activate the Previous button, even if we are on the first page (in this case the press action will have no effect); it is also possible, and here the consequences are more serious, to mask a mandatory page; in this case wizard logic cannot validate the 'Finish' button (since a mandatory page has not been

displayed), and the trigger must itself manage this button if we wish to allow the wizard to terminate satisfactorily.

When several triggers implement the *WizPageCheck* function, each trigger receives as input value the result value of the previous trigger.

When a page has been masked, wizard logic carries out the same processing on the next page (if it exists).

Identifying wizard pages

In functions of a trigger referencing properties pages of the wizard, these are represented by objects of *MegaPropertyPage* type. In this component, pages can be identified by using the *GetID* function, which returns a string corresponding to a GUID. In the case of standard pages, this GUID (Global Unique IDentifier) is calculated from the absolute identifier for specific pages, and from that of the *Meta Wizard* for the standard page.la *MetaPropertyPage*

The following script code obtains the GUIDs of pages of a *Meta Wizard*:

```
Function GUIDFromMegaID(mObject)
  Dim xID
  xID = mObject.GetProp("_hexaidabs")
  GUID =  & mid(xID,5,4) & left(xID,4) & "-" & mid(xID,9,4) & "-" & right(xID,4)
  GUIDFromMegaID = "{" & GUID & "-FFFF-FFFFFFFFFFFF}"
End Function

Dim myCol
Set myCol = GetCollection("Meta Wizard").SelectQuery

Dim wiz
For each wiz in myCol
  print "Identifier of pages of wizard " & wiz.Name
  print "    " & GUIDFromMegaID(wiz) & " : Page Standard"
  Dim page
  for each page in wiz.MetaPropertyPage
    print "    " &  GUIDFromMegaID(page) & " : Page " & page.Name & "(" & page.MegaField & ")"
  next
Next
```

Modifying page sequencing

The page display order of a wizard is statically defined in the link between the *MetaWizard* and the *MetaPropertyPage*, using the *Order* and *Sequence* attributes. We can however imagine a wizard presenting pages in a different order depending on execution context, or one which requires looping on pages. In this case, it is possible to redefine succession of pages using functions **WizPageNextChange** and **WizPagePreviousChange**

Check before termination

A trigger can implement a specific check on pressing the 'Finish' button, in particular to prevent the action if a condition has not been satisfied. This check must be implemented in the *OnWizBeforeTerminate* function; the value of *result* must be set to "1" to prevent execution of the code terminating the wizard.

Reuse mode: implementing an addition wizard

You can transform a creation wizard into an addition wizard. In this case, the object returned by the wizard may already exist. When the occurrence to be reused has been defined, wizard logic should not try to create an object, and the wizard must be able to terminate without an additional transition.
This behavior is permitted by means of the *reusedID* function of the *MegaWizardContext* component. In assigning the identifier of an object by means of this function, we cause exit from the wizard after the current transition, and return by the wizard of the identifier specified.

Dynamic customization of the wizard user interface

You can dynamically customize the wizard user interface by means of the following functions of *MegaWizardContext*:

wizardCaption and addingCaption

This function enables redefinition of the creation wizard header; when you wish to transform a creation wizard into an addition wizard , we use the `addingCaption` function.

mainHelpTitle, mainHelpBody and hideMainHelp

These functions enable redefinition of the title and content of the help area displayed in the wizard default page. This help area can also be hidden by `hideMainHelp`.

Wizard specific use and display data

In examples up to this point, pages display data relating to MEGA objects which exist or will exist; this data is 'justified' or 'mapped' by elements of the metamodel.

During creation of a wizard however, you are quickly led to display and enter elements that do not correspond to MEGA data, but to data local to the wizard. This data can be assimilated in variables.

Wizards can handle such variables by means of cookies. The possibility of displaying cookies in wizard pages is not however envisaged for the moment.

Defining a collaboration between the wizard and its pages

As long as data displayed in wizard pages corresponds to objects described in the metamodel, an implicit collaboration can be included via the *template* object; this object can be handled in the properties pages like a standard MEGA object, and can be accessed and handled in the code of the wizard and its triggers. However, to display data specific to the wizard, we must define a collaboration.
From the viewpoint of the properties page, this collaboration is necessary to indicate that the data it will display does not come from the *template* object but from another data source.

This other data source is defined by means of an informal query, which allows definition of the description of the collaboration of the wizard.
This collaboration must be declared in the wizard configuration text as follows:

[Wizard]

Description = <ID of the query>

On completion of this declaration, the properties and collections declared in the collaboration are accessible from the context of the wizard via the MegaObject "CookieObject". Note that these properties and collections are also accessible as a cookie.

To initialize or modify a property of the collaboration, you can therefore choose from:

- `context.property(cookieID) = <value>`
- `context.cookieObject.getProp(cookieID) = <value>`

To initialize a collection, you must use:

- `context.property(cookieID) = <Collection>`

  When the collection is not initialized in this way, it cannot be used, and the call to GetCollection produces an error "No collection associated to this cookie". In addition, you cannot modify this collection after initialization.

Declaration of this collaboration produces automatic injection of the "Context" map in all wizard properties pages.

- You can therefore include a property or collection of the collaboration in a properties page of the wizard by declaring the element in this map by means of keyword `From(Context)`.

{compatibility} Declaration of the description in configuration of the wizard avoids:

- use of function context.cookieDescription, which is therefore obsolete,
- explicit declaration of the collaboration map in the [template] of the properties page

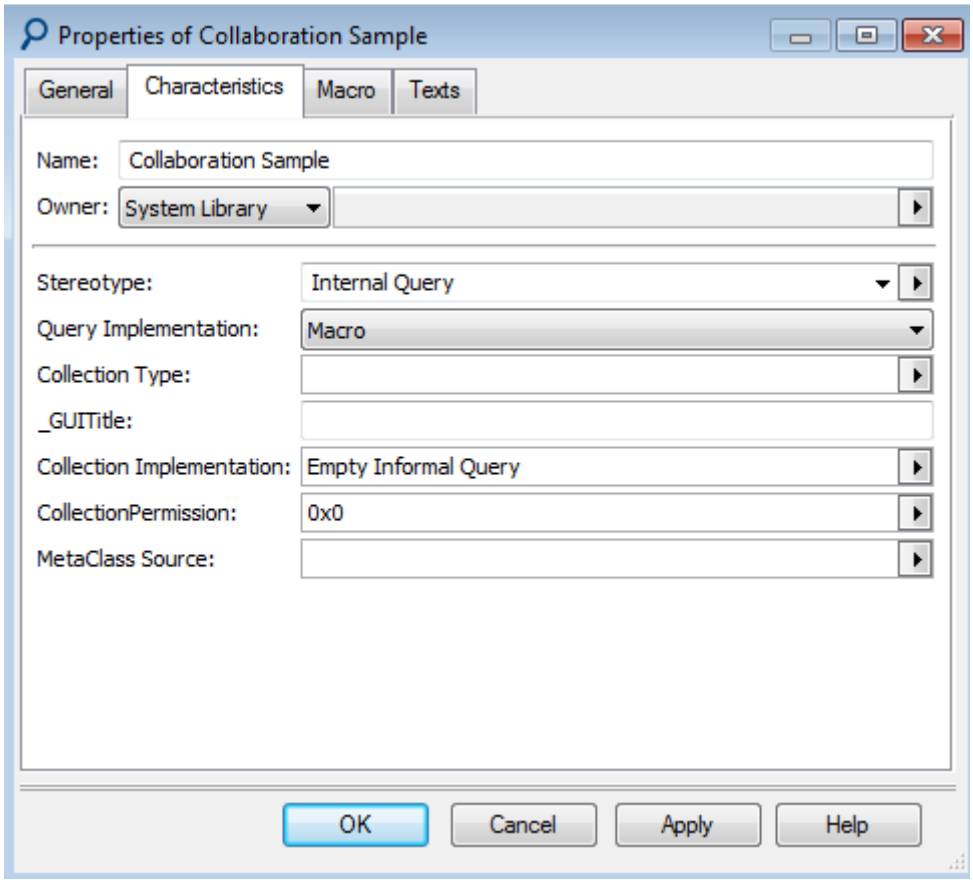
  ```
  Cookies=Map(<Description>,Cookie(~DutIllKV5X40[Meta Wizard])
  ```

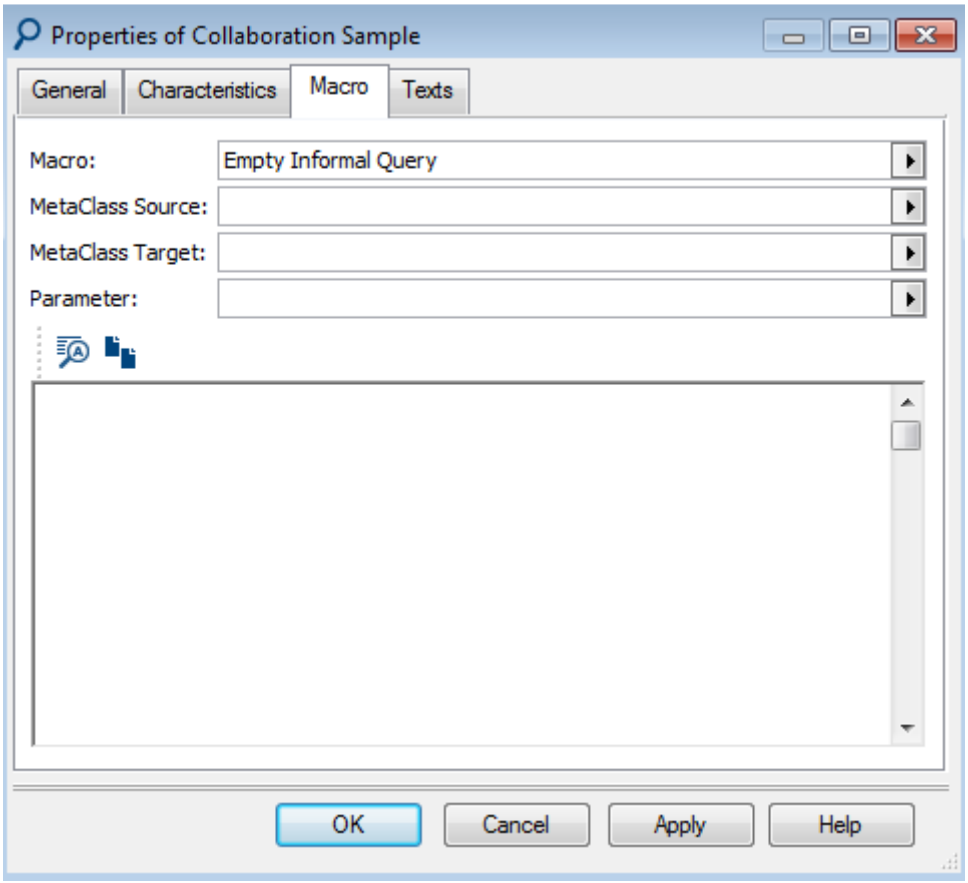
  this map **Cookies** being replaced by the implicit map **Context**.

Although possible technically, the list of cookies does not naturally correspond to the list of properties defined for a *MetaClass*, *MetaAssociation* or *MetaAssociationEnd*. Nor is it reasonable to create such concepts in the metamodel with the sole aim of defining collaboration of a wizard.
To do this, we favor use of an informal query as collaboration of a wizard.

Defining an informal query

Informal queries are queries not associated with a target MetaClass; they cannot therefore be used in the standard query tool. They can however be associated with a macro which implements their supply.
These queries are mainly used by script tools, and enable definition of collections of informal objects used as *MegaObjects* within a *MegaCollection*.
To create an informal query, you must create a query of 'Macro' type and associate it with the "Empty Informal Query" macro.


Properties of Collaboration Sample
General
Characteristics
Macro
Texts
Name:
Collaboration Sample
Owner:
System Library
Stereotype:
Internal Query
Query Implementation:
Macro
Collection Type:
_GUITitle:
Collection Implementation:
Empty Informal Query
CollectionPermission:
0x0
MetaClass Source:
OK
Cancel
Apply
Help

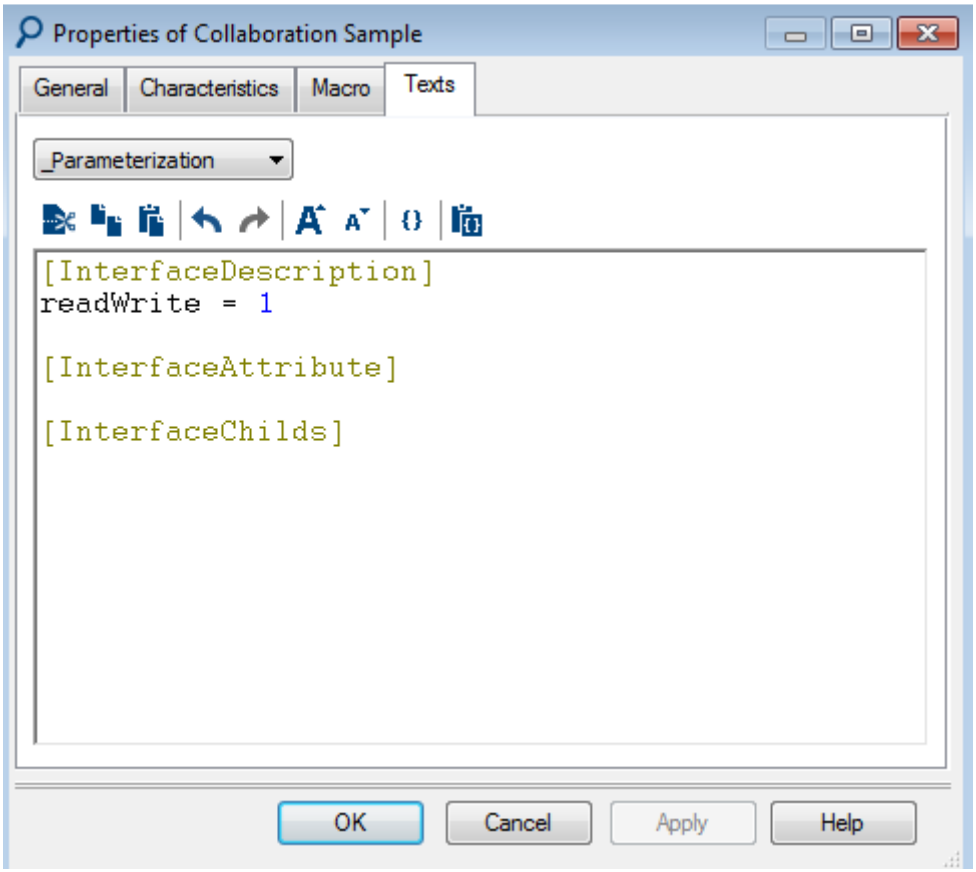


Description of an informal query not being based on a MetaClass, you must define it specifically; no metamodel extension is used for this definition, which uses the _parameterization text of the query.

- The [Interface Description] paragraph enables definition of general data of the informal query, and in particular if it is possible to freely create objects (key ReadWrite = 1). You can in this paragraph redefine the attribute which plays the role of identifier (by default this is the 'Order' Mega attribute). Attributes defined by default for these roles are generally satisfactory and it is not necessary to redefine these.

To add:

- properties to informal queries, insert keys in the [InterfaceAttribute] section.
- collections accessible from objects corresponding to this description, insert keys in the [InterfaceChilds] section.



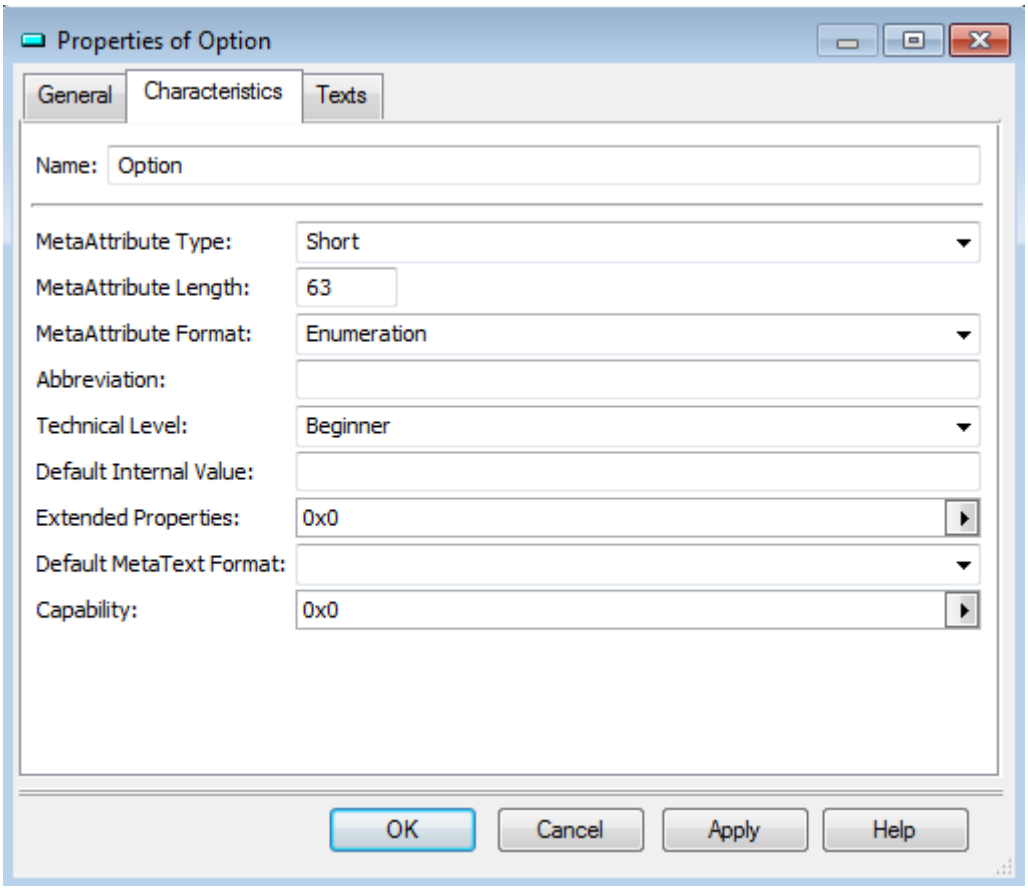
In your « Collaboration sample » informal query example, you shall define:

- a property of list type enabling definition of an action to be selected from 'Option 1' 'Option 2' and 'Option 3'
- a collection of Keywords

The list of *[InterfaceChilds]* can accept *MetaClasses*, *MetaAssociationEnds* or Queries (which can also be informal). The collection of Keywords can appear, either via a query with Keyword target, or via the Keyword MetaClass itself. We choose the latter solution, the first only being useful if we wish to define several collections based on Keyword.

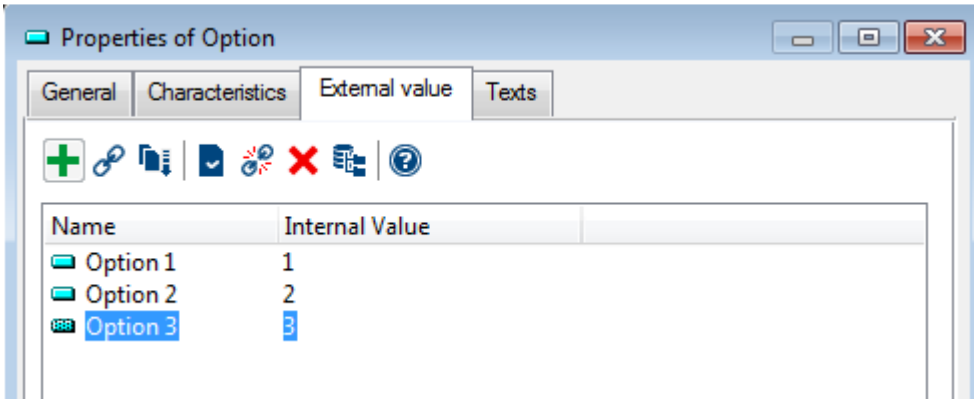
1. Insert:
   `[InterfaceChilds]`
   `~VrUiN9B5iSN0[Keyword] = XREF`
   XREF indicates that we define a reference to a collection definition.

2. To define the requested list, create an "Option" TaggedValue of 'Short' type and 'Enumeration' format.



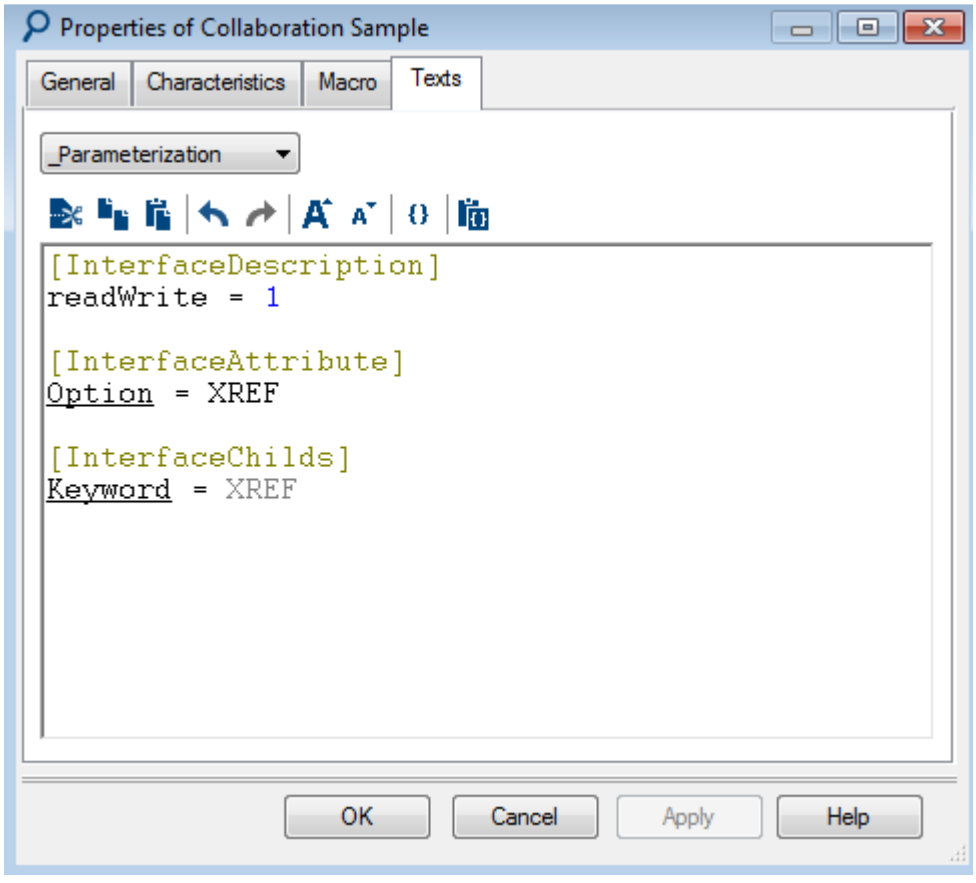
3. Close the properties dialog box of the TaggedValue and reopen it.
   The **External Values** tab appears.

4. On this taggedValue, define 3 tabulated values with internal values 1, 2 and 3 corresponding to the 3 desired options.



5. Insert this *taggedValue* in the list of properties of the "Collaboration Sample" query:

```
[InterfaceAttribute]
Option = XREF
```



In the above trigger you can use the following collaboration cookies:

- Property("<Option>") which will contain a numerical value
- Property("<Keyword>") which will contain a list of keywords

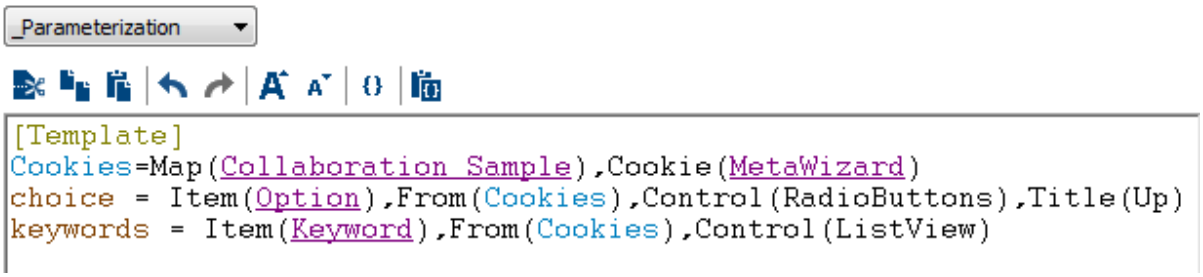
Data between square brackets indicates fields corresponding to identifiers of the objects being handled.

6. You will define a page preparatory to the Employee creation wizard by displaying the data mentioned above.

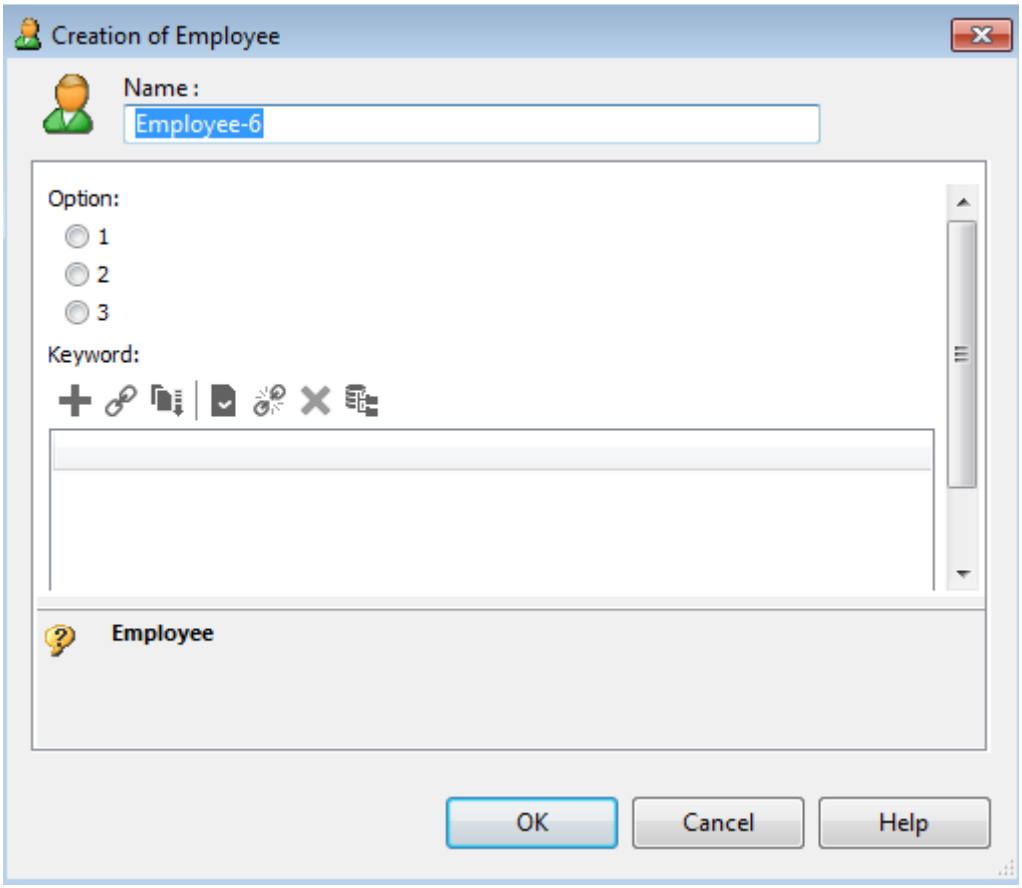
   To do this, insert the following lines in the [Template] paragraph of the "Employee.CreationWizard.Implementation" macro. Remember to associate the trigger macro to the wizard.

```
[Template]
Cookies=Map(<Collaboration Sample>),Cookie(~DutIllKV5X40[Meta Wizard])
choice = Item(<Option>),From(Cookies),Control(RadioButtons),Title(Up)
keywords = Item(<Keyword>),From(Cookies),Control(ListView)
```

Note : <objet> corresponds to IdAbs of Object[Object Name]



7. Create an Employee.



Initializing collaboration cookies

It is essential to initialize collaboration cookies producing collections, as is the case in the previous example for the Keyword collection. When displayed, this page will request the context for the collection of keywords via the <Keyword> property. When this collection has been requested, it will be displayed in the list and no longer requested from the wizard context.

If the context of the wizard wishes to check the content of this cookie, it should be initialized before display of the page, without change of collection; otherwise there will be inconsistency between cookie content and the page list.

In the example above, you can initialize this collection with the following keyword selection:

VBScript :

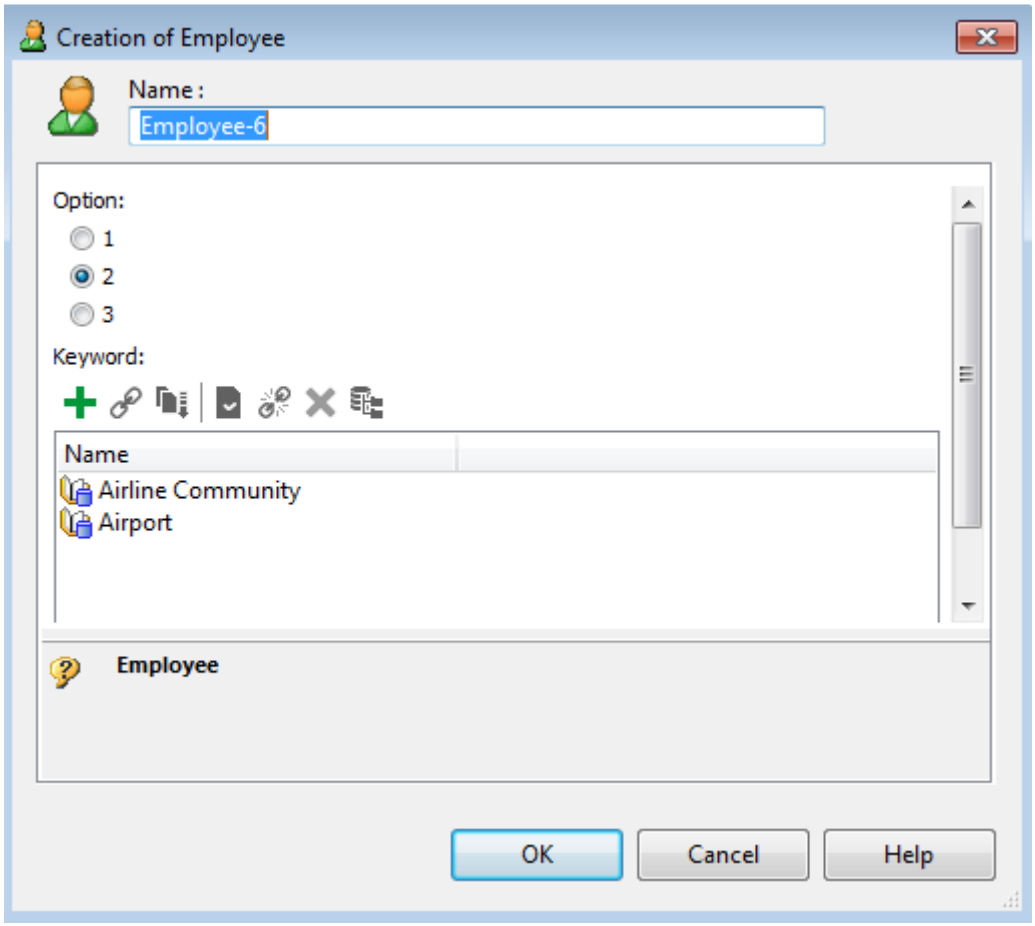
```
Sub OnWizInitialize(oContext As MegaWizardContext)
  oContext.cookieDescription = "<DescriptionId>"
  Dim mcCollect As MegaCollection
  mcCollect = oContext.template.getRoot.getSelection("Select Keyword Where Name
Like 'A#'")
  oContext.property("~VrUiN9B5iSN0[Keyword]") = mcCollect
  oContext.property("<OptionId>") = 2
End Sub
```

```
Sub OnWizInitialize(oContext As MegaWizardContext)
  oContext.cookieDescription = "Collaboration Sample"
    Dim mcCollect As MegaCollection
    mcCollect = oContext.template.getRoot.getSelection("Select keyword Where Name Like 'A#'")
    oContext.property("Keyword") = mcCollect
    oContext.property("Option") = 2
End Sub
```

Java :

```
public void OnWizInitialize(MegaWizardContext oContext) {
  oContext.cookieDescription("<DescriptionId>");
  MegaCollection mcCollect;
  mcCollect = oContext.template().getRoot().getSelection("Select Keyword Where Name
Like 'A#'");
  oContext.property("~VrUiN9B5iSN0[Keyword]", mcCollect);
  oContext.property("<OptionId>", 2);}
```

In the above example, you also initialize the value of the option (<OptionID>) cookie. The page now appears as follows:



In processing of the trigger, if you wish to modify content of this list, we should not change the collection assigned to this cookie; this change will not be taken into account by the properties page, except in the case of global update. It is preferable to modify content of the collection by means of the *insert* and *remove* functions of the *MegaCollection*.

Now having a preliminary page, the creation wizard can condition display of the following pages to the value of the 'option' cookie. Let us consider that the 'Option' induces as the next page a page chosen from three pages with different content. We define a filtering code in the WizPageCheck function, and mask the page if it does not correspond to the value selected in 'Option'.

Wizards on abstract MetaClasses

You cannot create an occurrence of abstract class; you can however define a wizard on an abstract class:

These wizards enable definition of pages and triggers common to all concrete MetaClasses derived from this abstract class.

Pages and triggers defined on the wizard of the abstract class are inserted in respective sequences by consolidation of order numbers to connected wizards (for example, the trigger order number 10 on the wizard of the abstract class will be called before the trigger order number 20 on the wizard of the concrete class).

Where order numbers are equal, the trigger of the concrete class is called last.

Interactive tool based on a MEGA wizard

We shall now broaden our field of study; use of a MEGA wizard is not restricted to customization of object creation, but can assist in implementation of all kinds of interactive tools. The limitation of this use is the contour of what can be managed by a MEGA standard properties page.
Such wizards can be for example tools for modification or transformation of MEGA occurrences, or tools for import or export to third party tools.

Differences from a creation wizard

A MEGA wizard not being a creation wizard, its operation and use are different:

- the *template* available in the context does not represent an object in creation phase.
- there is no default page; configuration relating to this page defined in the *Meta Wizard*, or specified in the context of the wizard, is not taken into account.
- page sequences are not used, in particular those which reference a creation phase.
- the creation phase itself does not exist, and the *OnWizRealize* function of triggers is not called.
- pages do not display by default the name of the object.

Concerning use, calling a wizard is not by means of the *InstanceCreator* function.
In addition it is possible to redefine in greater detail the appearance and *frame* of such a wizard.

system process call

MEGA wizards can be called by means of the *MegaWizard* component.
This component can be created by the *WizardRun* function. This function can be called on any *MegaObject*, providing as parameter the absolute identifier of the *Meta Wizard* to be called. In Java, builders of the Wrapper MegaWizard class calling this function are made available.
The *MegaObject* to which the *WizardRun* function relates, which can be a *MegaRoot*, will be accessible from the context of the wizard via the *template* member, which in this case represents a real object.
The MegaWizard component has the following functions:

- **context**: makes available the *MegaWizardContext* of the wizard. It is therefore possible to configure the wizard before calling it.
- **picture(<PictureID>)**: enables definition of the image displayed in the wizard; this image can be defined in wizard configuration.
- **run**: this function calls the wizard.

Configuring a wizard

A generic wizard is modeled in MEGA by a *Meta Wizard*; however, this *Meta Wizard* is not associated with a *MetaClass*, as a creation wizard could be.
Static configuration of these wizards (defined in the *_Parameterization* text of the *Meta Wizard*) has been enhanced; in particular, it is possible for these wizards to define an image displayed in their *frame*, while the creation wizard, for reasons of consistency, simply displayed the icon of the MetaClass to be created.
The parameters introduced are:

```
[WizardFrame]
      BitmapStyle = 1 or 2
            Indicates that the image is displayed at top (1) or on left (2).
      BitmapHorizontalMargin = n
            Indicates in pixels the horizontal margin of the bitmap.
      BitmapVerticalMargin = n
            Indicates in pixels the vertical margin of the bitmap.
```

`BitmapWidth = n`
Indicates in pixels the width of the bitmap.

`BitmapHeight = n`
Indicates in pixels the height of the bitmap.

`BitmapName = name`
Specifies the name of the file for the bitmap. This file should be located in one of the MEGA configuration files (Mega_Std or Mega_Usr). The .bmp file extension should not appear.

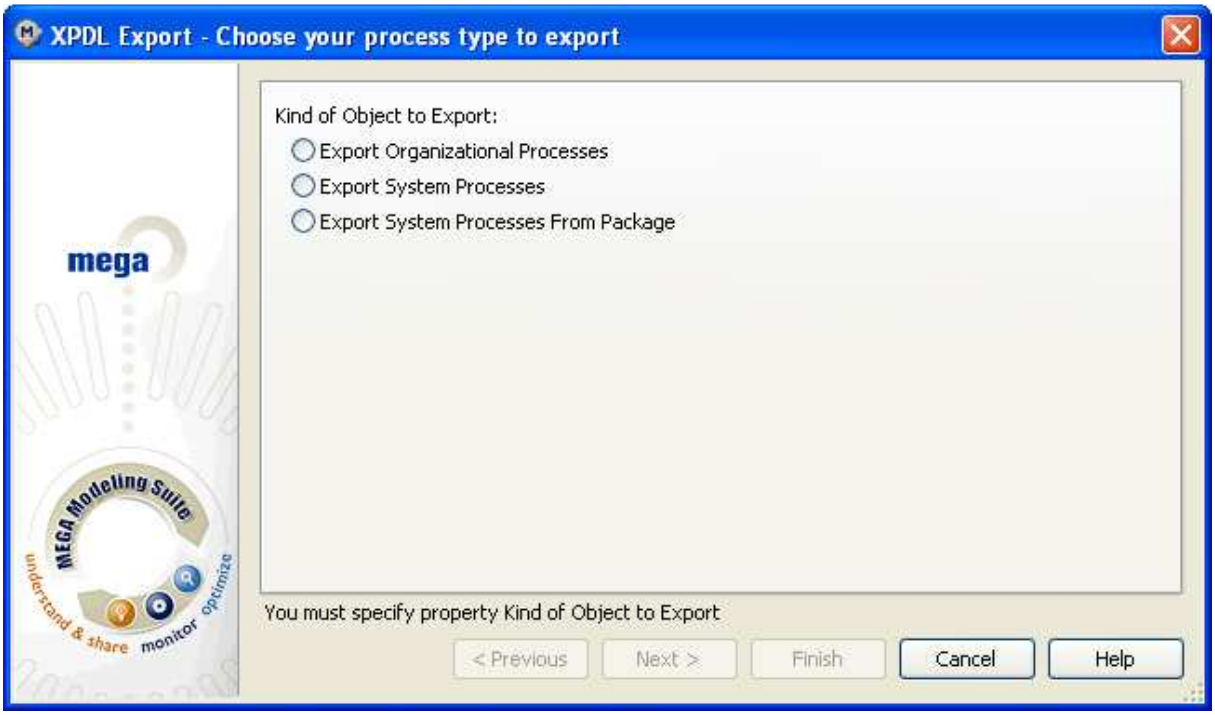
It is possible to redefine the icon of the wizard by specifying the picture in:

```
[Wizard]
Picture=<identifier>
```

Configuration example: export wizard XPDL BPMN

```
[WizardFrame]
BitmapStyle = 2
BitmapName = processWizard
BitmapWidth = 121
BitmapHorizontalMargin = 0
BitmapVerticalMargin = 0

[Wizard]
Picture = ~sj4c8ZPf2n40[BaseExport]
```

The configured frame becomes:


XPDL Export - Choose your process type to export
Kind of Object to Export:
Export Organizational Processes
Export System Processes
Export System Processes From Package
mega
MEGA Modeling Suite
understand & share
monitor
optimize
You must specify property Kind of Object to Export
< Previous
Next >
Finish
Cancel
Help

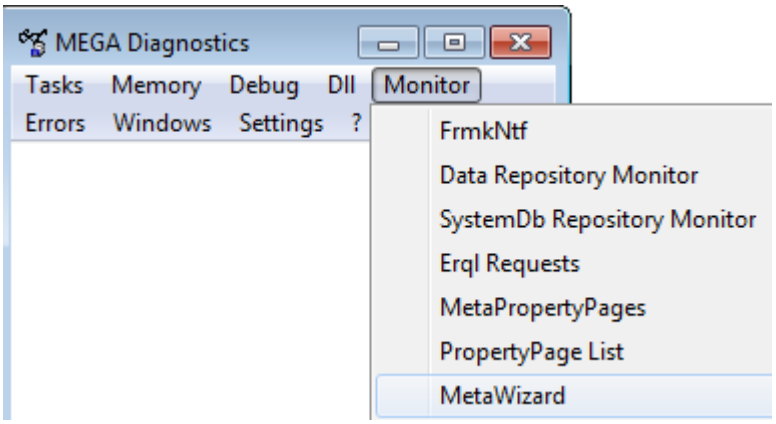
Appendices

Wizard execution tracking

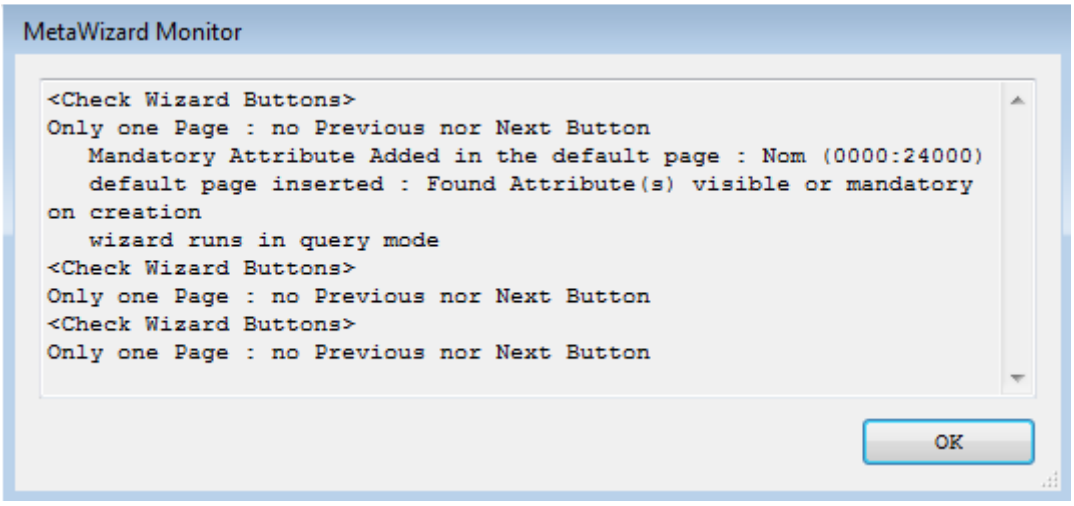
A utility enabling tracking of execution logic of a wizard is available. To activate this, the MEGA diagnostics window should be opened. This can be done using the following line of script:

```
currentEnvironment.site.toolkit.createMegaObject("Mapp.Diagnostics").open
```

In this window, the MetaWizard sub-menu is available in the Monitor menu when a wizard is started in MEGA. We must therefore start a wizard before selecting the menu.

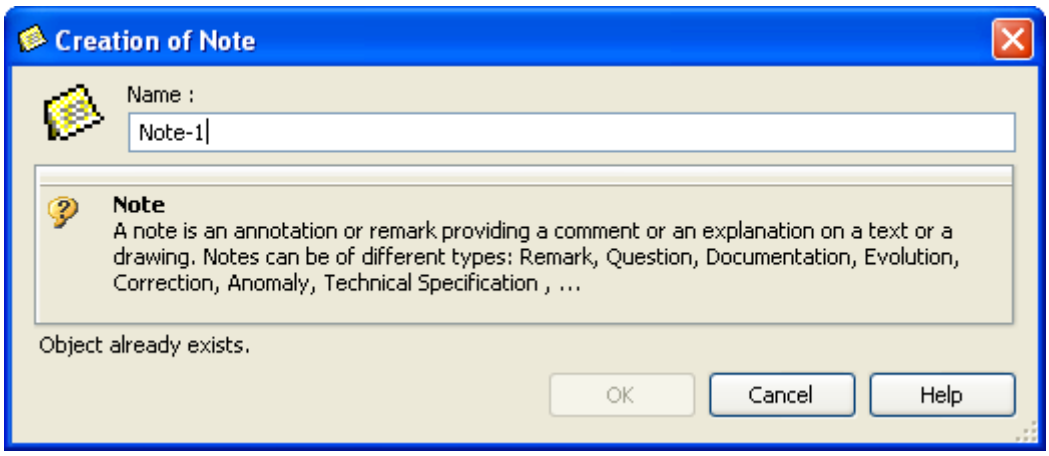


This command opens a window displaying elements relating to execution of wizards.



Wizard button dynamic activation

In creation wizards, the name edit area is treated in a specific way: besides its positioning, outside the page area, the content of the area acts dynamically on activation of buttons **OK** or **Finish**, in particular when the name is not specified, or when the name is already assigned to another object; in this case, a message of explanation is displayed: "This object already exists".



However, default behavior of the wizard is such that other areas are only checked at a transition; the transition is refused if all areas have not been validated:

- all mandatory areas must be specified.
- all specified values must be valid.

Additional checks can be defined:

- by specifying a mandatory element in the properties page:
  `myItem = Item(...)...,Mandatory(True)`
- or by implementing these in a trigger, in the *OnWizBeforeTerminate* function. This function being called when pressing the "Finish" button (or "OK" if there is only one page in the wizard), it enables prevention of validation of the wizard. It is compulsory in this case to inform the user, either by displaying a message box, or preferably by explicitly specifying the message area as follows:

In VBScript:

```
Page.Component.StatusMessage = "Message"
```

In Java :

```
page.component().toMegaPropertyPageStandard().setStatusMessage("Message");
```

This check should not however be implemented in the *WizPageCheck* function when 'Previous' and 'Next' buttons are not active; this function being called at display of the page, the button cannot be reactivated and it will not be possible to terminate the wizard.

This behavior is functionally satisfactory, but you can arrange that behavior similar to that of the name area can be applied to other areas of the wizard, that is activation of buttons can evolve dynamically, according to modifications carried out in the page.

To do this, you must configure the page so as to activate instantaneous check; this can be done at initialization of the page, in the *OnWizPageInitialize* function.

VBScript :

```
Sub OnWizPageInitialize(Manager As MegaWizardContext,Page As MegaPropertyPage)
 Page.Component.ActiveValidation = True
End Sub
```

Java :

```
public void OnWizPageInitialize(MegaWizardContext wctx,MegaPropertyPage mpp){
  MegaPropertyPageStandard mpps = mpp.component().toMegaPropertyPageStandard();
  mpps.setActiveValidation(true);
}
```

You can also activate customized checks by establishing a connection in the properties page; this connection enables a specific code to be activated on each user action in the page. This connection can also be activated at page initialization; in Java, the specific code is implemented via the interface `MegaPropertyPageStandard.PageConnectionPoint`

```
mpp.connect(this); // the wizard class implements the PageConnectionPoint interface
```

The *PageConnectionPoint* interface comprises the function:

```
public void OnCommand(MegaPropertyPageStandard page, int notif, int control)
```

Notifications received depend on the elements contained in the page and are not the subject of a detailed description here. The example below enables tracking of events received in the connected pages:

```
public void OnCommand(MegaPropertyPageStandard page, int notif, int control) {
  String itemName = page.currentControl().getProp("ItemName");
  page.getRoot().print("OnCommand:" + itemName + " (" + notif + ")");
}
```

(To view displays controlled by the 'print' function we must activate tracking of macros by menu "Monitor>Mega Scripts Output" of the MEGA diagnostics window mentioned above).

Versatile Desktop

Desktop Creation and Customization

1. INTRODUCTION

1.1 Aim

The **Versatile Desktop** functionality enables creation and customization of the work environment of MEGA users. A user can have different desktops adapted to his/her requirements and to the actions he/she must execute as a function of a given role.

1.2 Public concerned

Desktop creation and definition is intended for:

- Functional developers, for MEGA desktops supplied as standard
- Product Engineers or Administrators, for customized desktops for client accounts.

2 OVERVIEW

2.1 Diagram

A user can connect to MEGA Applications via customized desktops according to the actions to be executed.

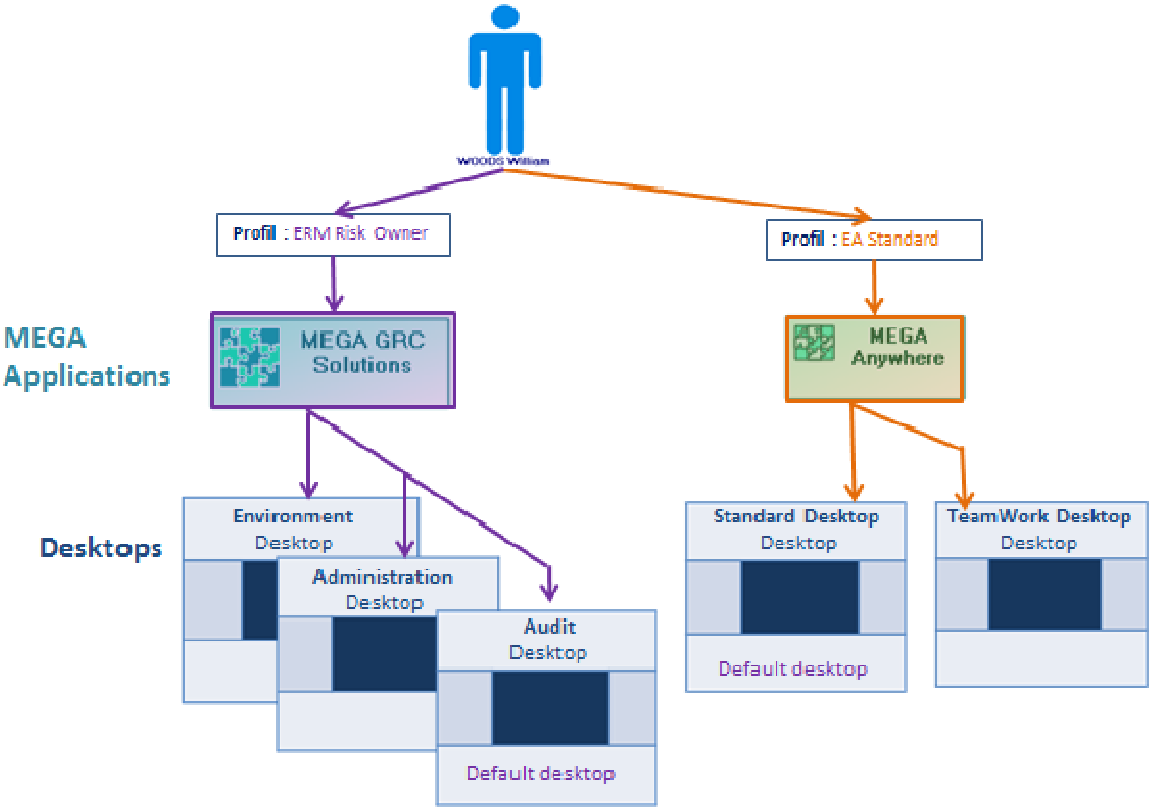


Figure 1 : Overview schema

The properties defined on each desktop of a MEGA Application define:

- the session access mode for this desktop:
  - **Read-only workspace**: the application opens in read-only mode in the current state. Updates are not allowed.
  - **Public workspace**: the application opens in the current state and data can be modified. All updates are visible by all users using the application at the same time.
  - **Private workspace**: the application opens in the current state and data can be modified. All updates made by the user are kept in the private space of the user until he/she decides to dispatch them.
- (when the session access mode is **Public workspace**) the session connection mode for this desktop:
  - **Single session**: end users do not share the same process. They might not have the same view of the repository.
  - **Multi-session**: end users share the same process. They must have the same view of the repository.

Note: **workspace** is the new wording for **transaction**.

2.2 Elements of a work environment

To customize the work environment of users, the following elements are available:

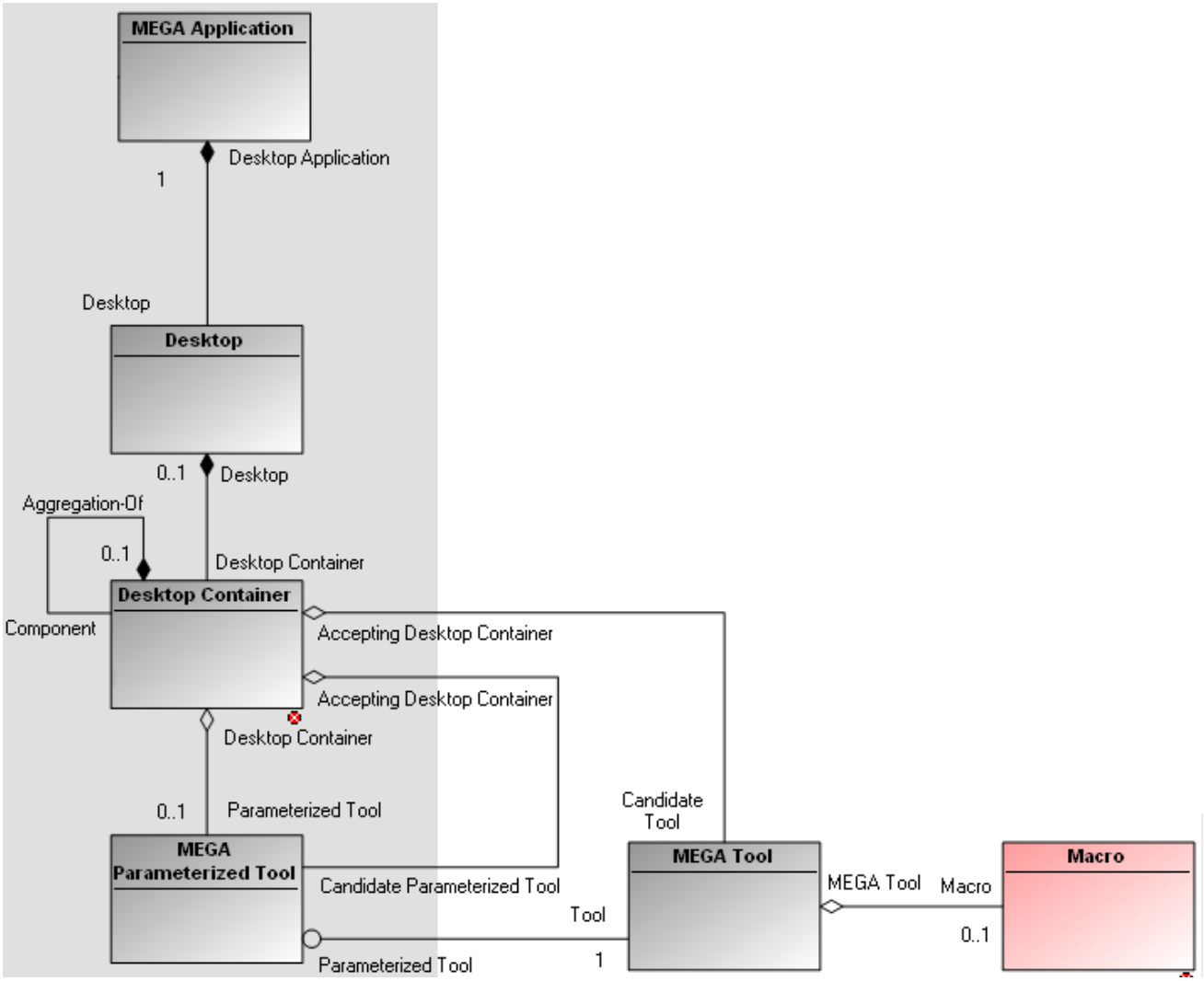
- **a desktop**, which can contain one or several containers
- **containers**, which can contain other containers or desktop components
- **desktop components**, which are:
  - tools associated with a configuration (**MEGA Parameterized Tools**)
    eg: tree, list, menu, HTML formatters.
  - tools (**MEGA Tool**)
    eg: Query, Properties pages, Wizard, Widget, diagram edotors and HTML.

A desktop cannot directly contain a MEGA Tool. The desktop contains a MEGA Parameterized Tool which contains the MEGA Tool.

2.3 Metamodel

The following MetaModel schema shows architecture and links between:

- **Application**
- **Desktop**
- **Container**
- **Desktop components**
- **Tools**


MEGA Application
Desktop Application
1
Desktop
Desktop
0..1
Desktop
Aggregation-Of
0..1
Desktop Container
Desktop Container
Component
Accepting Desktop Container
Accepting Desktop Container
Desktop Container
0..1
Parameterized Tool
Candidate Tool
MEGA Parameterized Tool
Candidate Parameterized Tool
MEGA Tool
MEGA Tool
Macro
Macro
Tool
0..1
Parameterized Tool
1

Principle :

A **user** can connect to one or several desktops depending on his/her profile and authorizations. The user passes from one desktop to another by logoff/login or by using the MEGA Tool **Desktop Switcher**.

A **desktop** comprises one or several containers, in which are defined tools that will be displayed.

A **container** can contain only a single tool.

A **tool** must be hosted by a container.

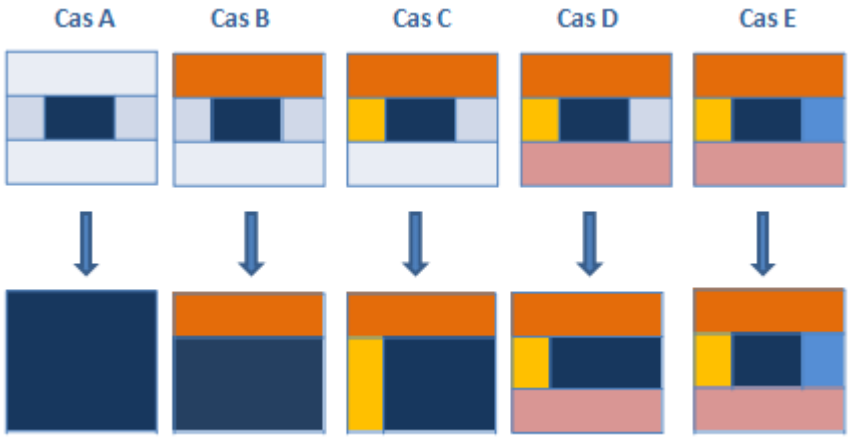
To create and customize the desktop of an application, you should follow these steps:

Step	Action	See chapter	Page
1.	Define desktop configuration	3	8
2.	Define desktop component elements	4	9
3.	Create desktop structure	5	14
4.	Create desktop Containers	6	17
5.	Define characteristics of Containers in work environment	7	32
6.	Configure desktop (pop-up menus, toolbars)	8	40

3 DESKTOP CONFIGURATION

Before configuring a desktop, you must define:

- the elements the desktop will contain and how these elements will be arranged
- the number of containers the desktop will contain.
  The desktop must contain at least one Container (the Center Container).



	Number of elements	Actions
Case A	One	- Define Center Container
Case B	Two	- Define Center Container - Define Top Container
Case C	Three	- Define Center Container - Define Top Container - Define Left Container
Case D	Four	- Define Center Container - Define Top Container - Define Left Container - Define Bottom Container
Case E	Five	- Define Center Container - Define Top Container - Define Left Container - Define Bottom Container - Define Right Container

4 DESKTOP COMPONENT ELEMENTS

A desktop comprises Containers and desktop components (Parameterized Tools).

4.1 Container types

In your desktop you can insert the following **Container** types:

- **Container of Tool**: this Container is a tool recipient.
- **Container of Sub-Containers**: this Container is a recipient designed to receive other Containers of **Container of Sub-Containers** or **Container of Tool** type.

4.2 Container of Sub-Containers

A **Container of Sub-Containers** can be of type:

- **TabPanel**
- **Accordion**
- **Border Layout**

4.2.1 Border Layout

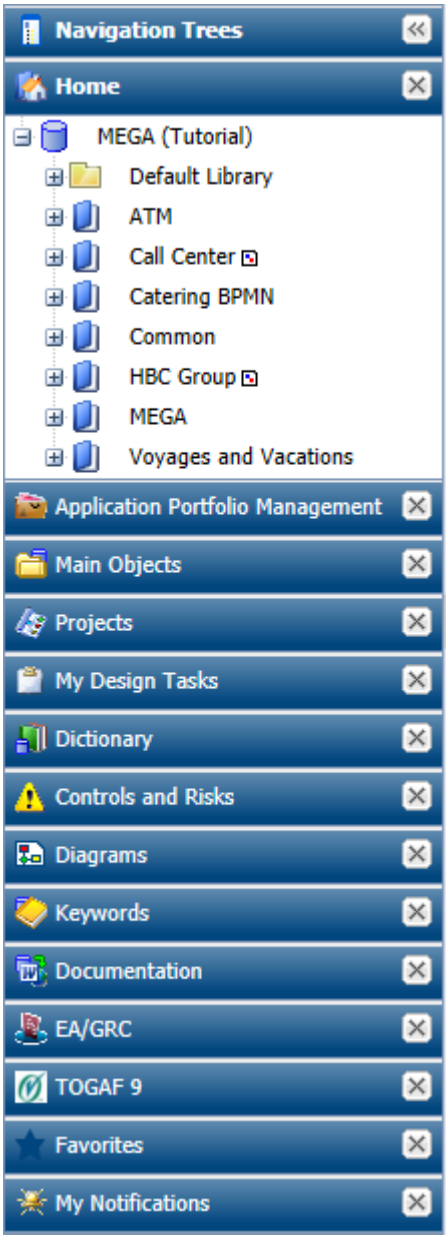
The **Container of Sub-Containers** of **Border Layout** type can comprise one to five Containers. It is represented as here:

The **Center Container** is mandatory.

The **Top**, **Left**, **Right** and **Bottom Containers** are optional.

4.2.2 Accordion

The **Container of Sub-Containers** of **Accordion** type comprises a stack of pages containing desktop components. It can be used for example in a **Left Container**. It is represented as here:



4.2.3 TabPanel.

The **Container of Sub-Containers** of **TabPanel** type comprises tabs. It can be used for example in a **Left Container**. It is represented as here:

4.2.4 Container attributes

Each Container can be customized by means of its attributes. Attributes of a **Container** are defined in its Properties page, Characteristics tab. Certain attributes vary according to Container type:

- **General attributes**:
  - name in user interface
    _GUIName

The name of the Container in the user interface is specified only in the case of a Container of Containers. If the Container contains a tool, it is the tool name that is displayed.

  - icon in user interface
    MetaPicture

The icon in the user interface is specified only in the case of a Container of Containers. If the Container contains a tool, it is the tool icon that is displayed.

  - display or not the title and image of the container in the user interface
    Display Mode
    Default value: "Name and image" for a container of tools and "none" for a container of containers.
  - position of title (top, bottom, left, right)
    Title Position: Top (default value), Bottom, Left, Right
  - type of Container (frame, accordion, tab, none)
    Container Layout: Border Layout, Accordion, TabPanel, (none)
  - position relative to parent (desktop or Container in the case of a Container of Border Layout type) (top, bottom, left, right, center)
    Position:Top, Bottom, Left, Right, Center

- **Dimension** (width, height, minimum height, minimum width)

  Dimension attributes depend on Container type:
  Left Container: Width, minimum Width
  Right Container: Width, minimum Width
  Top Container: Height, Minimum Height
  Bottom Container: Height, Minimum Height

  The center Container has no Dimension attribute, it adapts to desktop dimensions depending on other Containers.

- **Behavior attributes**:

  These attributes simplify desktop readability.

  - closable, collapsible, collapsed, hidden
    Is Closable, is Collapsible, Is Collapsed, Is Hidden
  - default container, container that receives tools that do not have candidate container to receive them
    Is Default Container

- drop for objects authorized or not in its component(s)
  Accepts Dropped Items
- resizable
  Is Resizable (default value: True)

- **Desktop components (MEGA Parameterized Tool):**

  MEGA Parameterized Tool attributes enable insertion of already-parameterized tools.

  Examples:

  - Administration tree
    Administration MetaTree Component
  - Collaboration tree
    Collaboration MetaTree Component
  - Controls and risks tree
    Controls and Risks MetaTree Component
  - Diagrams tree
    Diagrams MetaTree Component
  - Properties page
    Docked PropertyPage Component
  - Documentation tree
    Documentation MetaTree Component
  - Main toolbar undo/redo tool
    Undo/redo

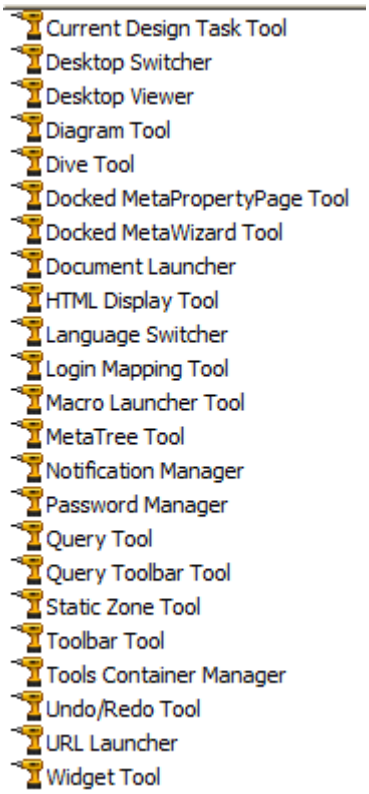
  For example, desktop components of tree type are implemented by the **MEGA Tool** "MetaTree Tool":

  Example: the **MEGA Parameterized Tool** "Documentation MetaTree Component" is implemented by the **MEGA Tool** "MetaTree Tool".

4.3 Tools

Tools (**MEGA Tools**) enable viewing and/or integration with repository data. A MEGA Tool cannot be used directly in a desktop, but only via a desktop component (**MEGA Parameterized Tool**).

Examples of tools:



Characteristic attributes of tools:

- Asynchronous: the tool can be loaded in parallel on the desktop. There is no need to wait for the desktop to be loaded to load the tool (example: ToolbarTool)
  Asynchronous
- Batch Tool: the tool executes without user interface, indicating whether display should be managed or not.
  Is Batch Tool

Examples of use of the **MetaTree Tool** tool:

➢ See Giving a title and/or icon dynamically to a Container p. 32.

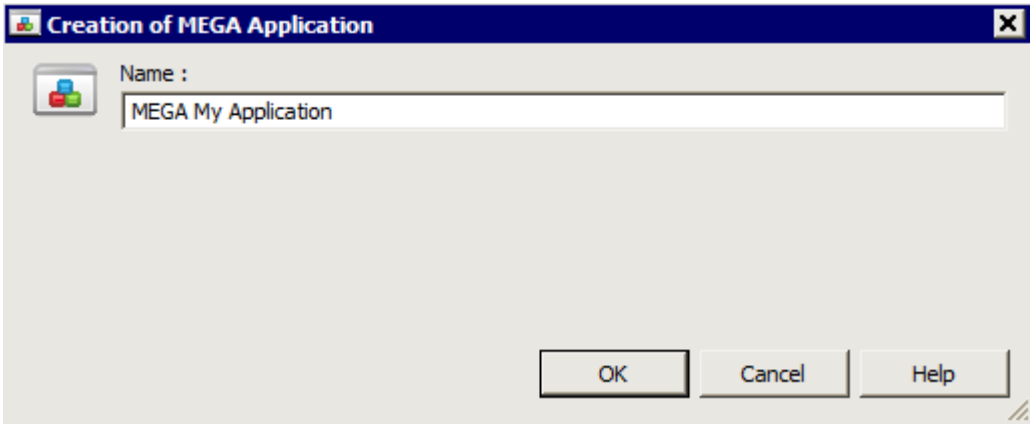
5 CREATING A DESKTOP

You can create a desktop from scratch or from a model already created.

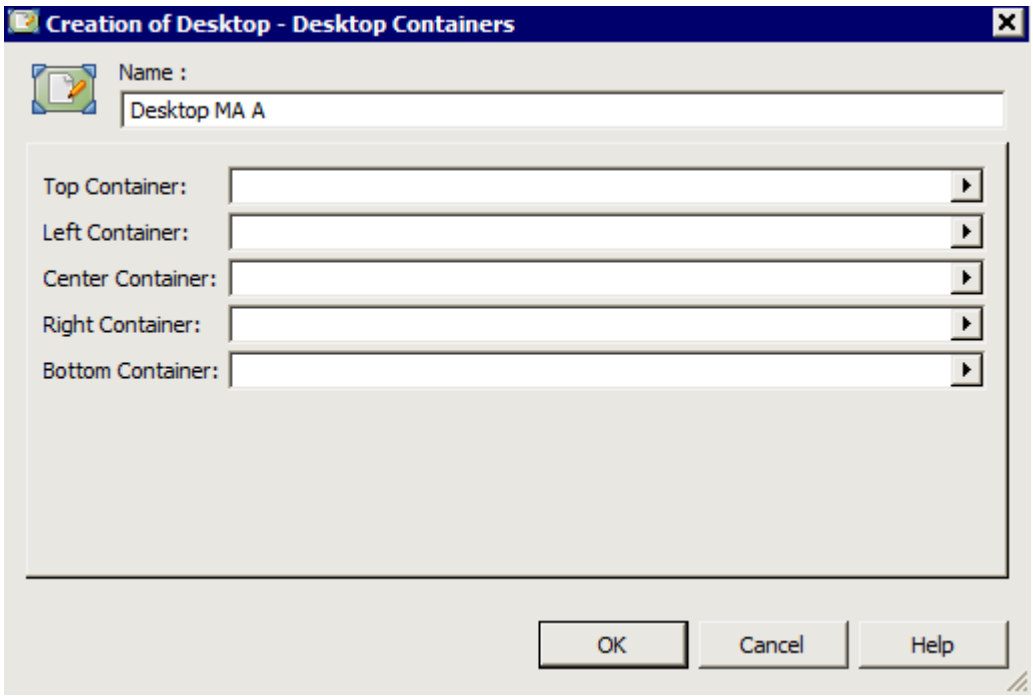
5.1 Creating a desktop from scratch

To create a desktop from scratch:

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Right-click the **MEGA Application** folder and select **New > MEGA Application** to create a MEGA Application.
3. In the **Creation of MEGA Application** dialog box, enter the **Name** of the MEGA Application you want to create (example: « MEGA My Application").



4. Click **OK**.
5. Right-click the folder of the MEGA Application you have just created ("MEGA My Application") and select **New > Desktop**.
   - ➢ The **Creation of Desktop** dialog box appears.
6. Enter the **Name** of the desktop (example: "Desktop MA A").
7. Click **Next**.



8. Depending on configuration of the desktop you want to create (see Desktop configuration) you must specify the required fields:

   - **Center Container** (Mandatory)
   - **Top Container**
   - **Left Container**
   - **Right Container**
   - **Bottom Container**

To specify these fields, see the corresponding procedures, for example see:

- Creating a Desktop Container of Border Layout type p. 17
- Creating a Desktop Container of Accordion type p. 23
- Creating a Desktop Container of TabPanel type p. 27
- Creating a Desktop Container of Tool type p. 30

5.2 Creating a desktop from a model

You can create your desktop starting from a model. This method allows you to start from a structure already set up and simplifies your work. To do this you must duplicate an existing model.

To create a desktop starting from a model:

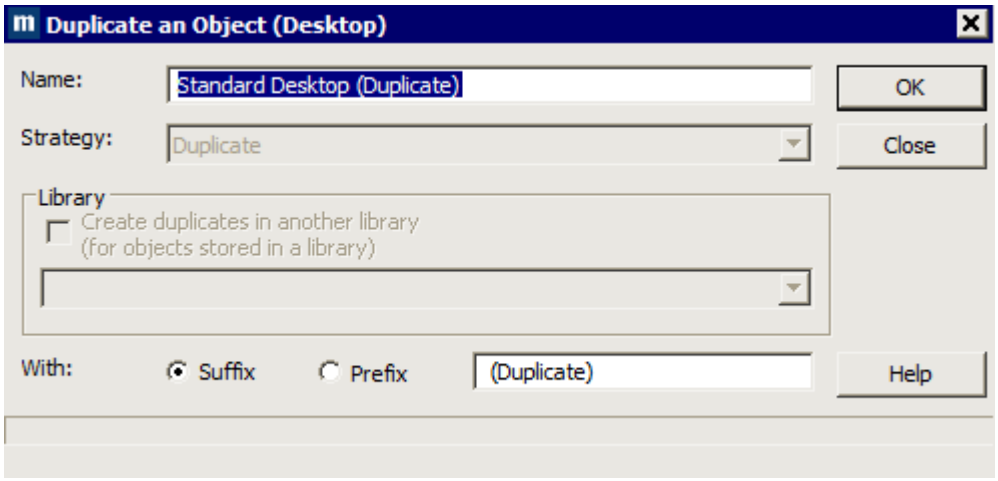
1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Right-click the **MEGA Application** folder and select **New > MEGA Application** to create an Application.
3. In the **Creation of MEGA Application** dialog box, enter the **Name** of the MEGA Application you want to create (example: "MEGA My Application").

4. Click **OK**.

5. Expand the application that contains the desktop you want to use as a model.

6. Right-click the desktop name (example: Standard Desktop) and select **Duplicate**.

   ➢ The **Duplicate an Object (Desktop)** dialog box appears.



7. By default the name of the duplicated desktop is: <Name of desktop> (Duplicate). If you want to change the format of the duplicated desktop name to: Duplication of <Name of desktop>, select **Prefix**.

Warning: If you then modify the name of the desktop in the **Name** field, the trace of duplication is no longer visible.

---

8. Click **OK**.

   ➢ The duplicated desktop is displayed in the application of the model desktop.

9. Select the duplicated desktop (example: Standard Desktop (Duplicate)) and drag-and-drop to the folder of your application (example: "MEGA My Application").

10. (Optional) You can rename the desktop. To do this, select the name of the desktop and press key F2, or from the desktop pages select the **Characteristics** tab.

11. You can rename, move, modify and/or delete Containers from your desktop. This does not modify the model desktop.

Warning: If you modify the name of a tool or a command, the name of this tool or command is also modified in the model desktop.

   ➢ To modify the desktop, see Modifying a desktop p. 74.

6 CREATING DESKTOP CONTAINERS

Prerequisites	Before creating Containers and desktop components, you must: • have already defined desktop components, see Desktop component elements • have already created desktop structure, see Creating a desktop

A **Container** can be represented in the following forms:

- borders (**Border Layout**), see Creating a Desktop Container of Border Layout type p. 17
- accordion (**Accordion**), see Creating a Desktop Container of Accordion type p. 21
- tabs (**TabPanel**), see Creating a Desktop Container of TabPanel type p. 27
- tool (**Container of Tool**), see Creating a Desktop Container of Tool type p. 30.

A **Container** can contain:

- a group of tools, see Creating a toolbar p. 55
- a group of Commands, see Creating pop-up menus p. 43.

6.1 Creating a Desktop Container of Border Layout type

The **Desktop Container**, is a container of **Border Layout** type for the desktop. It comprises:

- a **Center Container** (mandatory container)
- **Top**, **Left**, **Right**, and **Bottom Containers** (optional).

For example, the Desktop Container of **Border Layout** type can contain:

- a **Center Container**, which can contain the home page, see 6.1.1 p. 17.
- a **Bottom Container**, which can contain the Properties page, see 6.1.2 p. 21.

Creation of this Desktop Container of Border Layout type calls three creation wizards:

- **Desktop Container** creation wizard ("Edit Area")
- **Center Container** creation wizard ("Main Page") of **Desktop Container** ("Edit Area")
- **Bottom Container** creation wizard ("Properties Page") of **Desktop Container** ("Edit Area").

6.1.1 Creating the Center Container of a Border Layout Container

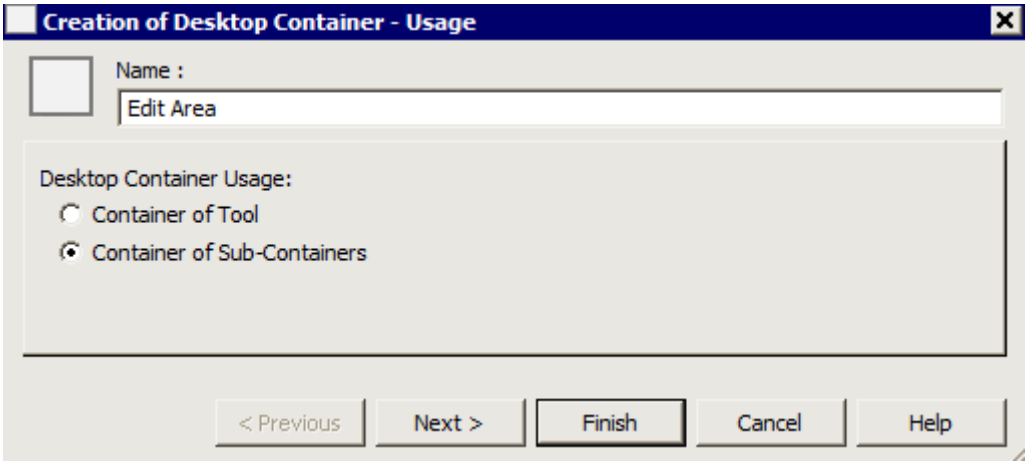
To create the Center Container of a Desktop Container of Border Layout type:

1. From the **Creation of Desktop – Desktop Containers** dialog box (see Creating a desktop), click the **Center Container** field arrow and select **New**.

   ➢ The first step of the **Creation of Desktop Container** wizard appears: **Usage**.

2. Enter the **Name** of the Container of Border Layout type (example: «Edit Area").

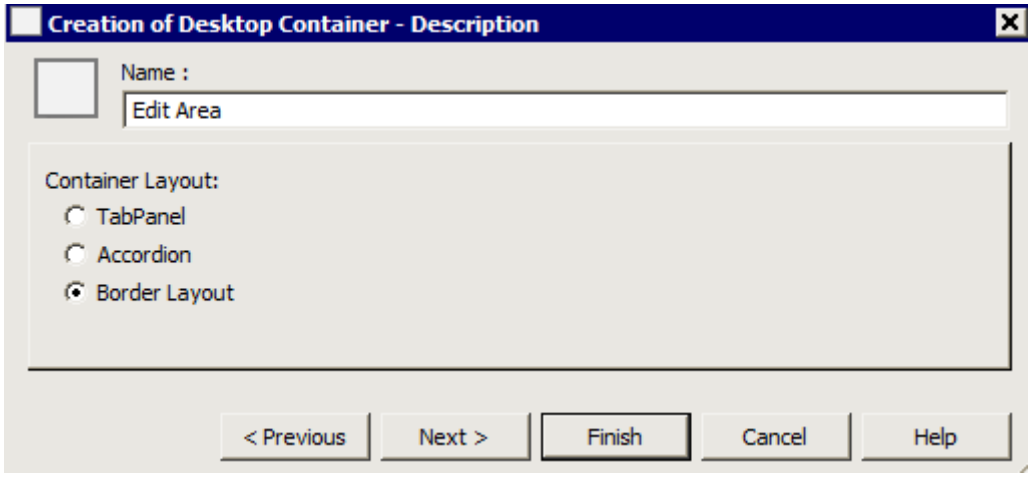
3. Select **Container of Sub-Containers**.



4. Click **Next**.

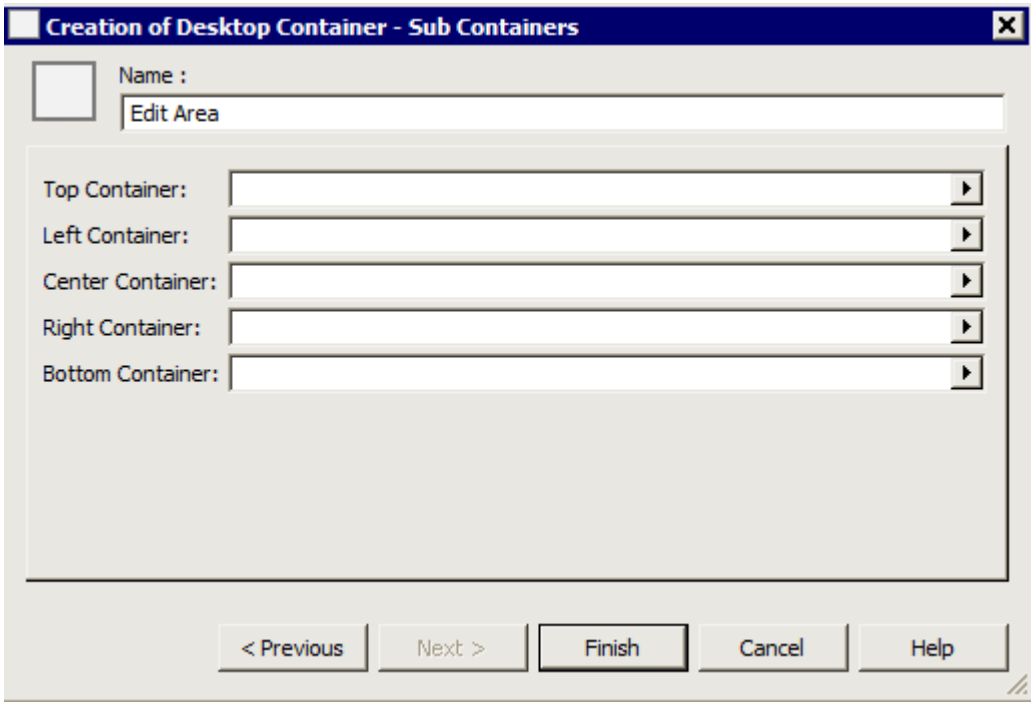
   ➢ The second step of the **Creation of Desktop Container** wizard appears: **Description**.

5. In the **Container Layout** frame, select the Container layout type, example: **Border Layout**.



6. Click **Next**.

   ➢ The third step of the **Creation of Desktop Container** wizard appears: **Sub Containers**.



7. Click the **Center Container** field arrow and select **New**.

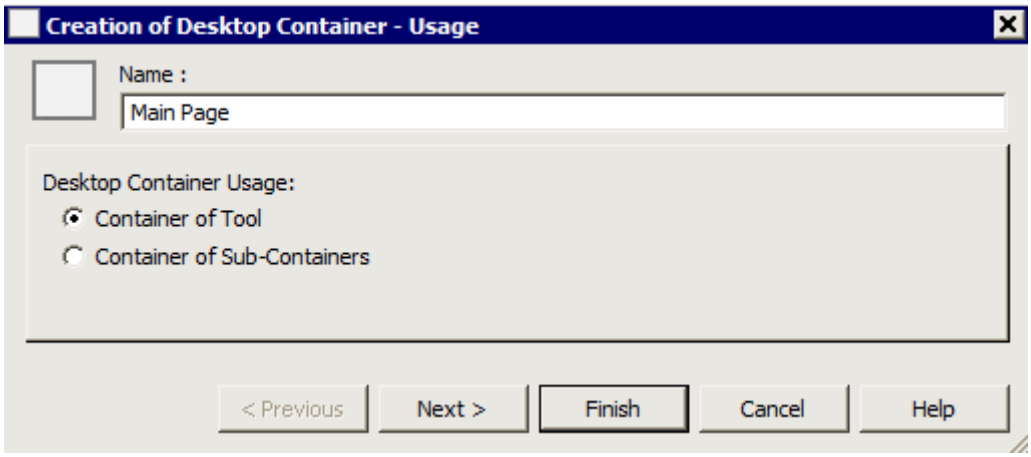
   ➢ The first step of a new **Creation of Desktop Container** wizard appears: **Usage**. This enables definition of use of the Center Container of the "Edit Area" Container.

Depending on the layout of the Container you want to create, click arrows of the **Top**, **Left**, **Right** and/or **Bottom Container** fields and select **New**.

---

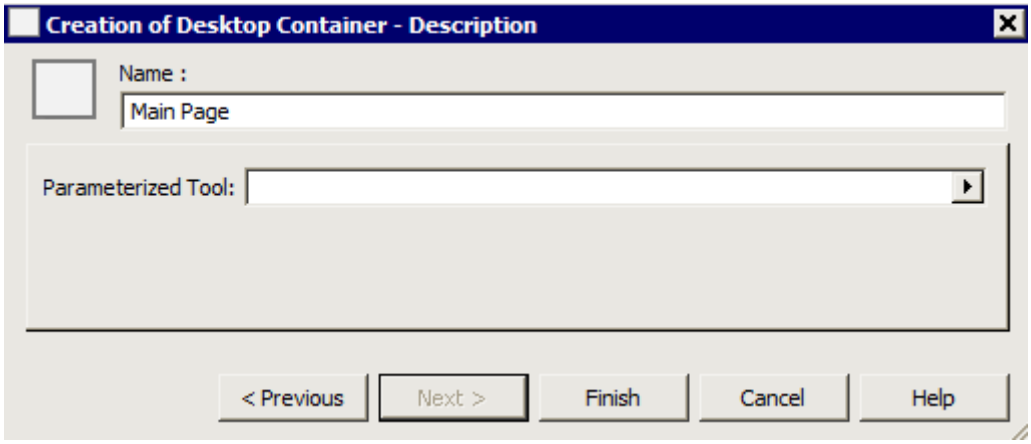
8. Enter the **Name** of the container (example: "Main Page").

9. In the **Desktop Container Usage** frame, select **Container of Tool**.

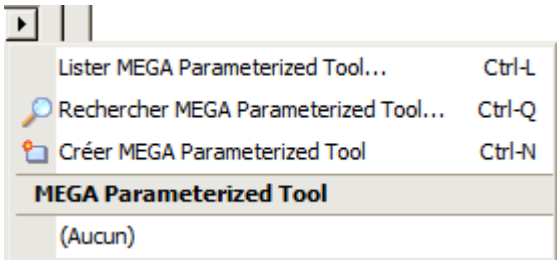


10. Click **Next**.

   ➢ The second step of the new **Creation of Desktop Container** wizard appears: **Description**. This enables description of the Center Container of the "Edit Area" Container.

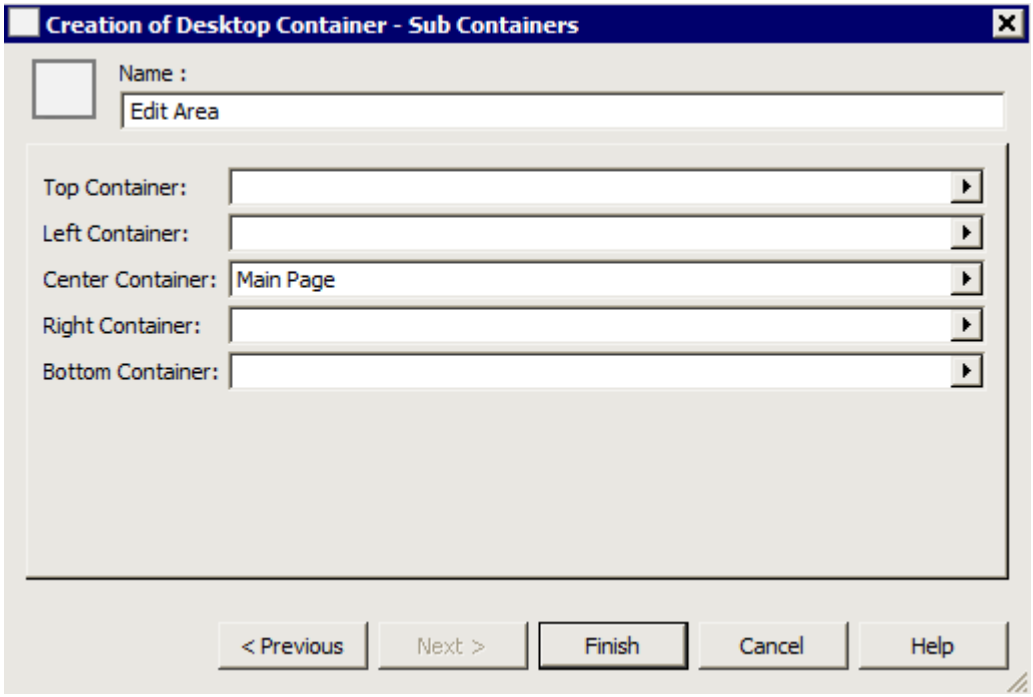


11. In the **Parameterized Tool** field, click the arrow and select or create the tool you require (example: "Widget Component").



12. Click **Finish**.

    - "Main Page" (which contains the "Widget Component" tool) appears in the **Center Container** field.



13. See Creating the Bottom Container of a Border Layout Container before clicking **Finish**.

    - The Container of the home page is created.

6.1.2 Creating the Bottom Container of a Border Layout Container

To create the Bottom Container of a Desktop Container of Border Layout type:

1. From the **Creation of Desktop Container – Sub Containers** dialog box, click the **Bottom Container** field arrow and select **New**.

   ➢ The first step of a new **Creation of Desktop Container** wizard appears: **Usage**. This enables definition of use of the Bottom Container of the "Edit Area" Container.

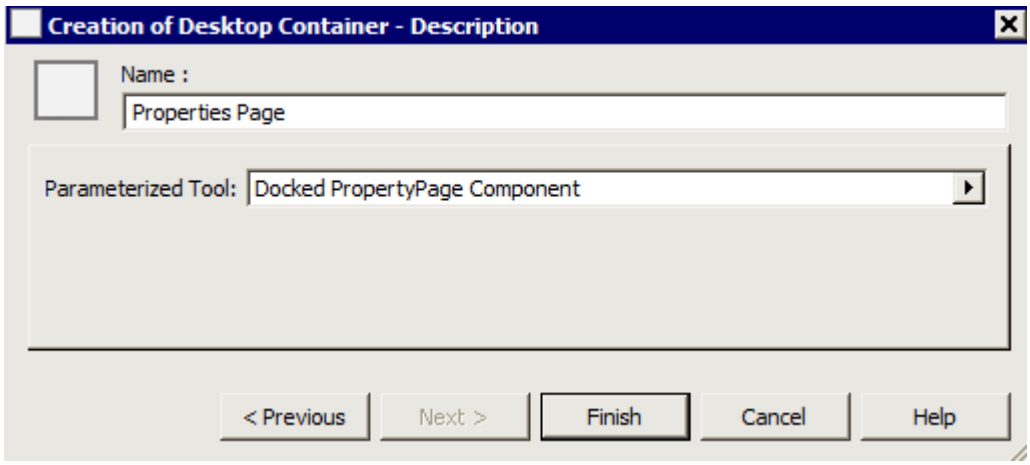
2. Enter the **Name** of the Bottom Container (example: "Properties Page").

3. In the **Creation of Desktop Container** frame, select **Container of Tool**.

4. Click **Next**.

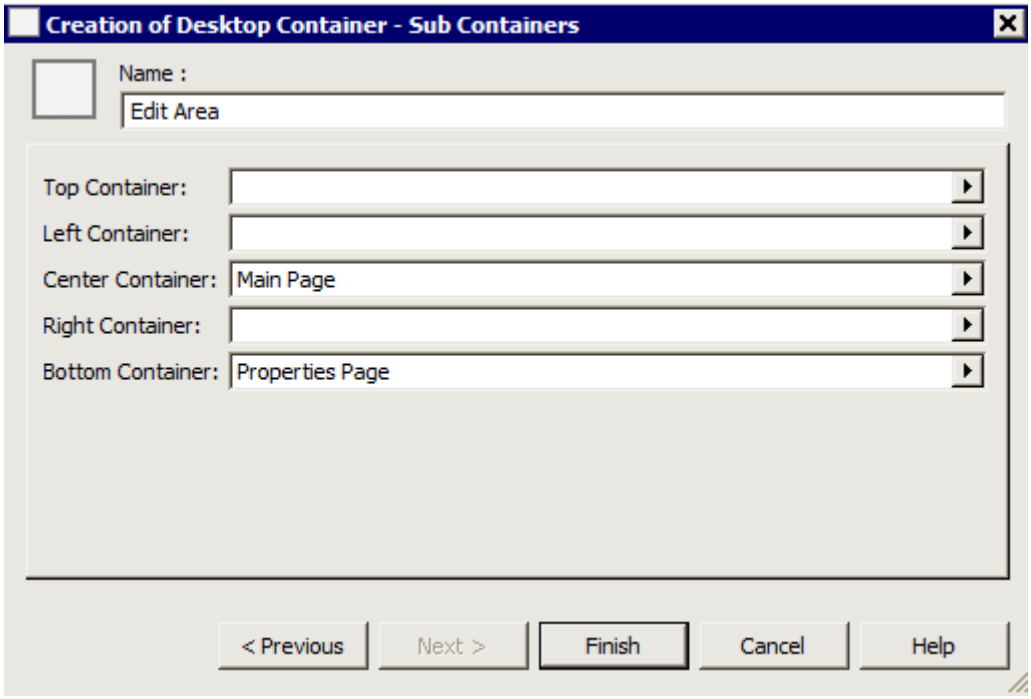
   ➢ The second step of the new **Creation of Desktop Container** wizard appears: **Description**. This enables description of the Bottom Container of the "Edit Area" Container.

5. In the **Parameterized Tool** field, click the arrow and select (or create) the MEGA Parameterized Tool you require (example: **Docked PropertyPage Component**).



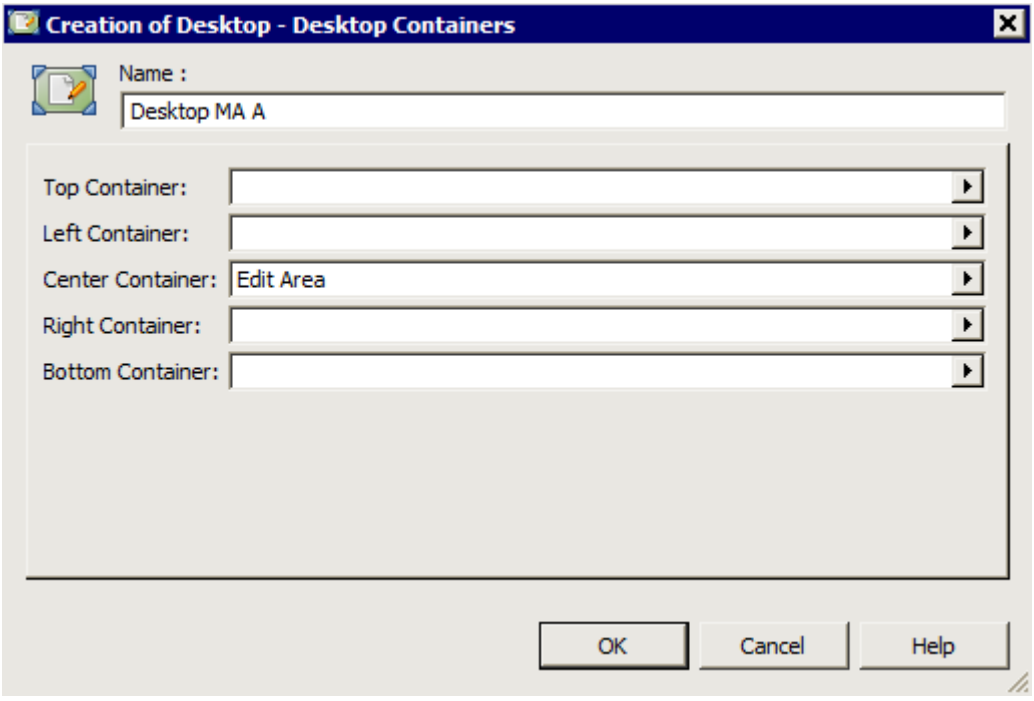
6. Click **Finish**.

   ➢ The **Properties of Desktop Container** "Edit Area" dialog box appears.



- The two Containers ("Main Page" and "Properties Pages") of the "Edit Area" Center Container are created.

7. Click **Finish**.



- The "Edit Area" **Center Container** of Border Layout type of desktop ("Desktop MA A") is created and configured.

When the **Center Container** has been specified, you can interrupt creation of other Containers of your desktop.
To create a new Container later, see Adding a Desktop Container to a desktop p. 74.

6.2 Creating a Desktop Container of Accordion type

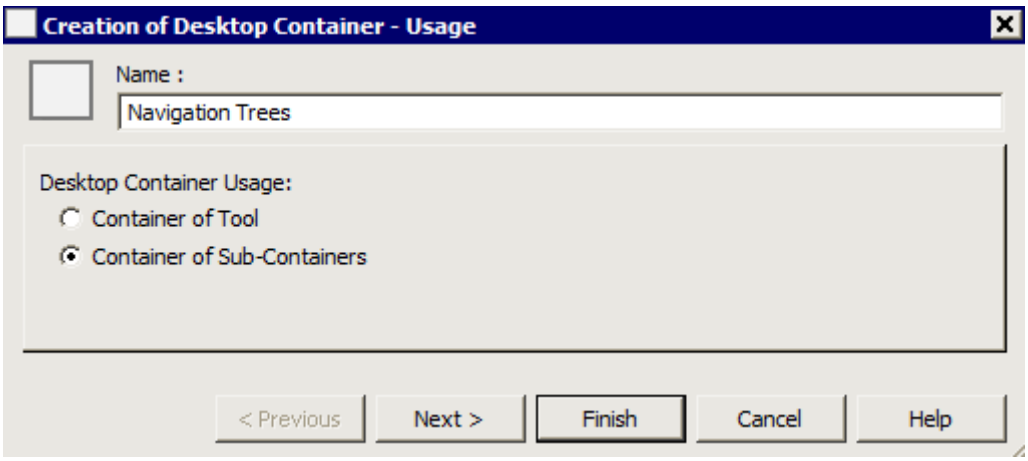
To create a Desktop Container of Accordion type:

1. From the **Creation of Desktop – Desktop Containers** dialog box (see Creating a desktop), click the **Left Container** field arrow and select **New**.

   ➢ The first step of the **Creation of Desktop Container** wizard appears: **Usage**.

2. Enter the **Name** of the Desktop Container of Accordion type (example: "Navigation Trees").

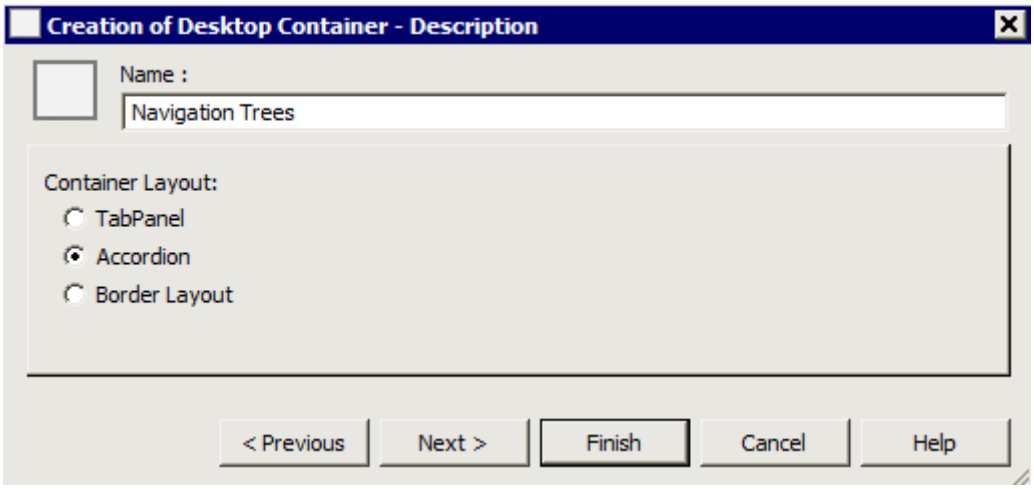
3. Select **Container of Sub-Containers**.



4. Click **Next**.

   ➢ The second step of the **Creation of Desktop Container** wizard appears: **Description**.

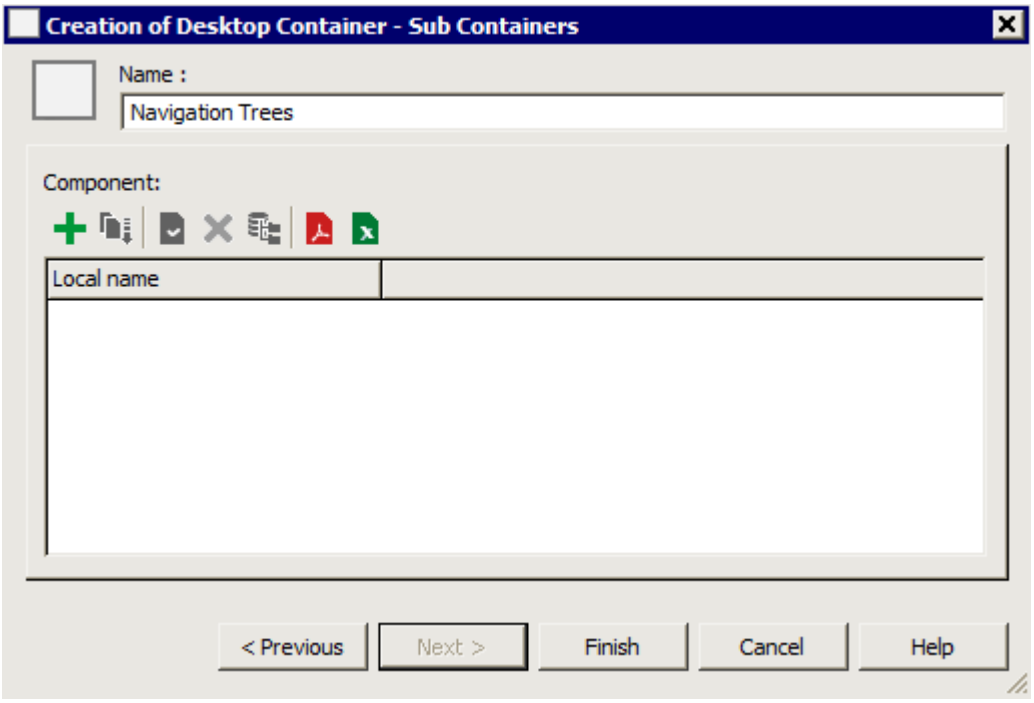
5. In the **Container Layout** frame, select the Container layout type: **Accordion**.



6. Click **Next**, to specify content of the **Accordion** Container ("Navigation Trees").

Click **Finish**, if you want to postpone till later the specification of the content of the **Accordion** Container ("Navigation Trees").

➢ The third step of the **Creation of Desktop Container** wizard appears: **Sub Containers**.

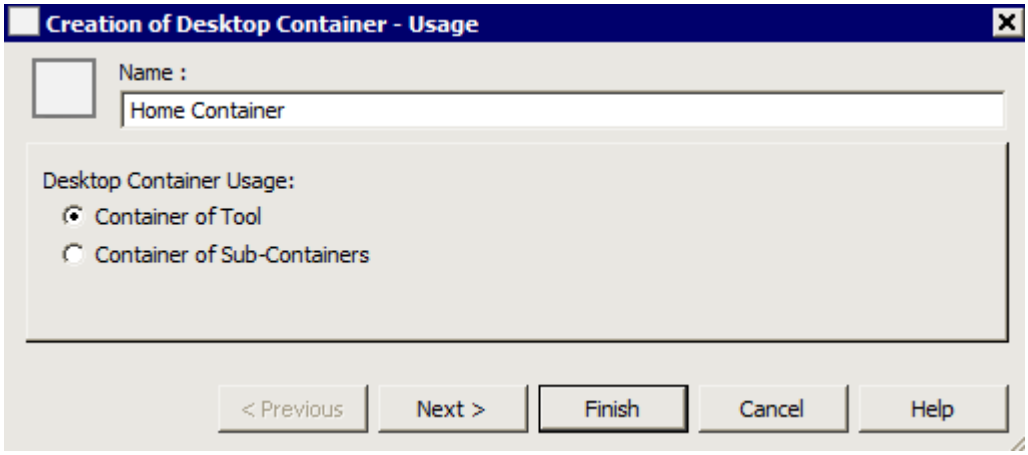


7. In the **Component** frame, click **New** .

➢ The first step of a new **Creation of Desktop Container** wizard appears: **Usage**. This enables definition of use of the first Container of the "Navigation Trees" Container.

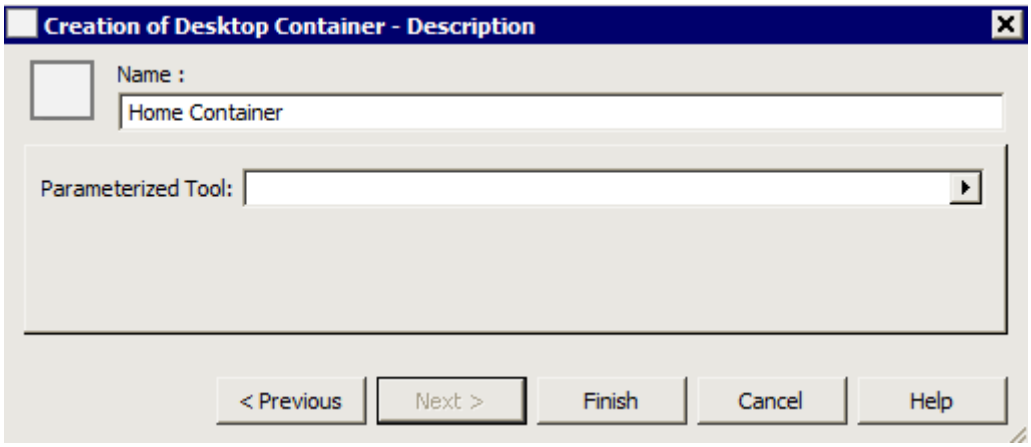
8. Enter the **Name** of the Container (example: "Home Container").

9. Select for example **Container of Tool**.

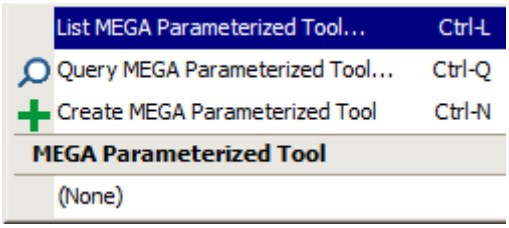


10. Click **Next** (or Finish if you wish to complete later).

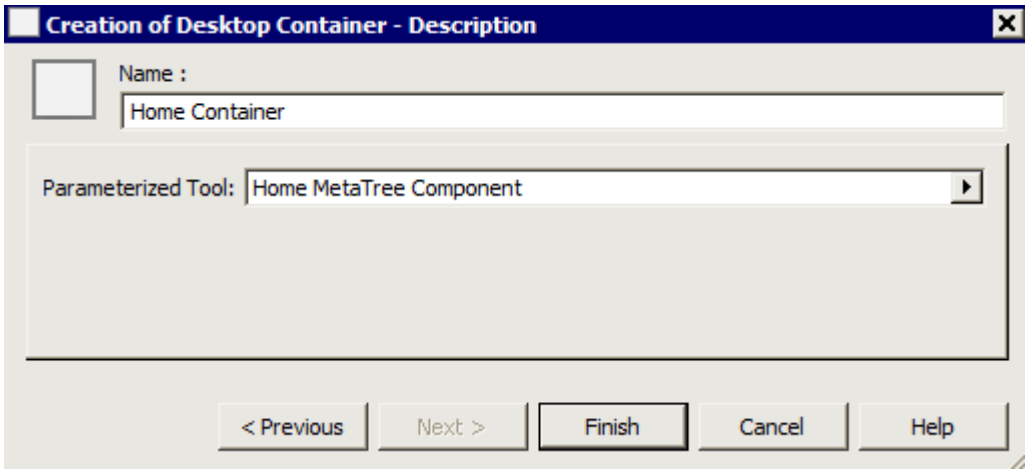
➢ The second step of the new **Creation of Desktop Container** wizard appears: **Description**. This enables description of the first Container of the "Navigation Trees" Container.



11. In the **Parameterized Tool** field, click the arrow and select or create the tool you require (example: "Home MetaTree Component").

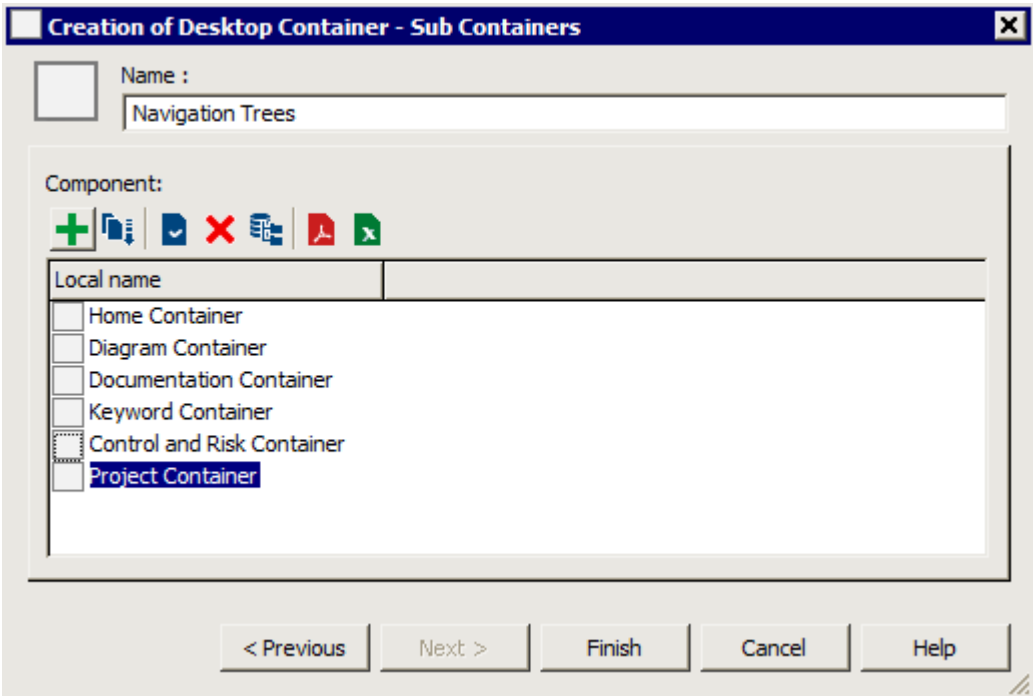


12. Click **OK**.



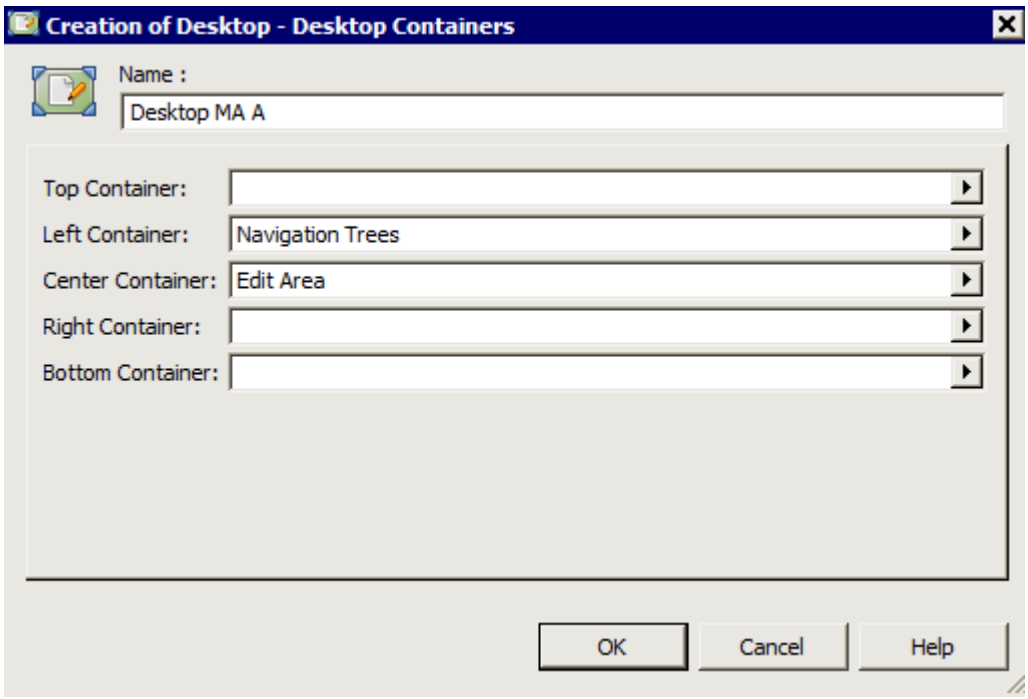
13. Click **Finish**.

14. Repeat steps 7 to 13 and create other Containers of the "Navigation Trees" Container.



15. Click **Finish**.

➢ "Navigation Trees" (which contains the trees created) appears in the **Left Container** field of the desktop ("Desktop MA A").



6.3 Creating a Desktop Container of TabPanel type

To create a Container that contains tabs that are always visible, you must create a **Desktop Container** of **TabPanel** type.

To create tabs on the fly, see Defining Container candidates p. 35.

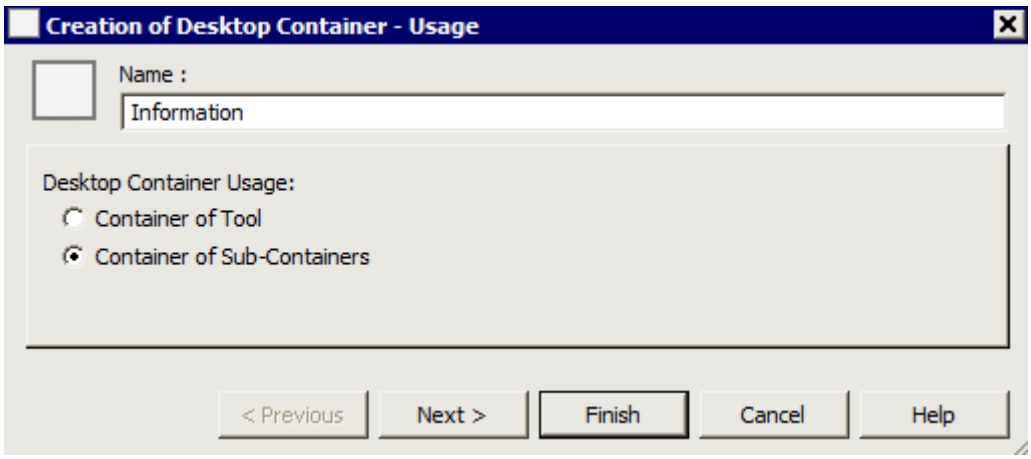
To create a Desktop Container of TabPanel type:

1. From the **Creation of Desktop – Desktop Container** dialog box (see Creating a desktop), in the **Bottom** (**Top**, **Left**, **Center** or **Right**) **Container** field, click the arrow and select **New**.

   ➢ The first step of the **Creation of Desktop Container** wizard appears: **Usage**.

2. Enter the **Name** of the Container (example: "Information").

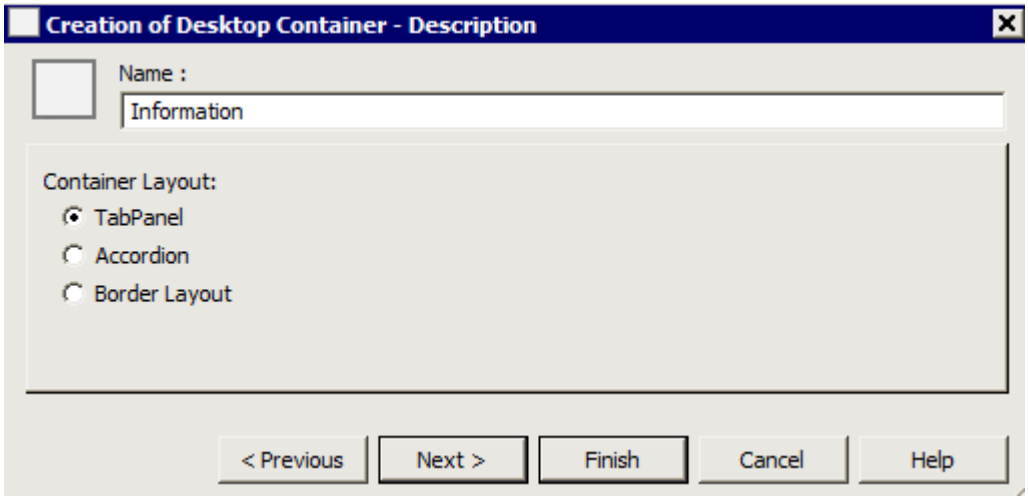
3. In the **Desktop Container Usage** field, select **Container of Sub-Containers**.



4. Click **Next**.

   ➢ The second step of the **Creation of Desktop Container** wizard appears: **Description**.

5. In the **Container Layout** frame, select the Container layout type: **TabPanel**.



6. Click **Next**.

   ➢ The third step of the **Creation of Desktop Container** wizard appears**: Sub Containers**.

7. In the **Component** frame, click **New** to insert the tabs you require.

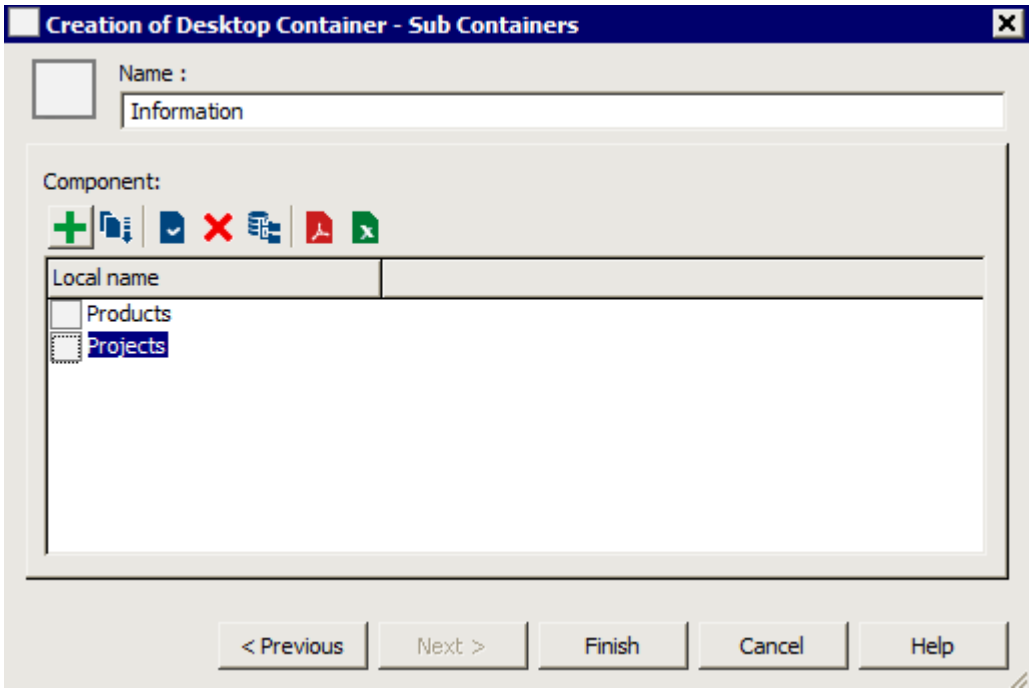
   ➢ The first step of a new **Creation of Desktop Container** wizard appears: **Usage**. This enables definition of use of the first tab of the "Information" Container.

8. Enter the **Name** of the tab (example: Products).

9. In the **Desktop Container Usage** frame, select **Container of Tool**.

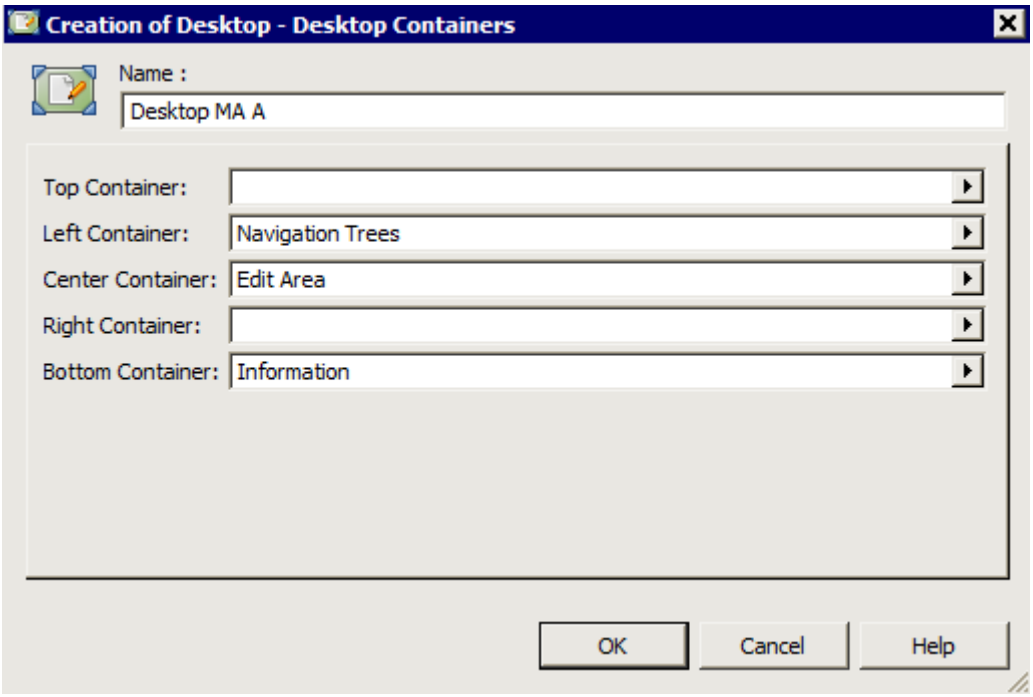
10. Click **Finish** (you can create tools later).

11. Repeat steps 7 to 10 as many times as there are tabs to define.



12. Click **Finish**.

➢ "Information" (which contains tabs "Products" and "Projects") appears in the **Bottom Container** field of the desktop ("Desktop MA A").



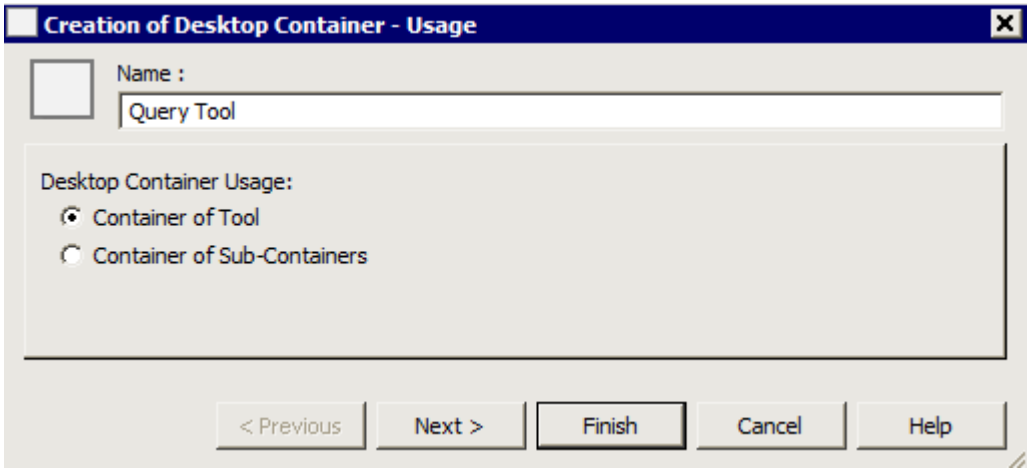
6.4 Creating a Desktop Container of Tool type

To create a Desktop Container of Tool type:

1. From the **Creation of Desktop – Desktop Containers** dialog box (see Creating a desktop), in the **Right Container** (**Top**, **Left**, **Center** or **Bottom**) field, click the arrow and select **New**.

   ➢ The first step of the **Creation of Desktop Container** wizard appears: **Usage**.

2. Enter the **Name** of the Container (example: "Query Tool").



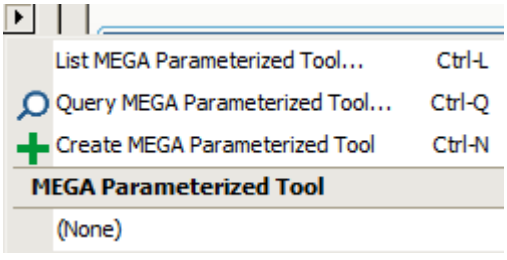
3. In the **Desktop Container Usage** frame, select **Container of Tool**.

4. Click **Next**.

   ➢ The second step of the **Creation of Desktop Container** wizard appears: **Description**.

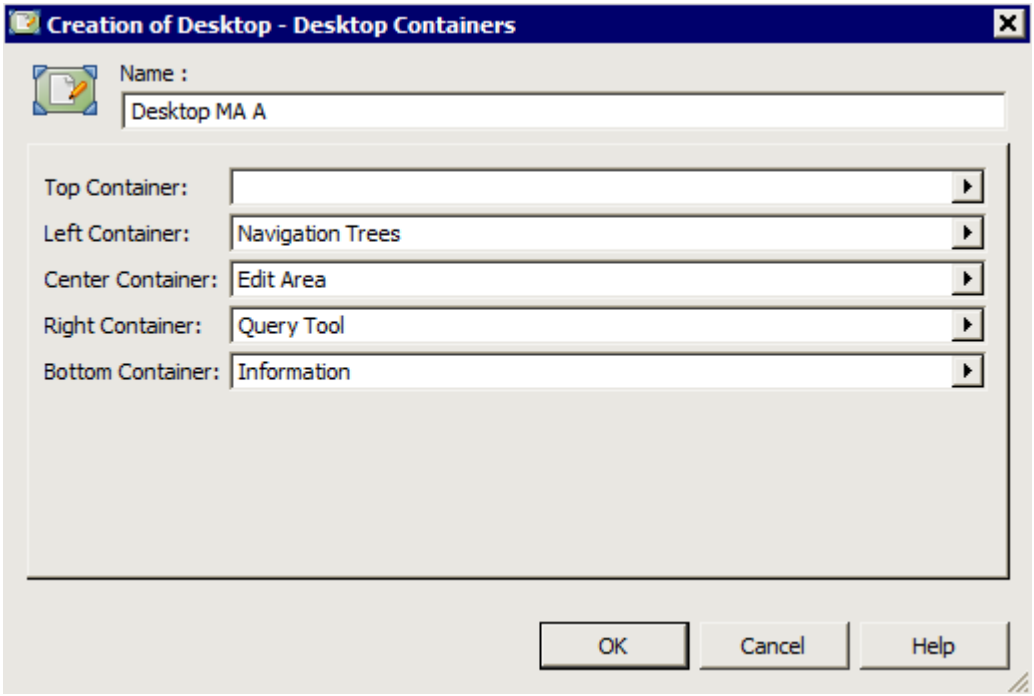
8. Click the **Parameterized Tool** field arrow, select **List MEGA Parameterized Tool** and select the tool you require (example: "Query Component").

Alternatively: if you know the name of the tool, select **Query MEGA Parameterized Tool**; if the Tool you want to insert does not exist, select **Create MEGA Parameterized Tool**.



9. Click **Finish**.

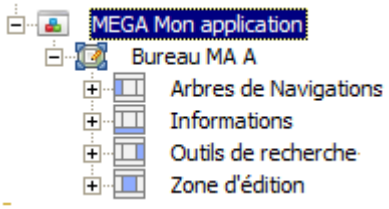
   ➢ The name of the tool (example: "Query Tool") appears in the field concerned (example: **Right Container**).



6.5 Completing Desktop Container creation

When Desktop Containers have been specified, to complete desktop creation:

- From the **Creation of Desktop – Desktop Containers** dialog box (see Creating a desktop), click **OK** (or **Finish**).
  - The desktop and its Containers appear in the tree of your application.



When the **Center Container** has been specified, you can interrupt creation of the other Containers (optional) of your desktop.
To create a new Container, see Adding a Desktop Container to a desktop p. 74.

---

7 DEFINING CONTAINER CHARACTERISTICS

7.1 Customizing Containers

Having created a Container, you can customize its display and size in the workspace. To do this, you must specify its characteristics (see Container attributes p. 10) in its Properties pages:

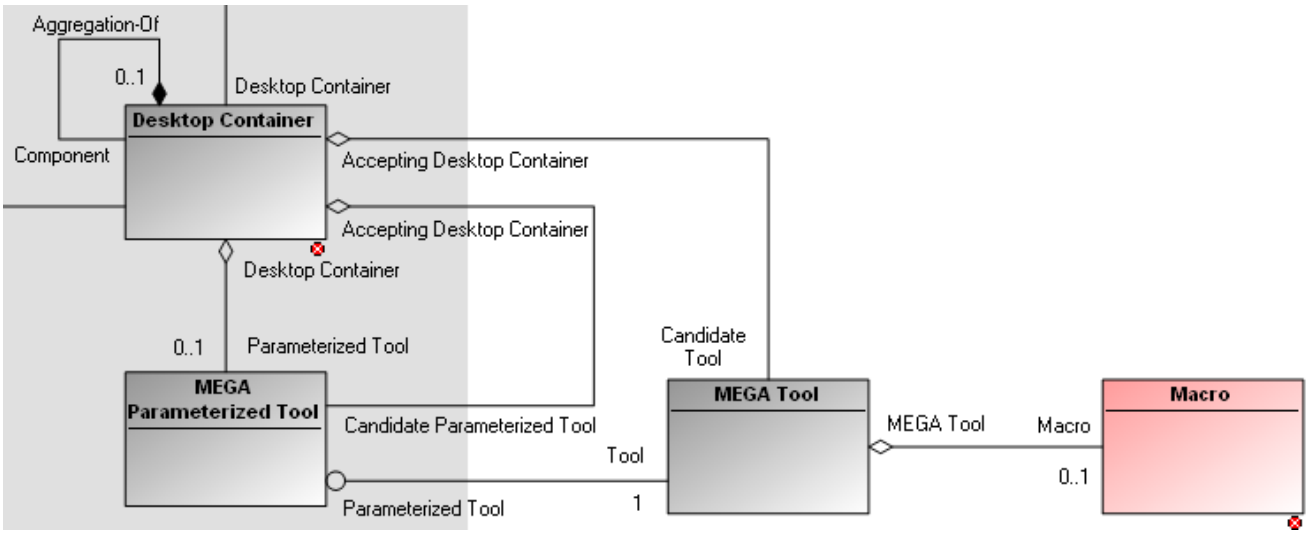
1. Open the **Properties** dialog box of the Container.
2. Select the **Characteristics** tab.
3. In the **Presentation** frame, you can for example specify the name of the Container in the user interface (_GUIName) and its corresponding icon (MetaPicture).

_GUIName and MetaPicture are only specified in the case of a Container of Containers. If the Container contains a tool, then the _GUIName and MetaPicture of the tool are displayed.

Note: Alternatively, the title and icon of the Container can be specified dynamically in the user interface by means of a macro, see Giving a title and/or icon dynamically to a Container p. 32.

7.2 Giving a title and/or icon dynamically to a Container

To give a title and/or icon dynamically to a Container, you must create a macro, which you will connect to the **MEGA Tool** of the Container. It is the **MEGA Tool** (example: "MetaTree Tool") that gives the title (GUI name) and icon to the Container.



To give the title and icon of a Container dynamically in the user interface (example: "Home Container"):

1. In the **MEGA Desktop** workspace, execute a **Query** on the **MEGA Tool** concerned (example: "MetaTree Tool").

   Alternatively, if you do not know the name of the **MEGA Tool**:

   a. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.

   b. Expand the **MEGA Application** folder, then the application and desktop concerned.

   c. Expand the **Desktop Container** (example: "Navigation Trees"), then the **Desktop Container Component** (example: "Home Container" of) then the **MEGA Parameterized Tool** (example: "Home MetaTree Component") concerned.

2. Right-click the **MEGA Tool** (example: "MetaTree Tool") concerned and select **Explore**.

3. From the exploration tree of the **MEGA Tool** (example: "MetaTreeTool), right-click **Macro** and select New.

4. Edit the macro and specify in the script:

   - the function that will specify the title:

     Function GetTitle (mgRoot As MegaRoot, ParameterizedToolId as Object)
     …/…
     End Function

Example of macro implemented for the **MEGA Tool**"MetaTree Tool":

```
Function GetTitle(mgRoot As MegaRoot, ParameterizedToolId as Object)
  Dim mgScanner
  Set mgScanner = New Scanner
  Set mgScanner.mgResource = mgRoot.CurrentEnvironment.Resources
  Set mgScanner.mgToolkit = mgRoot.currentEnvironment().toolkit()

  mgScanner.mgScan = 1
  mgScanner.mgFunction = 1

  mgScanner.mgResource.ScanCollection ParameterizedToolId, "Abstract
Property", mgScanner ,1 , "Reference" & " " & "MetaAttribute Type" & ":T"
   If mgScanner.mgName = "" Then
      GetTitle = "!!! No GuiName Found"
Else
  GetTitle = mgScanner.mgName
 End If

 End Function
```

   - the function that will call the icon:

     Function GetPicture(mgRoot As MegaRoot, ParameterizedToolId as Object)
     …/…
     End Function

Example of macro implemented for the **MEGA Tool**"MetaTree Tool":

```
Function GetPicture(mgRoot As MegaRoot, ParameterizedToolId as Object)
Dim mgScanner
  Set mgScanner = New Scanner
  Set mgScanner.mgResource = mgRoot.CurrentEnvironment.Resources
  Set mgScanner.mgToolkit = mgRoot.currentEnvironment().toolkit()

  mgScanner.mgScan = 1
  mgScanner.mgFunction = 2
  mgScanner.mgResource.ScanCollection ParameterizedToolId,"Abstract
Property", mgScanner ,1 , "Reference" & " " & "MetaAttribute Type" & ":T"

GetPicture = mgScanner.mgPicture
  If mgScanner.mgPicture = "" Then
      GetPicture = ""
Else
  GetPicture                    =                    "~"                    &
mgRoot.CurrentEnvironment.Toolkit.GetString64FromID(mgScanner.mgPicture)
 End If
End Function
```

- In the user interface, the title and icon of the Container concerned (example: "Home Container") are specified dynamically. You do not need to specify **_GUIName** and **MetaPicture** in the **Properties** of the Container. When the name of the Container changes, modification is taken into account automatically.

7.3 Defining Container dimensions

You cannot define dimensions of the Center Container, this adapts to the other Containers to fill the space.

By default, the width of a Container (other than the Center Container) is defined as 600 pixels. You can modify this width and/or define a minimum width (**Minimum Width**).

To modify the dimensions of a Container:

1. Open the **Properties** dialog box of the Container.
2. Select the **Characteristics** tab.
3. In the **Dimension** frame, specify the parameters concerned (width, **Minimum Width**).

7.4 Defining Container behavior

Having created a Container, you must define its behavior in the workspace. To do this, you must specify its **Characteristics** (see Container attributes p. 10) in its **Properties** pages:

1. Open the **Properties** dialog box of the Container.
2. Select the **Characteristics** tab.
3. In the **Behavior** frame, select the required behaviors.

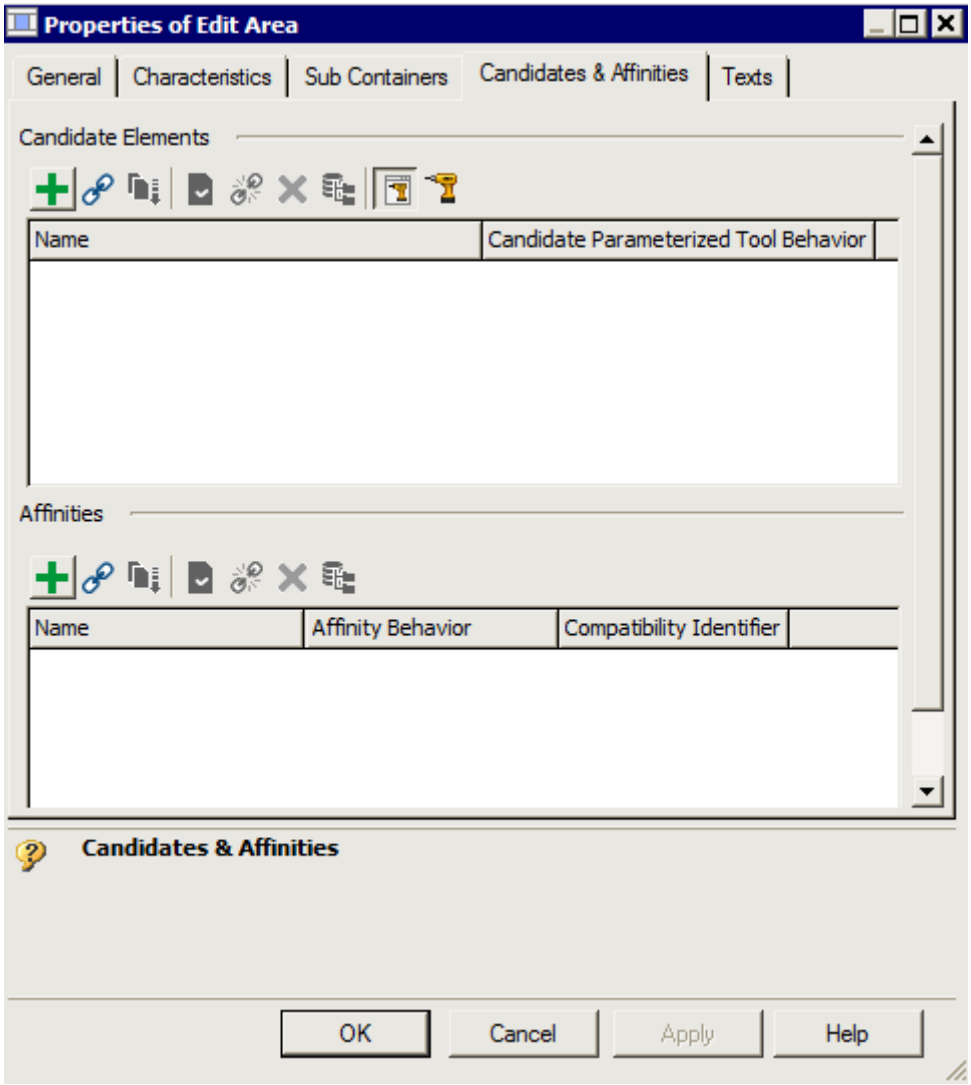
Note for example that by default a Container is visible and not resizable.

7.5 Defining Container candidates

To define where the **MEGA Tool** or the **MEGA Parameterized Tool** will open when you request its opening (from the pop-up menu of an object for example) you must define this **MEGA Tool** or **MEGA Parameterized Tool** as candidate for a Container.

1. Open the **Properties** dialog box of the Container in which you want to see the tool appear (example: "Desktop Container Edit Area").

2. Select the **Candidates & Affinities** tab.



3. In the **Candidate Elements** frame, click the icon of the candidate concerned **Candidate Parameterized Tool** or **Candidate Tool** .

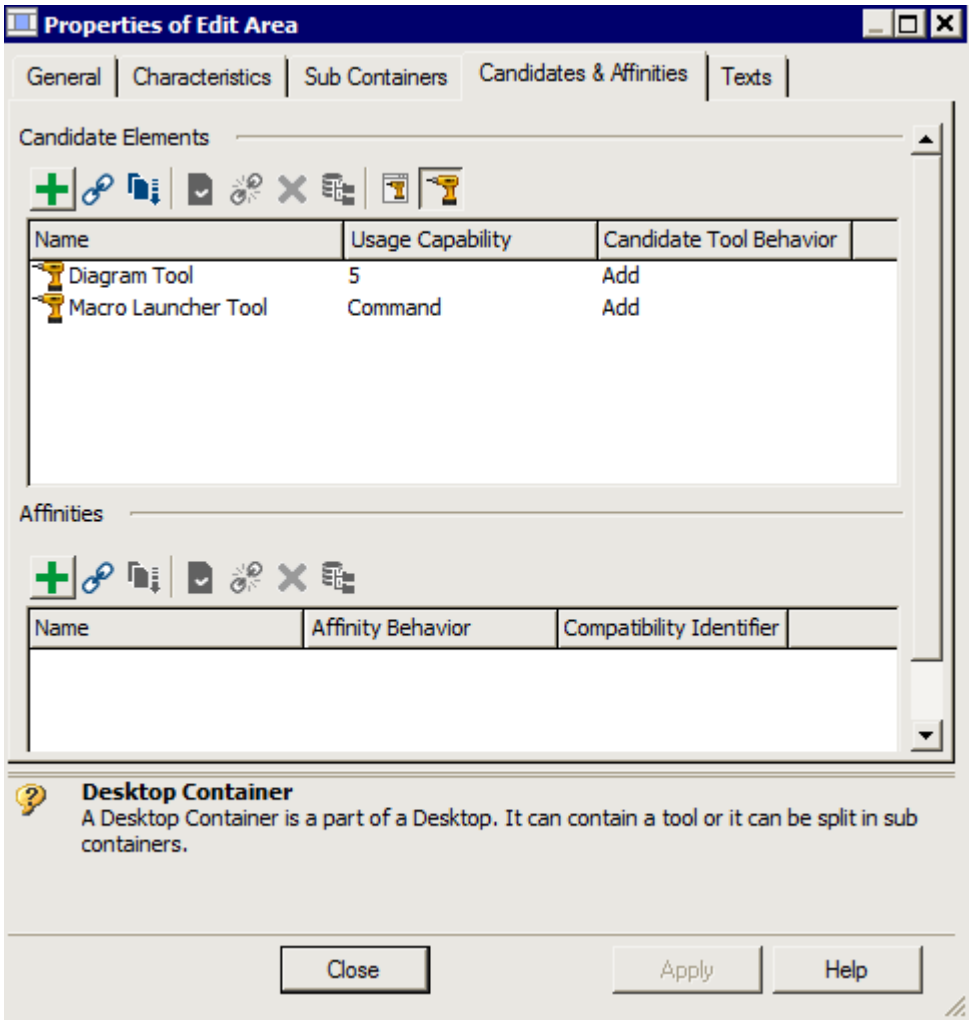
4. Click **Connect** (or **New** if the MEGA Parameterized Tool/Tool concerned is not yet created).

5. Select the MEGA Parameterized Tool/Tool and click **OK**.

- The Parameterized Tools/Tools are listed in the Candidate Elements frame of the Desktop Container (example: "Edit Area").

In the **Candidate Tool/Parameterized Tool Behavior** field, the value "Replace" indicates that the existing page will be replaced, the value "Add" indicates that a new page will be added.

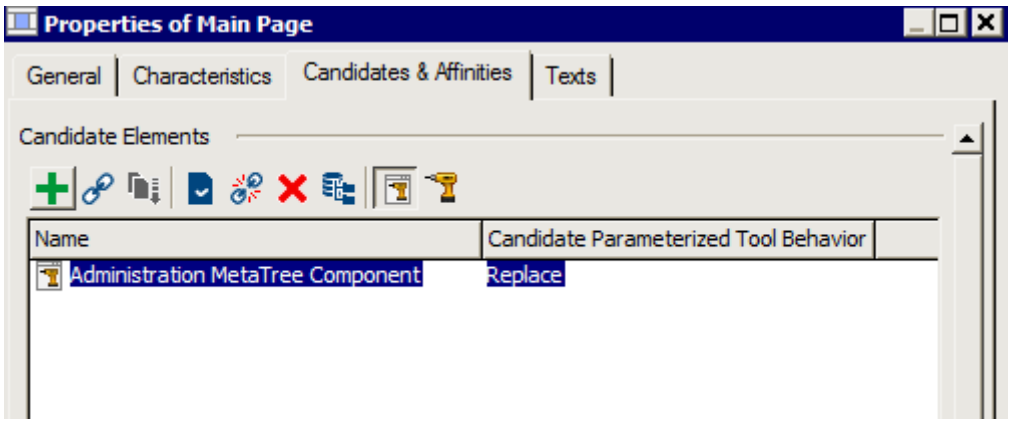
At an opening command of these **Parameterized Tools/Tools**, they open in the specified Container (example: "Edit Area").



Example:

In our case, the Left Container of **Accordion** type is candidate to receive all trees. For example, the Favorites tree opens in the Accordion Container.

- If you want a particular tree (example: Administration) to open in a Container A (example: "Main Page"), this Container A (example: "Main Page") must have as **Candidate Parameterized Tool** the Administration tree.



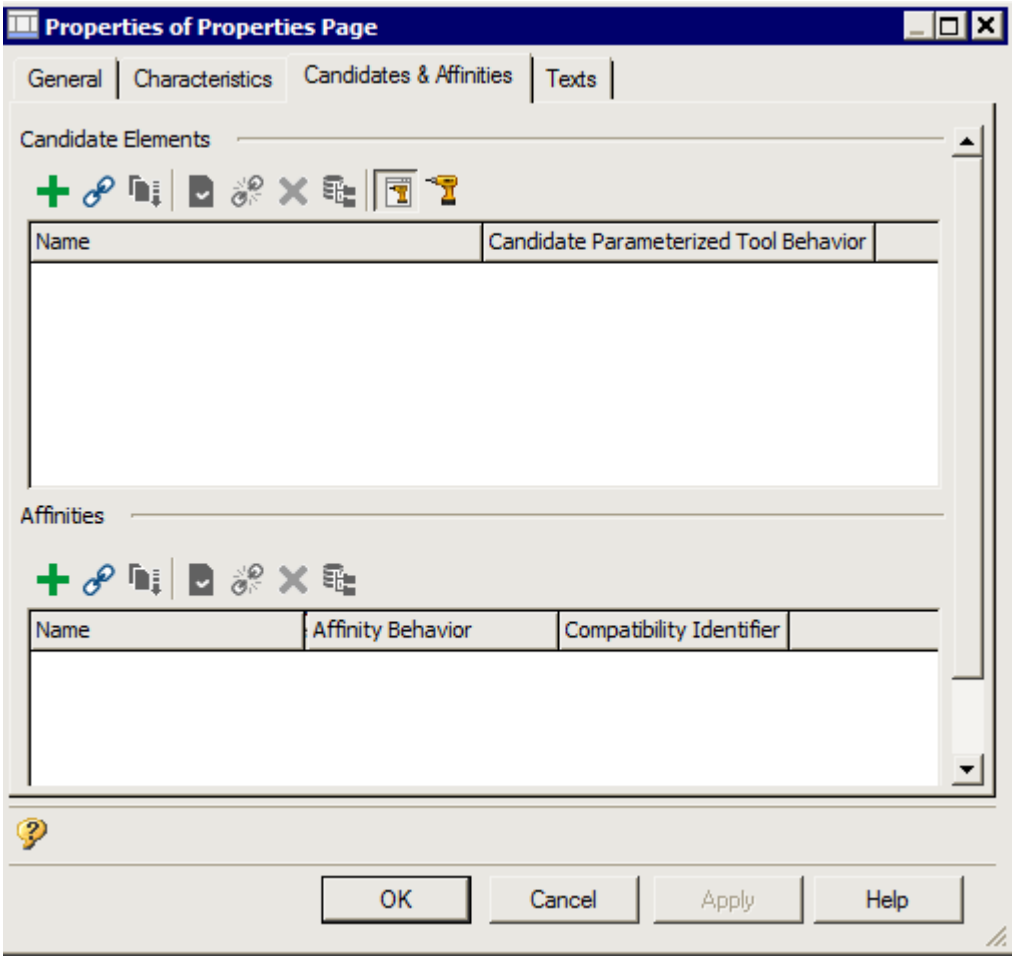
7.6 Defining Container affinities

An affinity enables characterization of a Container. You can for example define that a **Tool** or **MEGA Parameterized Tool** opens in a Container that has an affinity "Affinity Name".

An affinity is only used by code, unlike candidates which are defined in the metamodel. If no code is defined for opening of a tool with affinity, then the configuration defined for candidates is taken into account (see 7.5).

1. Open the **Properties** dialog box of the Container for which you want to define an affinity (example: Desktop Container "Properties Page").

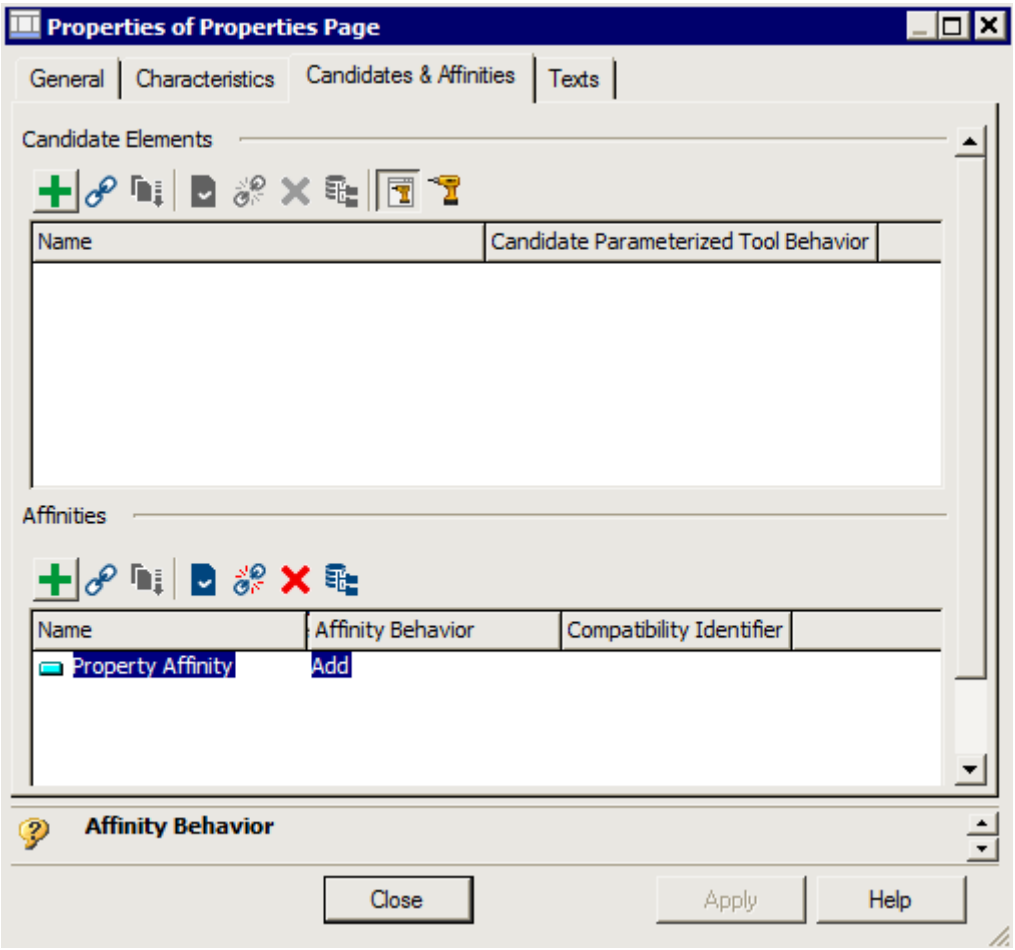
2. Select the **Candidates & Affinities** tab.



3. In the **Affinities** frame, click **New**  (or **Connect**  if the affinity concerned is already created).

4. Enter the Name of the affinity (example: "Property Affinity").

5. (Optional) In the **Affinity Behavior** field, by default the affinity behavior has value **Replace**, in this case the page will open replacing the previous content of the Container. To add rather than replace a Container tab, modify the affinity behavior parameter to **Add**.



6. (Optional) The **Compatibility Identifier** field enables unique definition of a Container. The tool to be opened will therefore always open in the Container that has this specific identifier.

   Example: This can be useful for example to always open analysis reports in a specific tab, and Properties pages in another specific tab.

8 DEFINING DESKTOP CHARACTERISTICS

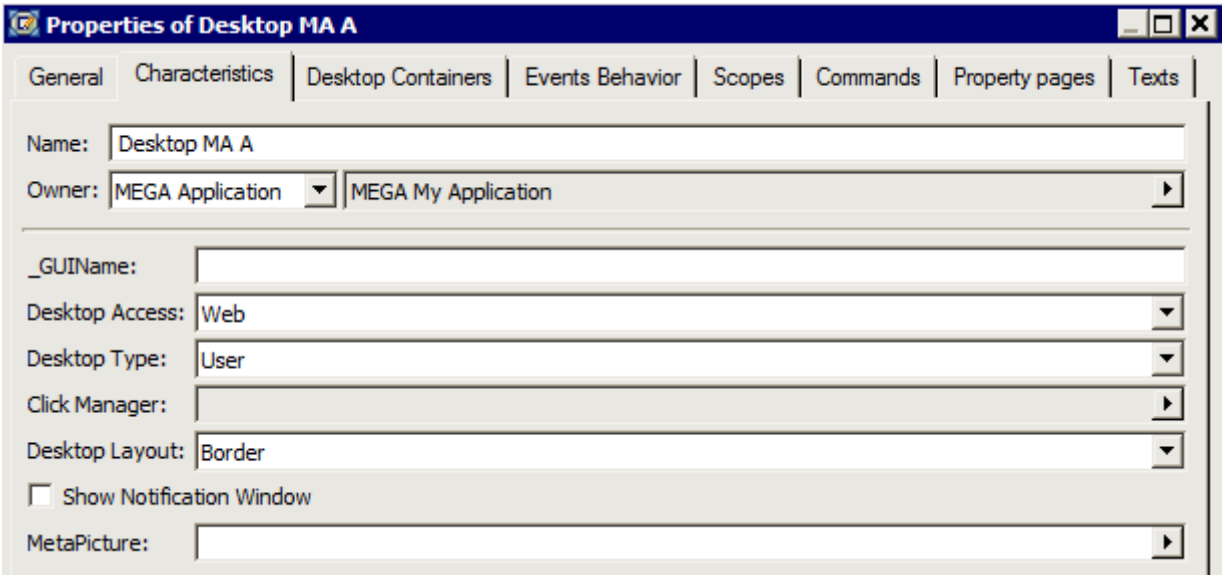
From the desktop Properties dialog box, you can modify default values and configure:

- desktop characteristics
- desktop connection configurations.

8.1 Defining desktop characteristics

To define desktop characteristics:

1. From the **Properties** dialog box of the desktop of which you want to specify characteristics (example: "Desktop MA A").



2. In the **Characteristics** tab, modify or specify the required fields:
   - **_GUIName**: defines desktop name display in the user interface (useful when using **Desktop Switcher** MEGA Tool).
   - **Desktop Access Mode**: defines if application is accessible via Web Front-End or Windows Front-End.

     Default value: "Web Front-End".
   - **Desktop Type**: defines if desktop is user or administrator type. A desktop of administrator type should have more rights and visibility.

     Default value: "User".
   - **Click Manager**: by default a (left) click manages standard current. You can configure alternative behavior of the (left) click on your desktop.

     See Configuring click (left) p. 50.
   - **Show Notification Window**: select this parameter to activate display of notifications at the bottom of the page when you execute an action.

- **MetaPicture** : defines desktop icon display in the user interface (useful when using **Desktop Switcher** MEGA Tool).

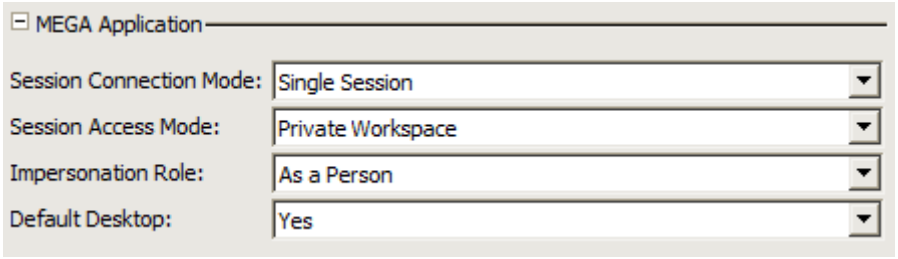
8.2 Modifying desktop connection configurations

A desktop is linked to a MEGA Application. To define connection modes of the desktop of an application, you must open the Properties dialog box of the desktop from the MEGA Application.

You cannot define desktop connection characteristics if you open its Properties dialog box from the query tool.

To modify desktop default connection characteristics:

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application concerned (example: "MEGA My Application").
3. Right-click the desktop concerned (example: "Desktop MA A") and select **Properties**.
4. Select the **Characteristics** tab.



5. In the **MEGA Application** frame, modify the required fields:

   - **Session Connexion Mode**: enables definition of whether users connected to MEGA share the same process or not. In "Multi-Session" mode, users share the same process and therefore have the same view of the repository. "Multi-Session" mode is more optimized but block systemdb repository updates.

     Default value: "Single Session".

   - **Session Access Mode**: enables definition of mode in which application will open.

     "Read Only workspace": the application opens in read-only mode in the current state. Updates are not allowed.

     "Public workspace": the application opens in the current state and data can be modified. All updates are visible by all users using the application at the same time.

     "Private workspace": the application opens in the current state and data can be modified. All updates made by the user are kept in the private Workarea of the user until he/she decides to dispatch them.

     Default value: "Private workspace".

Note: Private workspace is the new wording for transaction.

- **Impersonation Role**: enables definition of whether a user will connect as a person with his/her rights, or as a person group to which he/she belongs with rights associated with the group.

  Default value: "As Person".

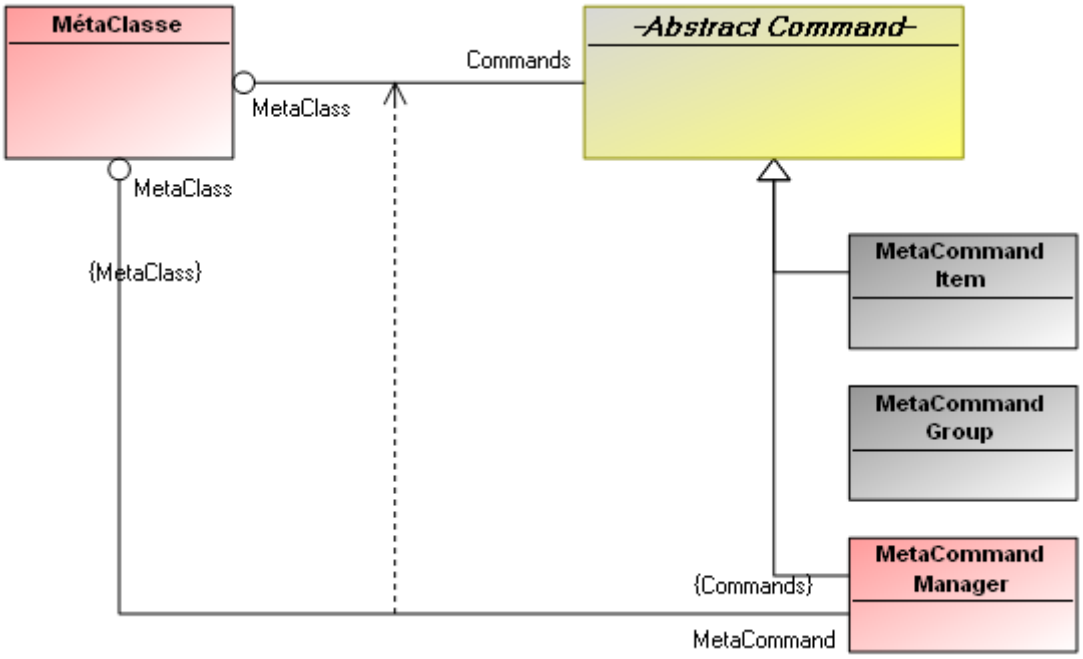
- **Default Desktop**: enables definition of this desktop as default desktop (or not). When you define a default desktop, this avoids having to select a desktop at connection.

  Default value: "No".

9 CONFIGURING THE DESKTOP

By default, no pop-up menu (right-click on object) is available in the desktop.

The MetaModel linked to commands is the following:



9.1 Creating pop-up menus on an object

A pop-up menu is a set of commands (**MetaCommand Manager/MetaCommand Item**) accessible by right-click on a desktop object. Each command is characterized by a Category.

Remarks:

- You can if necessary restrict a pop-up menu to a specific desktop, see Restricting a command to a specific desktop p. 45.

To define pop-up menus accessible (by right-click) from an object (MetaClass) of the desktop, you must define commands accessible from this object (MetaClass).

For each command (of **MetaCommand Item** or **MetaCommand Manager** type) you must define its name (**_GUIName**) and display icon (**MetaPicture**) in the user interface, as well as the **Command Category** to which it belongs. This category is one of the standard categories supplied by MEGA.

The pop-up menu (defined by the command) is classified in a category.

To define a pop-up menu (of MetaCommand Item type) of a desktop object:

1. In the **MEGA Desktop** workspace, run a query on **MetaClass**.

2. Right-click the MetaClass you want to configure (example: **Org-Unit**) and select **Properties**.

3. Select the **User Interface** tab and the **Menu Command** subtab.

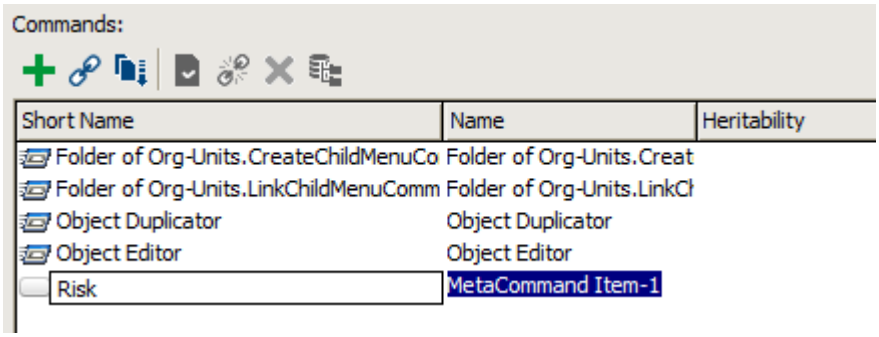
4. Click **New**  .

   ➔ The **Choice of MetaClass Abstract Command** appears.

5. In the **MetaClass** field, select **MetaCommand Item**.

   ➔ The new command appears in the frame listing commands (**MetaCommand Item** and **MetaCommand Manager**).

6. In the **Short Name** field, enter the command name (example: « Risk»).



   ➔ Command characteristics appear.

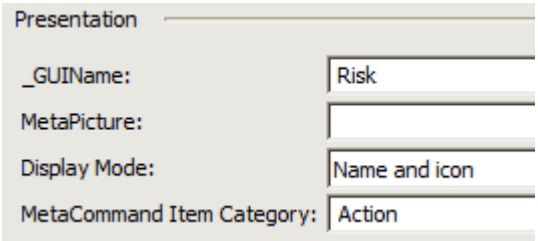
7. In the **Presentation** frame, enter the following fields:

   - in the **_GUIName** field, enter the display name of the command in the user interface.

   - in the **MetaPicture** field, click the arrow and specify the icon of the command.

   - in the **Display Mode** field, select the display mode.

   - in the **MetaCommandItem Category** field, select the command category (example: "Action").

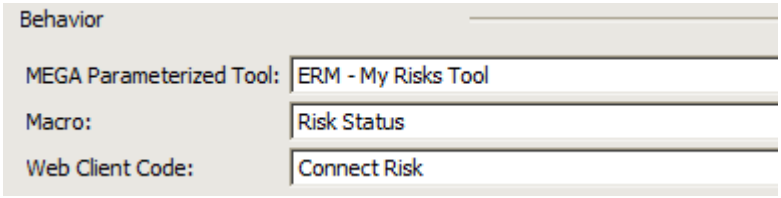


8. In the **Behavior** frame, specify one of the following fields. Click the arrow in field:

   - **MEGA Parameterized Tool** and select the **MEGA Parameterized Tool** that will implement the command (example: "ERM – Risk Consequences").

- **Macro** and select/create the macro that will implement the command (example: "Risk Status").

- **Web Client Code** and select/enter the JavaScript _Code Template (example: « Connect Risk »).



9. Click **OK**.

If you want a pop-up menu command to be specific to a single desktop, see Restricting a command to a specific desktop p. 45.

9.2 Restricting a command to a specific desktop

When a command defined on a MetaClass should be accessible from a particular desktop only, you must define this restriction in the desktop properties.

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application concerned.
3. Right-click the desktop (example: "Desktop MA A") and select **Properties**.
4. Select the **Commands** tab.
5. In the **Specific Command** frame, click **Connect** (or **New** if the command is not yet created) and select the command you want to restrict to this desktop (example: "Desktop MA A").

9.3 Examples of macros

9.3.1 Creating commands of a MetaClass

In the pop-up menu, MetaCommand Manager and MetaCommand Item coexist:

- **MetaCommand Manager** is implemented by a macro with Sub cmdInvoke:

```
Sub cmdInvoke(oBookPart, number)

End Sub
```

➔ For more information, refer to the ***MEGA Studio*** guide.

- **MetaCommand Item** is implemented by a macro with Function invokeOnObject:

```
Function InvokeOnObject(oRoot, sUserData)

End Function
```

9.3.2 Implementing a group of dynamic commands

The following is an example of a group of commands enabling addition of commands dynamically and specification of their behavior when clicked by a user.

- Associate the following macro with MetaCommand Group:

```
'=====================================================================
'Macro : ~j3IwNEVIGTfA[Languages.Implementation]
'---------------------------------------------------------------------
'MegaContext(Fields,Types)
Option Explicit

Class Language
  Public idabs
  Public guiname
  Public picture
End Class

Class ScannerLanguages
  Public mgResource
  Public mgToolkit
  Public mgbTrouve
  Public mgLanguages()
  public nbLanguages

  Public Sub OnItem(Context, Id)
    mgbTrouve = True
    nbLanguages = nbLanguages + 1
    ReDim Preserve mgLanguages(nbLanguages)

    Dim aLanguage
    Set aLanguage = New Language
    aLanguage.idabs = mgToolkit.getString64FromID(Id)
    aLanguage.guiname = mgResource.name(Context, "GuiName")

    Dim mgScannerMP
    Set mgScannerMP = New ScannerMetaPicture
    mgScannerMP.mgbTrouve = False
    mgResource.ScanCollection Id, "~A1mUbldyx840[MetaPicture]", mgScannerMP, 1,
""
    If mgScannerMP.mgbTrouve Then
      aLanguage.picture = mgToolkit.getString64FromID(mgScannerMP.idPicture)
    End If

    Set mgLanguages(nbLanguages) = aLanguage
    'Context.Abort
  End Sub
End Class

Class ScannerMetaPicture
  Public mgbTrouve
  Public idPicture
```

```
  Public Sub OnItem(Context, Id)
    mgbTrouve = True
    idPicture = Id
    Context.Abort
  End Sub
End Class

Function GetLanguages(mgRoot As MegaRoot)
  Dim mgScanner
  Set mgScanner = New ScannerLanguages
  Set mgScanner.mgResource = mgRoot.CurrentEnvironment.Resources
  Set mgScanner.mgToolkit = mgRoot.currentEnvironment.toolkit

  Dim idDepart
  Dim idLink
  Dim listAttributes

  idDepart = "~I9o3by0knG00" '~I9o3by0knG00[Neutral]
  idLink = "~41000000CW30[_Lower]"
  listAttributes = ""
  mgScanner.mgbTrouve = False
  mgScanner.nbLanguages = 0
  mgScanner.mgResource.ScanCollection idDepart, idLink, mgScanner, 1,
listAttributes
  if (mgScanner.mgbTrouve) Then
    GetLanguages = mgScanner.mgLanguages
  else
    GetLanguages = null
  end if
end function

Function count (mo, CommandGroupID, strUserData )
  Dim mgLanguages
  mgLanguages = GetLanguages(mo.GetRoot)
  If (Not IsNull(mgLanguages)) Then
    Dim i
    count = 0
    For i = 1 To UBound(mgLanguages)
      If
mo.GetRoot.CurrentEnvironment.Toolkit.IsAvailable(mgLanguages(i).idabs) Then
        count = count + 1
      End If
    Next
  End If
End Function

Sub CommandEnum(mo, CommandGroupIDParent, vCommandGroupID, oGenContext,
oMenuContext, strUserData)
  dim idabsMaCommande
  dim strTitle
  dim idabsImageMoniker
  dim strTooltip
  dim dwCategory
  dim dwStyle
  dim idabsMacro
  dim idabsParameterizedTool
  dim strToolbarPosition
  dim idabsCodeTemplate
```

```
  dim mgLanguages
  mgLanguages = GetLanguages(mo.GetRoot)

  dim i
  dim res
  If (Not IsNull(mgLanguages)) Then
    For i = 1 To Ubound(mgLanguages)
      If
mo.GetRoot.CurrentEnvironment.Toolkit.IsAvailable(mgLanguages(i).idabs) Then
        idabsMaCommande = mgLanguages(i).idabs
        dwCategory =10
        strTitle = mgLanguages(i).guiname
        idabsImageMoniker = "~" & mgLanguages(i).picture
        idabsMacro = "~m3IwTHVIGDjA" '~m3IwTHVIGDjA[Set Current Language.Macro]
        res = oMenuContext.CommandAdd(mo, "~3uw6D72eErUS[Command]",
idabsMaCommande, strTitle, idabsImageMoniker, strTooltip, dwCategory , dwStyle,
strUserData , idabsMacro, idabsParameterizedTool ,  strToolbarPosition,
idabsCodeTemplate)
      End If
    Next
  End If
end sub
```

This macro uses scanners to browse available languages. But commands essential to MetaCommand Group are CommandEnum and Count, which enable addition of commands dynamically.

- The macro called when a user clicks on any button dynamically creates:

```
'====================================================================
'Macro : ~m3IwTHVIGDjA[Set Current Language.Macro]
'--------------------------------------------------------------------
'MegaContext(Fields,Types)
Option Explicit

'--------------------------------------------------------------------------------
' FUNCTION : InvokeOnObject
' Function called when the command is triggered from a MegaObject.
' Example : click on a menu.
'
' @param mgobjSource
'            MegaObject on which the command is applied.
'--------------------------------------------------------------------------------
Function InvokeOnObject(mgobjSource as MegaObject, sUserData As Variant)
End Function


'--------------------------------------------------------------------------------
' FUNCTION : InvokeOnRoot
' Function called when the command is triggered from a mgRoot.
' Example : click on a menu.
'
' @param mgRoot
'            mgRoot on which the command is applied.
'--------------------------------------------------------------------------------
Function InvokeOnRoot(mgRoot as MegaObject, sUserData As Variant)
  Dim JSON
  Set JSON =
mgRoot.GetRoot.CurrentEnvironment.GetMacro("~j0Iw0fWIGLnA[GetCommandIdFromJSON]
")
```

```
  Dim languageId
  languageId = JSON.getCommandId(sUserData)
  If languageId <> "" Then
    languageId = "~" & languageId
    mgRoot.GetRoot.CurrentEnvironment.SetCurrentLanguage languageId
    InvokeOnRoot = "{refresh :true}"
  End If
  ' The desktop will be refreshed. To display the current language, you can use
the Macro ~52IwefUIGXMA[Get Current Language] that can be displayed in a static
zone tool
End Function
```

This macro operates the change of language based on the command clicked.

- To recover the command id, create a JavaScript macro, which will enable recovery of this information in the form json in the sUserData variable. Its code is the following:

```
'=====================================================================
'Macro : ~j0Iw0fWIGLnA[GetCommandIdFromJSON]
'---------------------------------------------------------------------
//Language:JavaScript

function getCommandId(szJSON) {
  var jsonObject = eval('(' + szJSON + ')');
  var szId = jsonObject.commandId;
  return szId;
}
```

A macro enables addition of commands.

You cannot have groups of commands in groups implemented by a macro.

Example: the macro **Languages.Implementation**

In the example below:

- the **count** function should return >0 for CommandEnum to be called.
- **count** indicates the number of commands that will be added to CommandGroup CommandGroupID.
- in CommandEnum you can add commands to CommandGroupID by calling function CommandAdd of oMenuContext.

```
Idabs = j3IwNEVIGTfA

Sub CommandEnum(mo, CommandGroupIDParent, CommandGroupID,
oGenContext, oMenuContext, strUserData)
  dim idabsMyCommand
  dim strTitle
  dim idabsImageMoniker
  dim strTooltip
```

```
  dim dwCategory
  dim dwStyle
  dim idabsMacro
  dim idabsParameterizedTool
  dim strToolbarPosition
  dim idabsCodeTemplate
Dim res
res = oMenuContext.CommandAdd(mo, "~3uw6D72eErUS[Command]",
idabsMyCommand, strTitle, idabsImageMoniker, strTooltip, dwCategory , dwStyle,
strUserData , idabsMacro, idabsParameterizedTool ,  strToolbarPosition,
idabsCodeTemplate)
End sub

Function count (mo, CommandGroupID, strUserData )

End function
```

9.4 Configuring click (left)

By default a (left) click manages change of standard current. It applies to MEGA occurrences.

You can configure alternative behavior of the (left) click on:

- a desktop, or
- a MEGA Parameterized Tool

To configure a (left) click:

1. Open the **Properties** dialog box of the desktop (example: "Desktop MA A") or of MEGA Parameterized Tool.
2. Select the **Characteristics** tab.
3. In the **Click Manager** field, click the arrow and select **Create** (or **List** if the implementation macro is already created).
4. (Optional) Enter the Name of the click manager.
5. In the **Implementation Macro** field, click the arrow and select **Create Macro** (or **List** if the macro is already created).
    a. In the **Creation of Macro** dialog box, select **Create (VB)Script Macro**.
    b. Click **Next**.
    c. Enter the **Name** of the macro.
    d. (Optional) Select **Reusable** if you want to be able to reuse the macro.
    e. Click **Finish**.

➔ The name of the implementation macro appears in the **Implementation Macro** field.

6. In the **Implementation Macro** field, click the arrow, select the macro you have just defined and select **Edit**.

7. In the script edit dialog box, enter the macro script:

```
Sub GenerateStream(oObject, oContext, sUserData, oStream)
End Sub.
```

- oObject: object to which command is applied
- oContext: defines execution context (Advisor, Hopex, Mega)
- sUserIn: defines any additional information that may be required to define action on object click
- oTextStream returns:
  - the JSON corresponding to the command to be executed, or
  - the JSON after which simple current management is required:

    {setCurrent: true).

Example of click on a tree element:

```
Sub GenerateStream(mgobjSource , oContext , sUserData , oStream)

oStream.Write("{setCurrent:true}")

End Sub
```

- mgobjSource: Mega object to which command is applied
- oContext: defines generation context (Advisor, Hopex, Mega)
- sUserData: defines any additional parameter that may be required to decide to run the action.
- oStream contains the result of JSON to be sent in return to the client
  - a standard JSON is returned for the setCurrent action:

    {setCurrent:true}
  - a dedicated JSON is returned if a command must be executed

The JSON awaited in return of the function is the JSON of a command. To do this, MEGA provides a standard function enabling recovery of the JSON associated with a MetaCommand Item: GetJSONCommand

```
Sub Generate(mgobjSource , oContext , sUserData , sResult )
'Dim oRoot
'Set oRoot = mgobjSource.GetRoot
'sResult = oRoot.GetJSONCommand("CommandIdabs")
End Sub
```

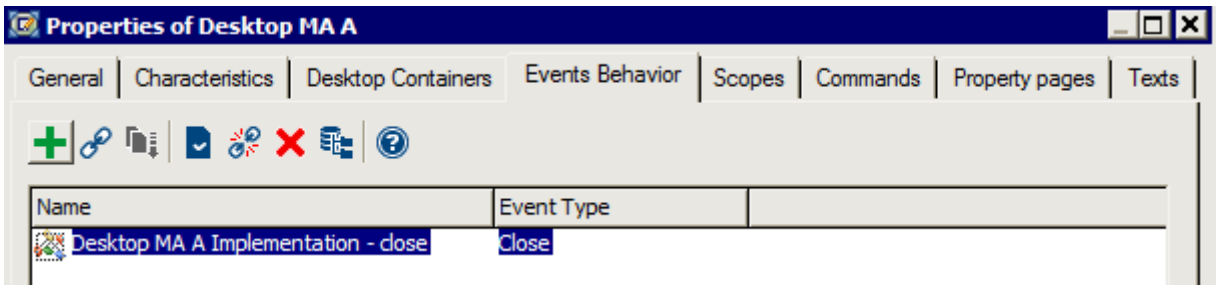
9.5 Configuring behavior on an event

You can configure your desktop to generate a specific behavior at opening, at closing and/or at backup of your desktop.

Example:

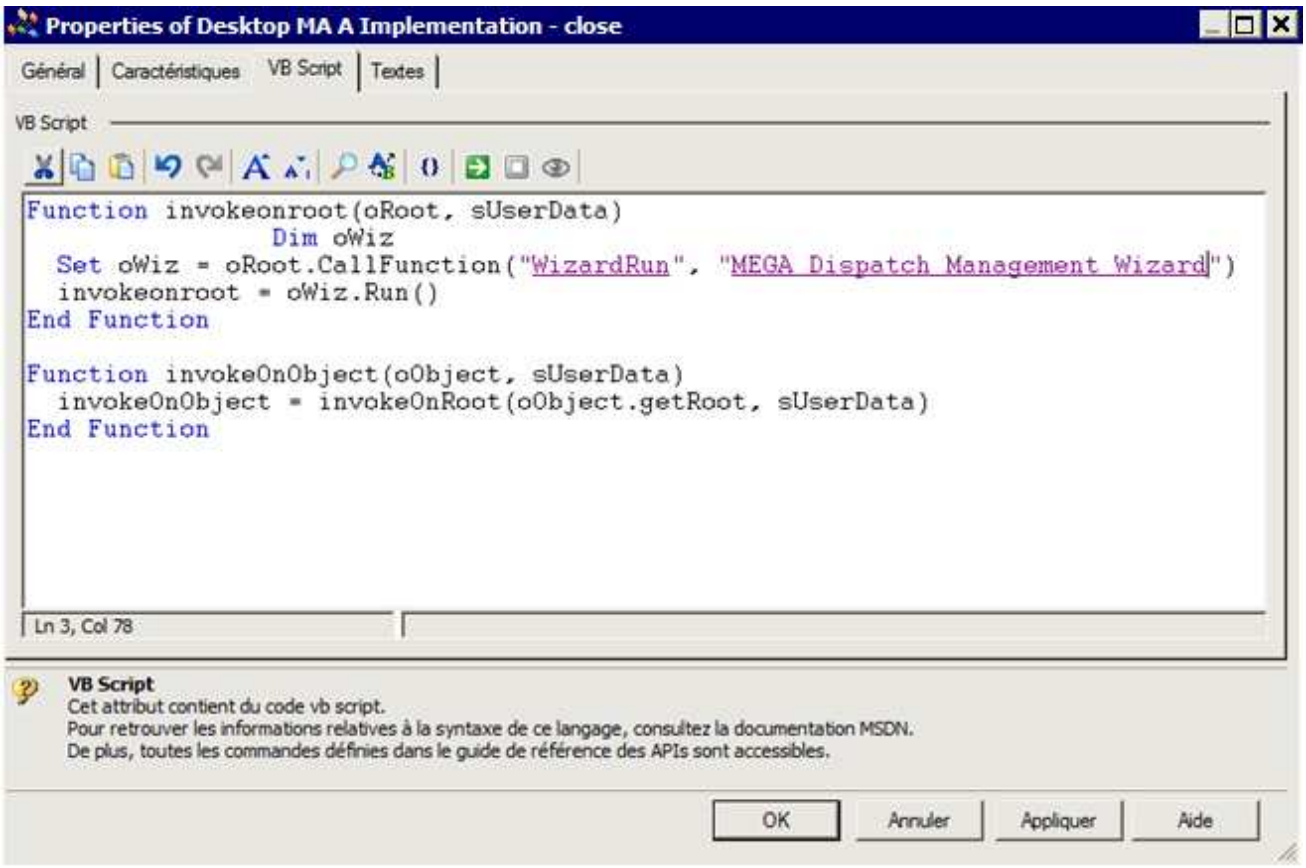
- when first opening the desktop, you can run a questionnaire HTML formatter.
- when closing the desktop**,** you can run a dispatch wizard.

1. Open the **Properties** dialog box of the desktop (example: "Desktop MA A") or of MEGA Parameterized Tool.
2. Select the **Events Behavior** tab.
3. Click **New**  .
4. Select **Manage Macro (VB)Script**.
5. Click **Next**.
6. Enter the **Name** of the macro (example: "Desktop MA A Implementation – Close").
7. Click **Finish**.
8. In the **Event Type** field, select **Close**.



9. Open the **Properties** dialog box of the macro you have just created (example: "Desktop MA A Implementation – Close").
10. Select the **VB Script** tab and enter the code of the macro.

    For example:



"MEGA Dispatch Management Wizard" is the dispatch wizard, of which the identifier is:

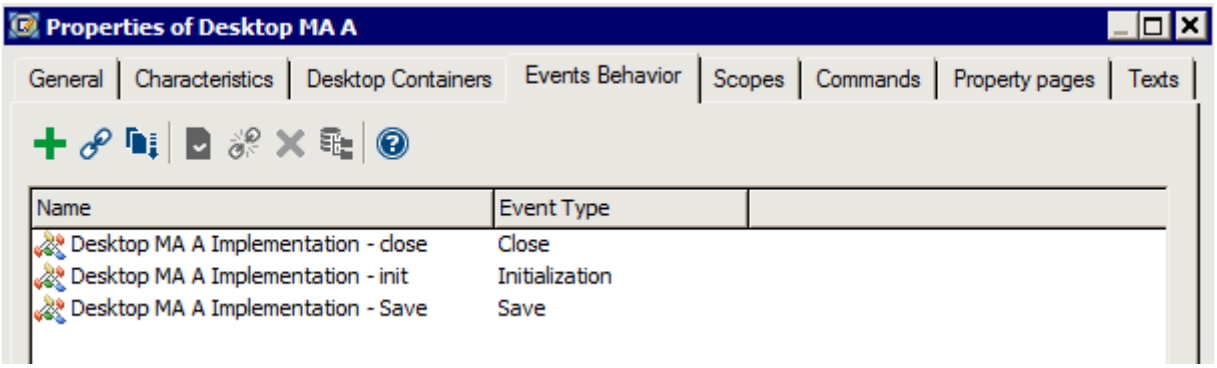
~)pVZ5wu47b00[**MEGA Dispatch Management Wizard**]

11. Repeat steps 3 to 10 to create specific behaviors.

For **Save** Event Types and the initialization macro, the prototype of functions to be implemented is the same:

```
Function invokeonroot(oRoot, sUserData)

End Function

Function invokeonobject(oRoot, sUserData)
End Function
```

Do not omit Function invokeonobject; although empty, it is necessary for operation.



9.6 Creating a toolbar

A toolbar comprises commands, groups of commands and/or MEGA Parameterized Tools. These components can be grouped by themes (tool groups).

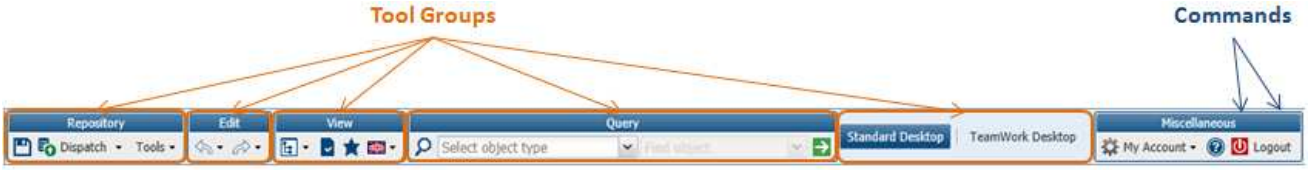


Figure 2 : Example of a toolbar

A tool group is a **MetaCommand Group** (example: "View") which can contain:

- **MetaCommand Groups**
  Example: "Language"
- **MetaCommand Items**
  Example: "Favorites", "Properties"
- **MEGA Parameterized Tools**

  Example: "Navigation Trees Manager"

Figure 3 : Example of a toolbar tool group ("View")

Creating a toolbar consists of creating a **Desktop Container** of **Container of Tool** type. This **Container of Tool** will contain the **Toolbar** desktop component (MEGA Parameterized Tool).

To create a toolbar, you should proceed as follows:

Step	Action	See	P.
1.	Creating your toolbar structure	0	55
2.	Creating toolbar tool groups	9.6.2	59
3.	Configuring toolbar tool group display	9.6.3	60
4.	Defining toolbar tool group commands	9.6.4	62
5.	Configuring toolbar commands display	9.6.5	63

Note: See Command group configuration examples p. 65.

9.6.1 Creating your toolbar structure

To create your toolbar, you must create a **Desktop Container** of **Container of Tool** type.

You will define this **Container of Tool** by a desktop component (**MEGA Parameterized Tool**) of **Toolbar** type, which has a group of commands as its parameters.

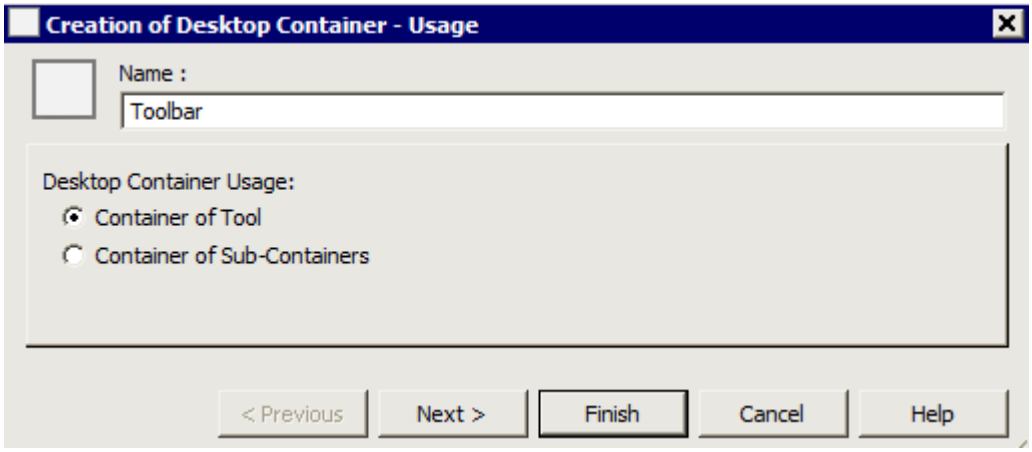
You will create the group of commands (**MetaCommand Group**) representing your toolbar, and define representation of your toolbar (ribbon, ribbon without frame or drop-down menus):

1. From the **Creation of Desktop – Desktop Containers** dialog box (see Creating a desktop), in the **Top Container** (**Left**, **Right**, **Center** or **Bottom**) field, click the arrow and select **New**.

   ➢ The first step of the **Creation of Desktop Container** wizard appears: **Usage**.

2. Enter the **Name** of the Container (example: "Toolbar").

3. Select the type of **Container of Tool** Desktop Container.



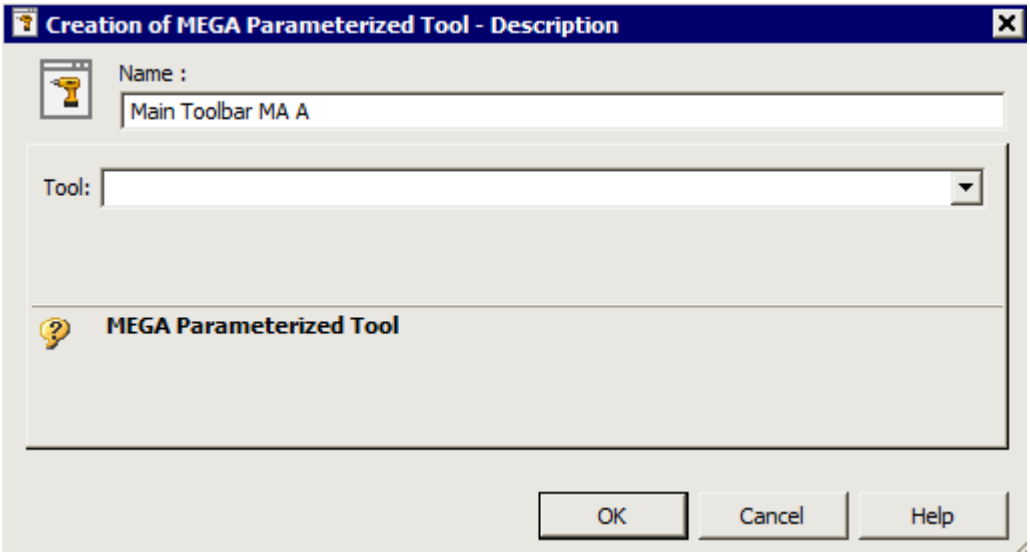
4. Click **Next**.

   ➢ The second step of the **Creation of Desktop Container** wizard appears: **Description**.

5. In the **Parameterized Tool** field, click the arrow and select **Create MEGA Parameterized Tool** (or **List MEGA Parameterized Tool** if the Parameterized Tool is already created).

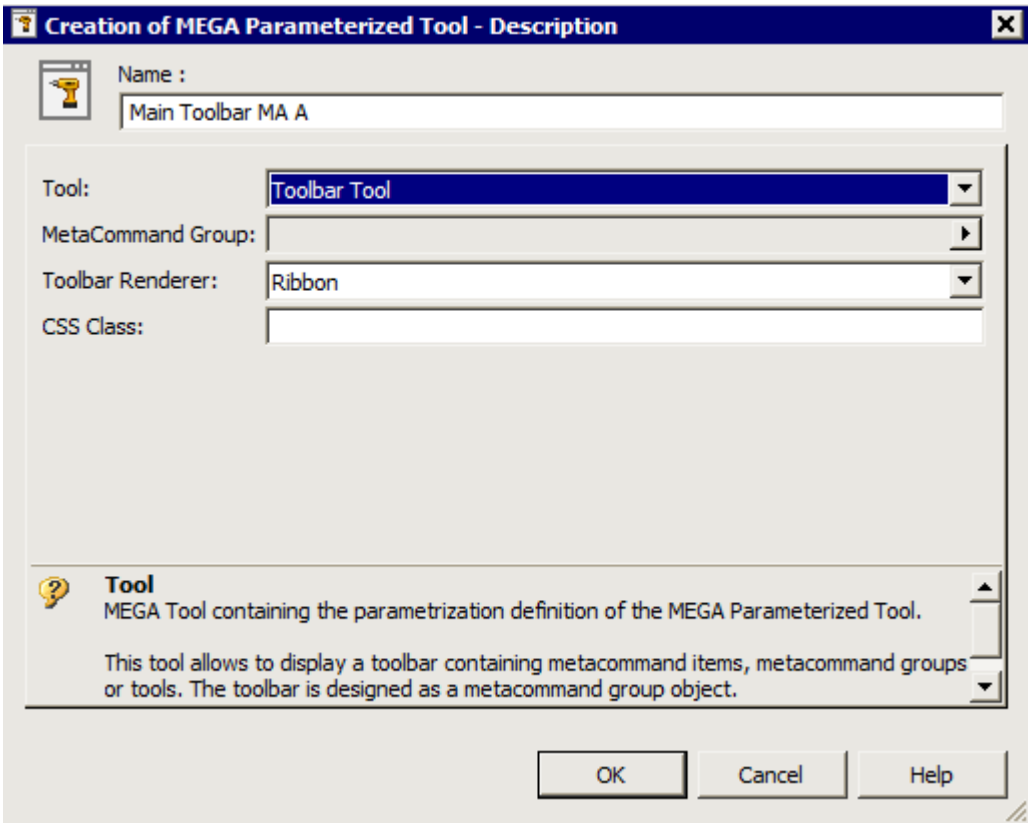
   ➢ The first step of a new **Creation of MEGA Parameterized Tool** wizard appears: **Description**.

6. Enter the **Name** of the desktop/Parameterized Tool component (example: "Main Toolbar MA A").



7. In the **Tool** field, click the drop-down menu arrow and select **Toolbar Tool**.

   ➢ **Toolbar Tool** is displayed in the **Tool** field. Fields **MetaCommand Group** and **Toolbar Renderer** appear.



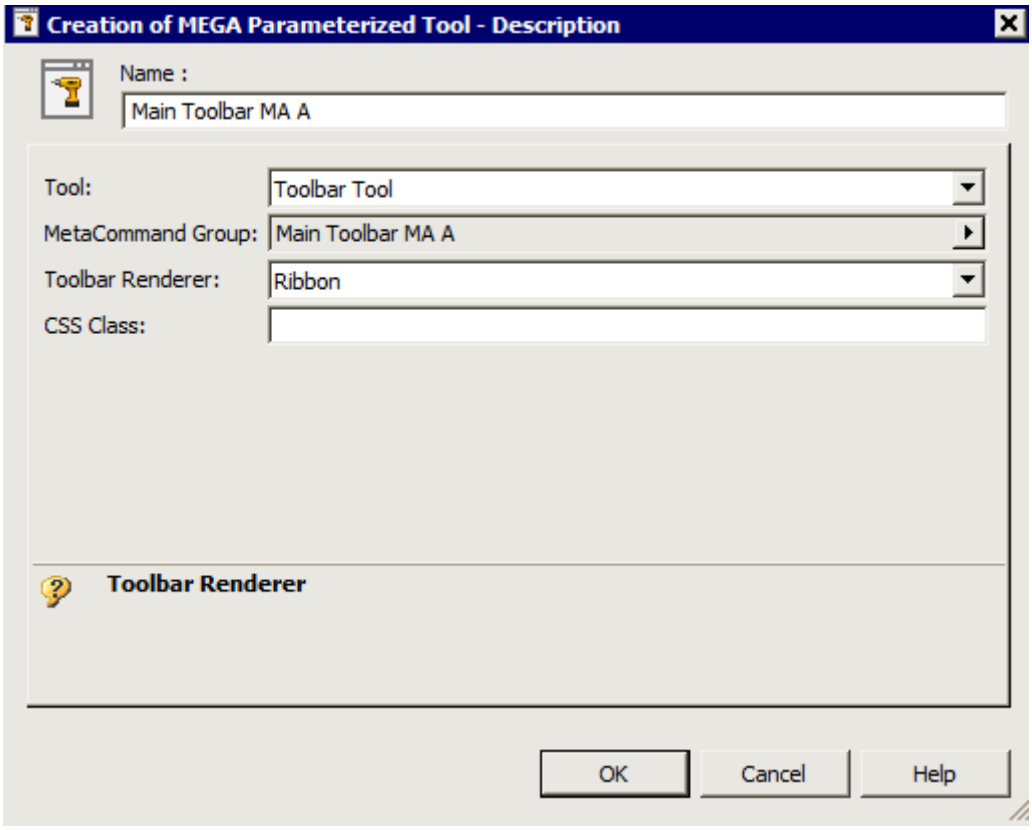
6. In the **MetaCommand Group** field, click the arrow and select **Create MetaCommand Group** (or **List MetaCommand Group** if the required MetaCommand Group is already created).

   ➢ The **MetaCommand Group** creation dialog box appears.

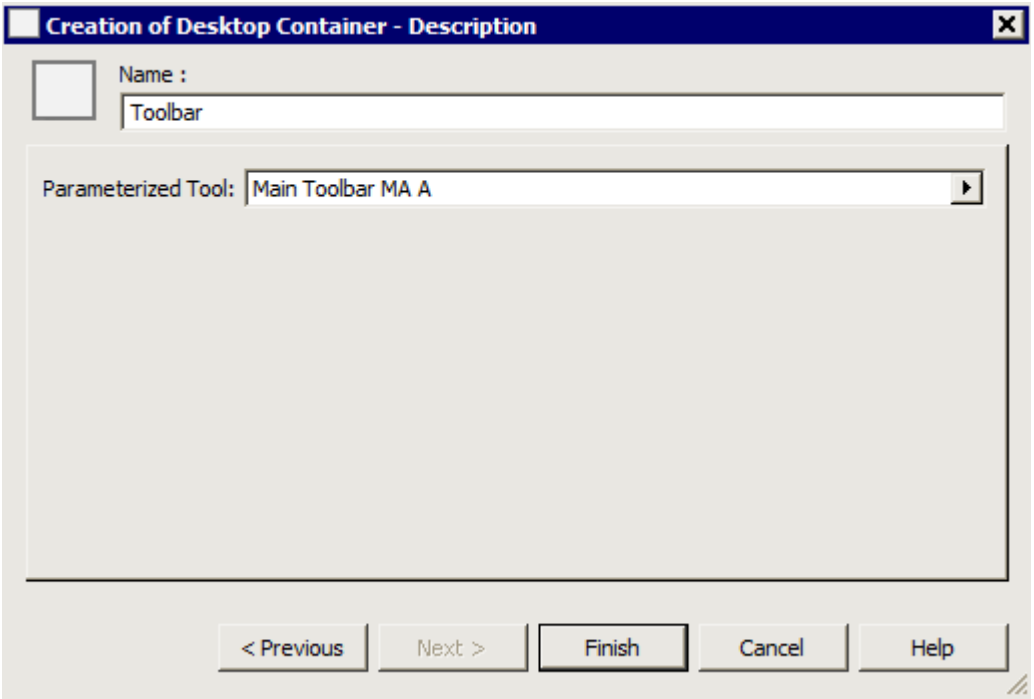
7. Enter the **Name** of the MetaCommand Group (example: "Main Toolbar MA A").

8. Click **OK**.

9. (Optional) In the **Toolbar Renderer** field, you can modify the toolbar type selected by default ("Ribbon").
   The toolbar can be presented in the form of a "Ribbon", a "Ribbon Without Frame" or a "Dropdown Menu.

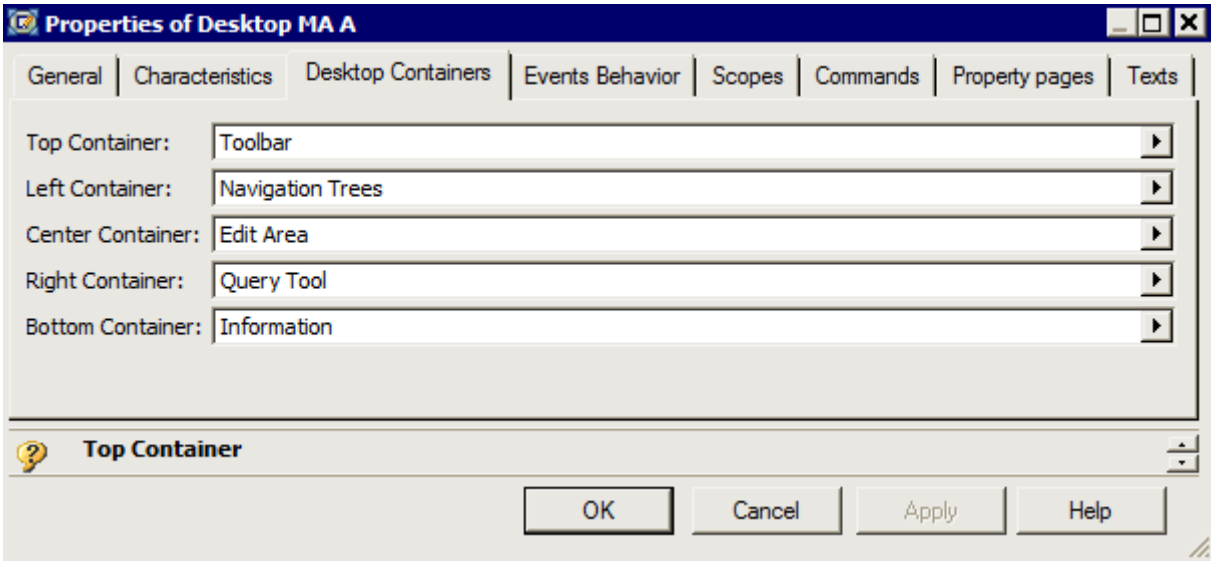


8. Click **OK**.



9. Click **Finish**.

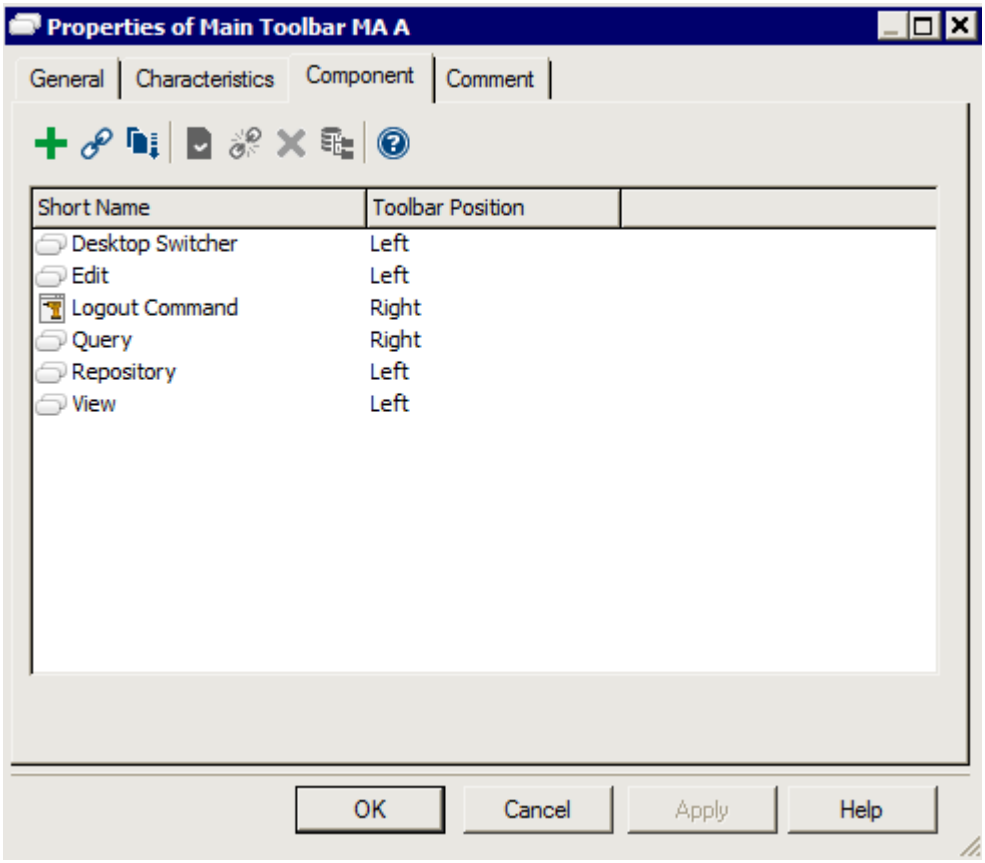
   ➢ The toolbar structure is created.



9.6.2 Creating toolbar tool groups

To create tool groups (MetaCommand Groups) and other components (MetaCommand Item and/or MEGA Parameterized Tool) of your toolbar:

1. Open the properties dialog box of your toolbar (example: MetaCommand Group "MA A Main Toolbar").
2. In the Component tab, click **Create** (or **Connect**).
3. Select the type of **MetaClass** you want to create **MetaCommand Group**, **MetaCommand Item** or **MEGA Parameterized Tool**.
4. Click **OK**.
5. Enter the **Name** of the MetaCommand Group, MetaCommand Item or MEGA Parameterized Tool.
6. Repeat steps 2 to 5 and create (or connect) all the **MetaCommand Groups**, **MetaCommand Items** or **MEGA Parameterized Tools** required.
7. By default, the **toolbar components** are aligned left in the toolbar; to align a component on the right, click in the **Toolbar Position** field of the **MetaCommand Group**, **MetaCommand Item** or **Parameterized Tool**, and select **Right** (example: disconnection command "Logout Command").



The order of presentation of the **MetaCommand Groups** in the list reflects the order of appearance of the tool groups and/or commands in the toolbar.

The value of the position of a **MetaCommand Group** (left/right) takes priority over its appearance order in the list.

To modify organization of tool groups and/or commands, see Modifying position of a tool group in the toolbar p. 75.

9.6.3 Configuring toolbar tool group display

To configure display of a toolbar tool group:

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application and desktop (example: "Desktop MA A").
3. Right-click the desktop tool group (example: "Main Toolbar MA A") and select **Properties**.
4. Select the **Description** tab.
5. In the **MetaCommand Group** field (example: "Main Toolbar MA A"), click the arrow and select **Properties** of **MetaCommand Group** (example: "Main Toolbar MA A").

6. Select the **Component** tab.

7. In the list of components, right-click a tool group (example: "View") and select **Properties**.

8. Select the **Characteristics** tab.

9. In the **Presentation** frame:

   - in the **_GUIName** field, enter the name under which the **MetaCommand Group** will appear in the user interface.

   - if required, in the **MetaPicture** field, click the arrow and select the image of the MetaCommand Group that will appear in the user interface.

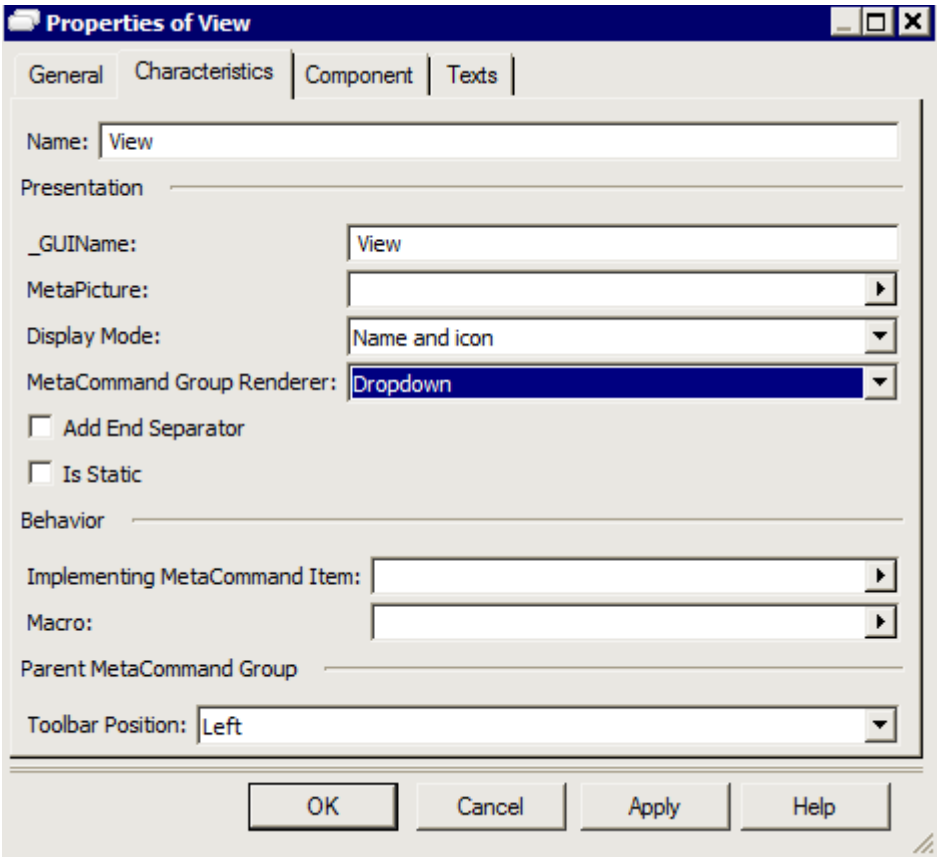
   - In the **MetaCommand Group Renderer** field, select tool presentation: Dropdown (drop-down) or Flat (horizontal).

     Note: this parameter is used essentially for drop-down menus.

   - Select **Add End Separator** if you want to add a separator bar at the end of the tool group.

   - Select **Is Static** to avoid recalculation of the Menu in Web.

10. Click **OK**.



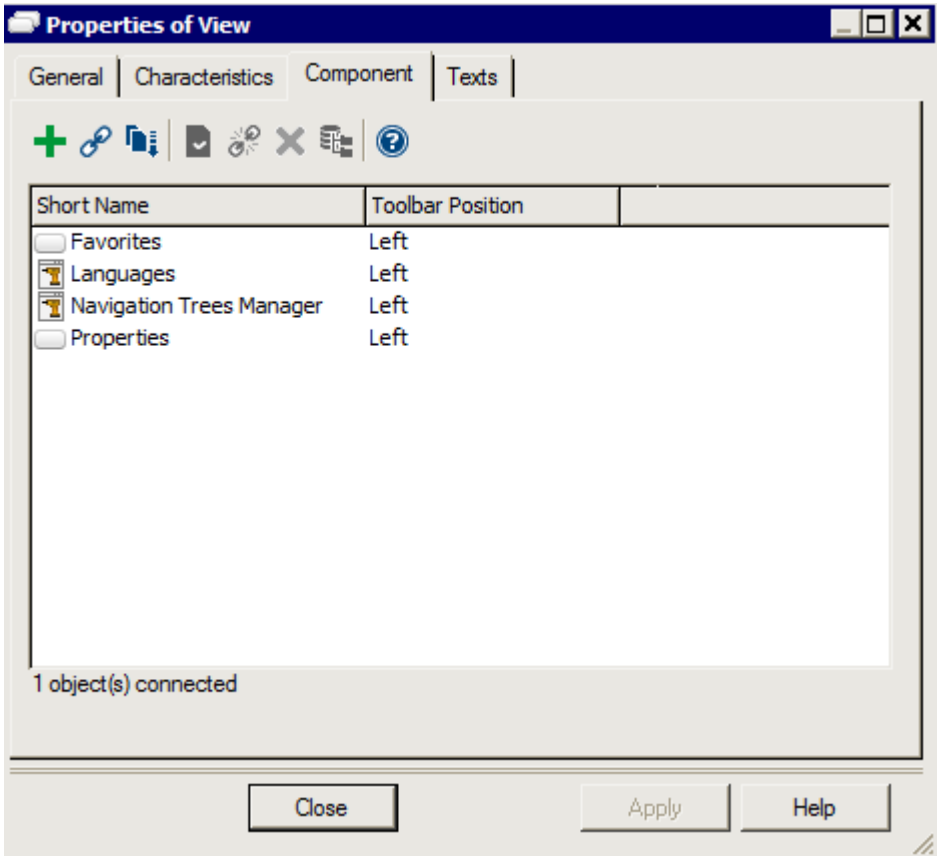
9.6.4 Defining toolbar tool group commands

To define commands of a toolbar tool group:

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application and desktop (example: "Desktop MA A").
3. Right-click the desktop tool group (example: "Main Toolbar MA A") and select **Properties**.
4. Select the **Description** tab.
5. In the **MetaCommand Group** field (example: "Main Toolbar MA A"), click the arrow and select **Properties** of **MetaCommand Group** (example: "Main Toolbar MA A").
6. Select the **Component** tab.
7. Right-click the tool group (example: **MetaCommand Group** "View") and select **Properties**.
8. Select the **Component** tab.
9. Click **Connect**  (or **New**  if the MetaCommand Item, MetaCommand Group or MEGA Parameterized Tool does not yet exist).

   ➔ The MetaClass selection dialog box opens.
10. Select the object type you want to connect. **MetaCommand Item**, **MetaCommand Group** or **MEGA Parameterized Tool**.
11. Click **Find**.
12. Select the object (MetaCommand Item, MetaCommand Group or MEGA Parameterized Tool) you want to connect.
13. Click **OK**.
14. Repeat steps 9 to 13 and create and/or connect all the components of the MetaCommand Group (example: "View").

    Example: Connect the **MetaCommand Items** "Favorites" and "Properties" and the **MEGA Parameterized Tools** "Navigation Trees Manager" and "Languages".



15. To change order of tool group components, see Modifying position of a tool group in the toolbar p. 75.

16. Similarly specify each tool group (**MetaCommand Group**) of the toolbar.

9.6.5 Configuring toolbar commands display

To configure characteristics of commands (MetaCommand Groups):

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application and desktop (example: "Desktop MA A").
3. Right-click the desktop tool group (example: "Main Toolbar MA A") and select **Properties**.
4. Select the **Description** tab.
5. In the **MetaCommand Group** field (example: "Main Toolbar MA A"), click the arrow and select **Properties** of the MetaCommand Group (example: "Main Toolbar MA A").
6. Select the **Component** tab.
7. In the list of toolbar components, right-click a component (example: "View") and select **Properties**.

8. Select the **Component** tab.

9. Right-click the **MetaCommand Item** for which you want to configure display and select **Properties**.

10. Select the **Characteristics** tab.

11. In the **Presentation** frame:

    - in the **_GUIName** field, enter the name under which the command will appear in the user interface.

    - in the **MetaPicture** field, click the arrow and select the image of the command that will appear in the user interface.

    - in the **Display Mode** field, select the display mode (image only, name only, or name and image).

    - Select **Is Separator** to add a separator between commands.

12. Click **OK**.

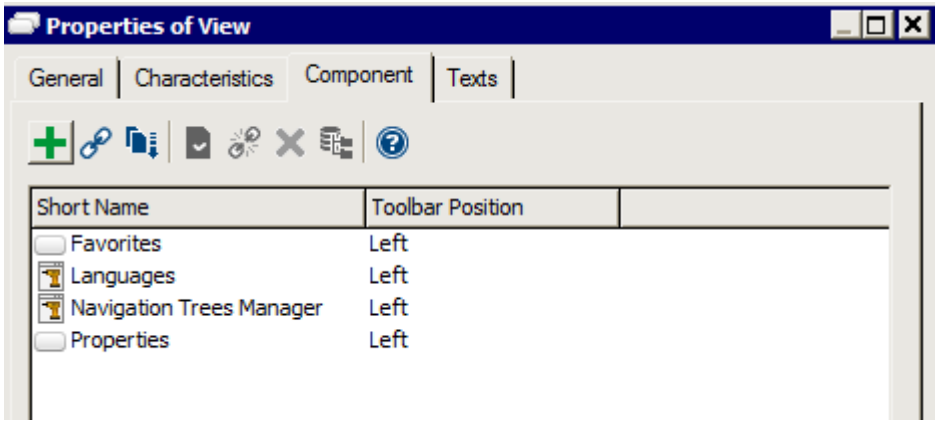
9.6.6 Command group configuration examples

Having defined commands of each group of commands of the desktop toolbar, you must configure each command (**MetaCommand Group**/**MetaCommand Item**).

Example 1: MetaCommand Group containing a **MEGA Parameterized Tool**

To configure a MetaCommand Group (example: "Navigation Tree") of a desktop toolbar:

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application and desktop (example: "Desktop MA A").
3. Right-click the desktop tool group (example: "Main Toolbar MA A") and select **Properties**.
4. Select the **Description** tab.
5. In the **MetaCommand Group** field (example: "Main Toolbar MA A"), click the arrow and select **Properties** of the MetaCommand Group (example: "Main Toolbar MA A").
6. Select the **Component** tab.
7. Select the MetaCommand Group (example: "View") and open its Properties dialog box.
8. Select the **Component** tab.
9. Click **Connect** and select the MEGA Parameterized Tool (example: "**Navigation Trees Manager**") required.



10. Click **OK**.

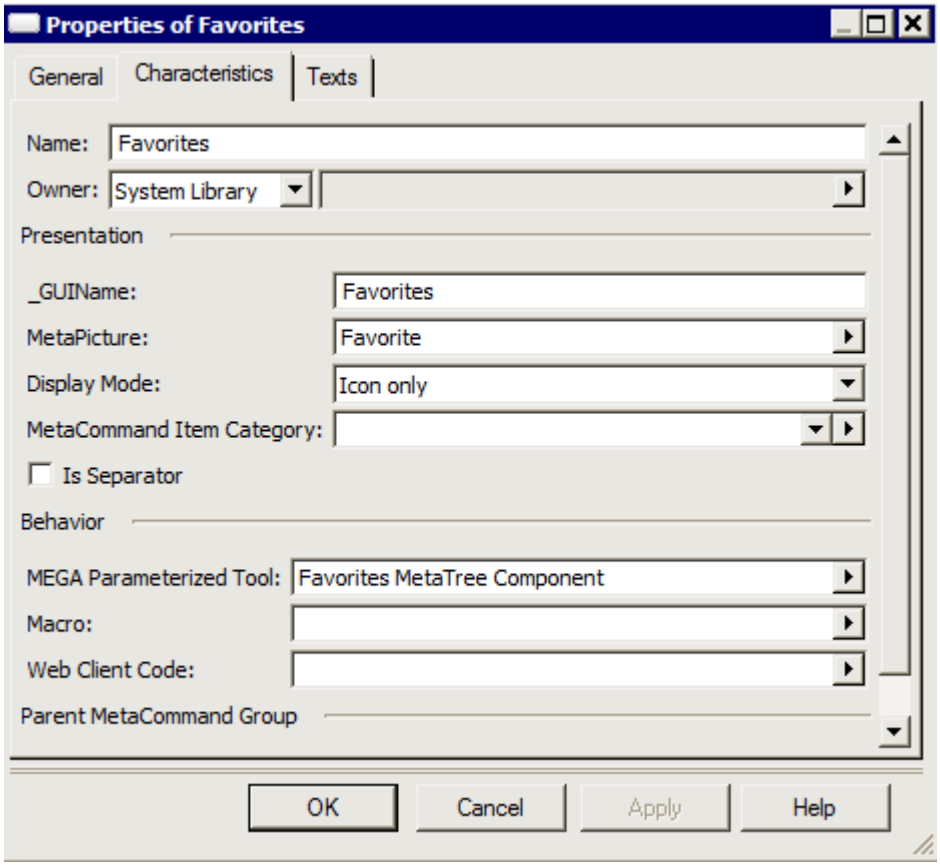
➔ In our case, the Left Container of the desktop is the Accordion type Container presenting navigation trees (see Creating a Desktop Container of Accordion type p. 23).

The **MEGA Parameterized Tool** "Navigation Trees Manager" is connected to Left Container. It shows the tools available in the selected Containers and hides the others. The Container is the tool parameter.

Example 2: MetaCommand Group containing a MetaCommand Item implemented by a MEGA Parameterized Tool

To configure a MetaCommand Item (example: "Favorites") of a desktop toolbar:

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application and desktop (example: "Desktop MA A").
3. Right-click the desktop tool group (example: "Main Toolbar MA A") and select **Properties**.
4. Select the **Description** tab.
5. In the **MetaCommand Group** field (example: "Main Toolbar MA A"), click the arrow and select **Properties** of the MetaCommand Group (example: "Main Toolbar MA A").
6. Select the **Component** tab.
7. In the list of toolbar components, right-click a component (example: "View") and select **Properties**.
8. Select the **Component** tab.
9. Select the MetaCommand Item (example: "Favorites") and open its **Properties** dialog box.
10. Select the **Characteristics** tab.
11. In the **Behavior** frame, click the **MEGA Parameterized Tool** field arrow and select the tool (example: **Favorites MetaTree Component**) required.



12. Click **OK**.

Example 3: MetaCommand Group containing a MetaCommand Item implemented by a macro

To configure a MetaCommand Item (example: "Documents") of a desktop toolbar:

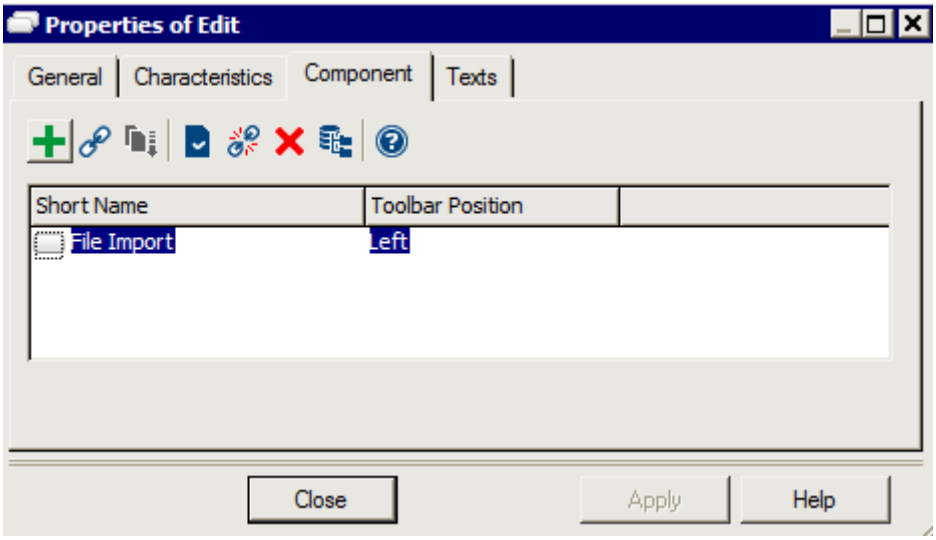
1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application and desktop (example: "Desktop MA A").
3. Right-click the desktop tool group (example: "Main Toolbar MA A") and select **Properties**.
4. In the **MetaCommand Group** field (example: "Main Toolbar MA A"), click the arrow and select **Properties** of the MetaCommand Group (example: "Main Toolbar MA A").
5. Select the **Component** tab.
6. In the list of toolbar components, right-click a component (example: "Edit") and select **Properties**.
7. Select the **Component** tab.

8. Click **New** .

9. In the **MetaClass** selection dialog box, select **MetaCommand item**.

10. Click **OK**.

11. In the **Name** field, enter the command name (example: "File Import").

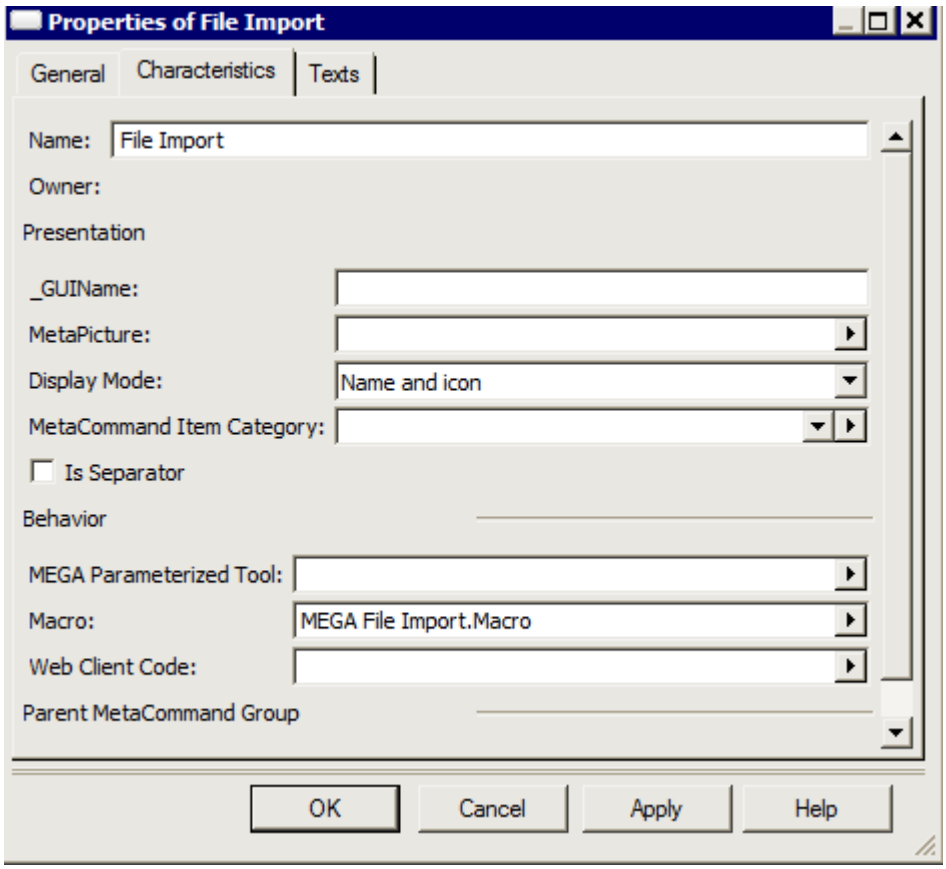


12. Open the Properties dialog box of the command you have just created (example: "File Import").

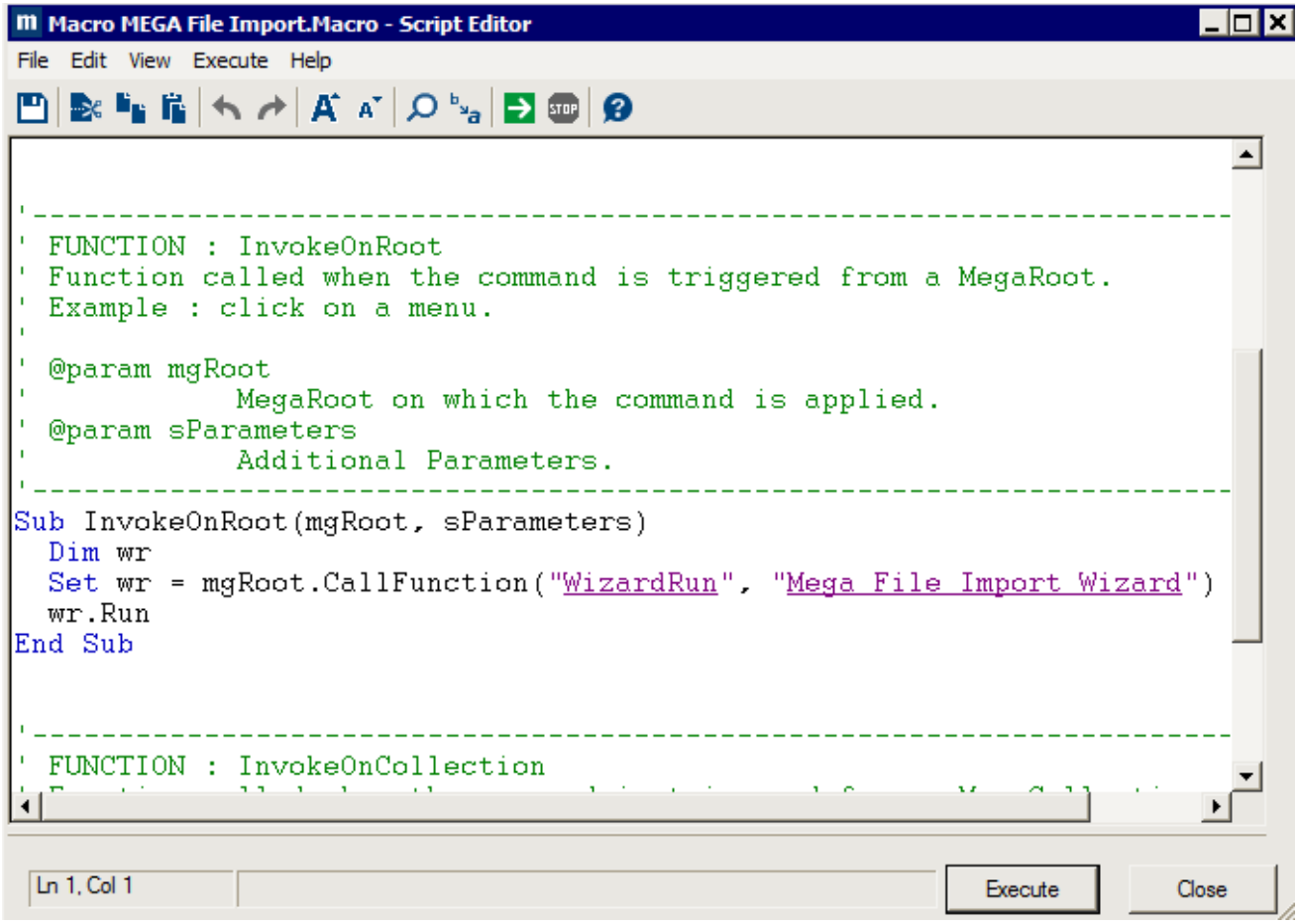
13. Select the **Characteristics** tab.

14. In the **Behavior** frame, click the **Macro** field arrow and select the required macro (example: "MEGA File Import.Macro").

15. Click **OK**.


Properties of File Import
General
Characteristics
Texts
Name:
File Import
Owner:
Presentation
_GUIName:
MetaPicture:
Display Mode:
Name and icon
MetaCommand Item Category:
Is Separator
Behavior
MEGA Parameterized Tool:
Macro:
MEGA File Import.Macro
Web Client Code:
Parent MetaCommand Group
OK
Cancel
Apply
Help

Example: macro "MEGA File Import.macro"



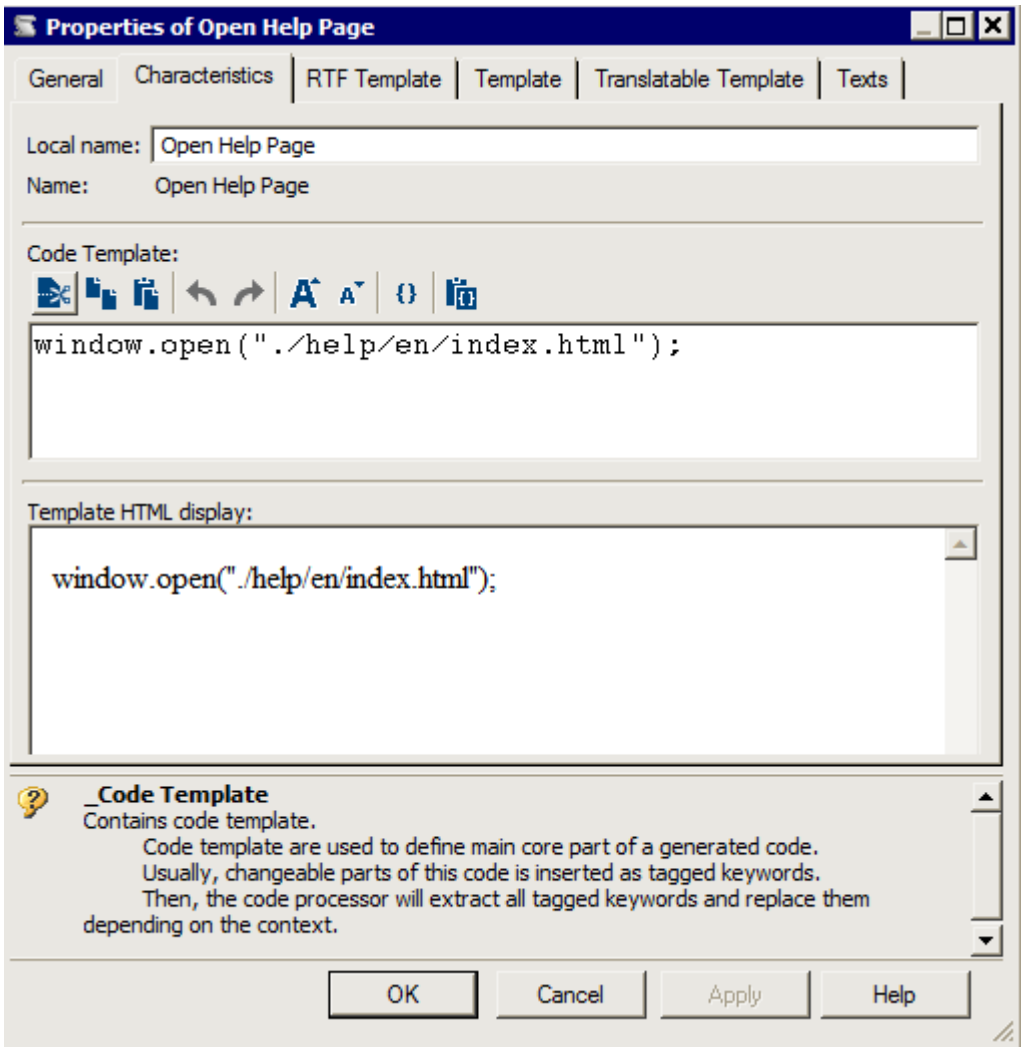
16. Click **OK**.

Example 4: MetaCommand Group containing a MetaCommand Item implemented by JavaScript code

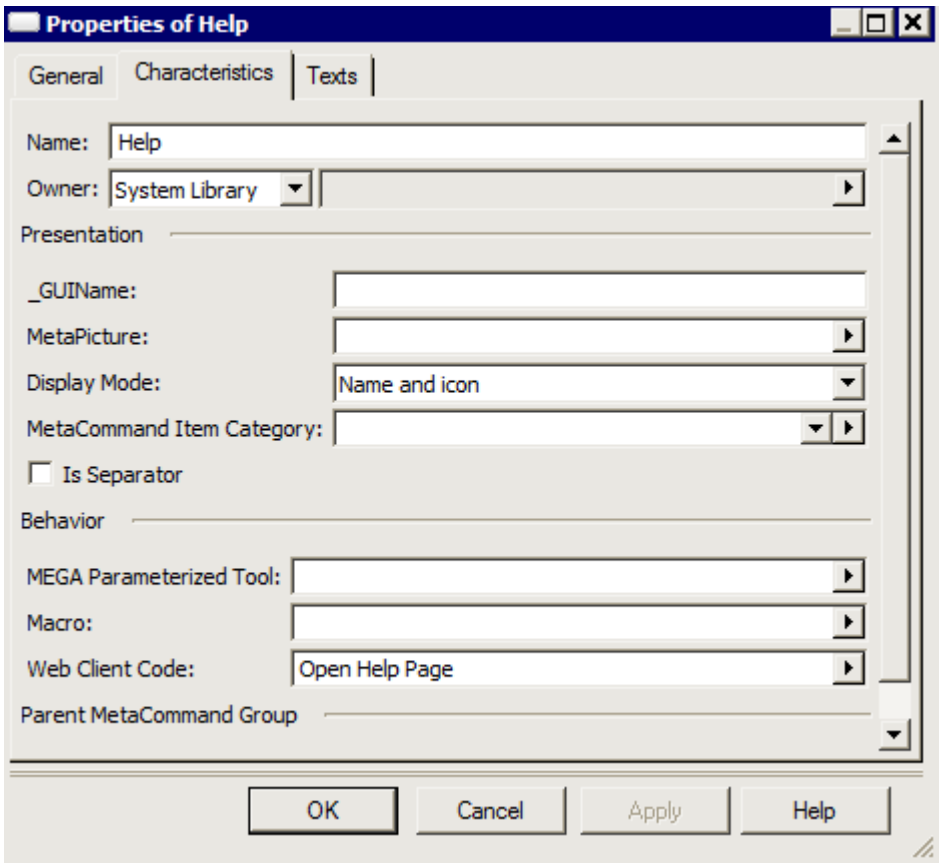
To configure a MetaCommand Group (example: "Miscellaneous") in a desktop toolbar, containing a command implemented by JavaScript code:

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application and desktop (example: "Desktop MA A").
3. Right-click the desktop tool group (example: "Main Toolbar MA A") and select **Properties**.
4. In the **MetaCommand Group** field (example: "Main Toolbar MA A"), click the arrow and select **Properties** of the MetaCommand Group (example: "Main Toolbar MA A").
5. Select the **Component** tab.
6. In the list of toolbar components, right-click a component (example: "Miscellaneous") and select **Properties**.
7. Select the **Component** tab.
8. Click **New**  .
9. In the **MetaClass** selection dialog box, select **MetaCommand item**.
10. Click **OK**.
11. in the **Name** field, enter the command name (example: "Help").
12. Click **OK**.
13. Open the Properties dialog box of the command you have just created (example: "Help").
14. Select the **Characteristics** tab.
15. In the **Behavior** frame, click the **Web Client Code** field arrow and select the _Code Template required (example: "Open Help Page").

Example: _Code Template "Open Help Page"



16. Click **OK**.


Properties of Help
General
Characteristics
Texts
Name:
Help
Owner:
System Library
Presentation
_GUIName:
MetaPicture:
Display Mode:
Name and icon
MetaCommand Item Category:
Is Separator
Behavior
MEGA Parameterized Tool:
Macro:
Web Client Code:
Open Help Page
Parent MetaCommand Group
OK
Cancel
Apply
Help

17. Click **OK**.

10 MODIFYING A DESKTOP

When a desktop has been created, you can modify it.

You can:

- add a **Desktop Container** to the desktop,

  ➔ see Adding a Desktop Container to a desktop p. 74.

- delete a **Desktop Container** from the desktop,

  ➔ see Deleting a Desktop Container from a desktop p. 75.

- modify position of a tool group (**MetaCommand Group**) in the toolbar,

  ➔ see Modifying position of a tool group in the toolbar p. 75.

- modify or customize a **Container**; to do this, you must specify its characteristics.

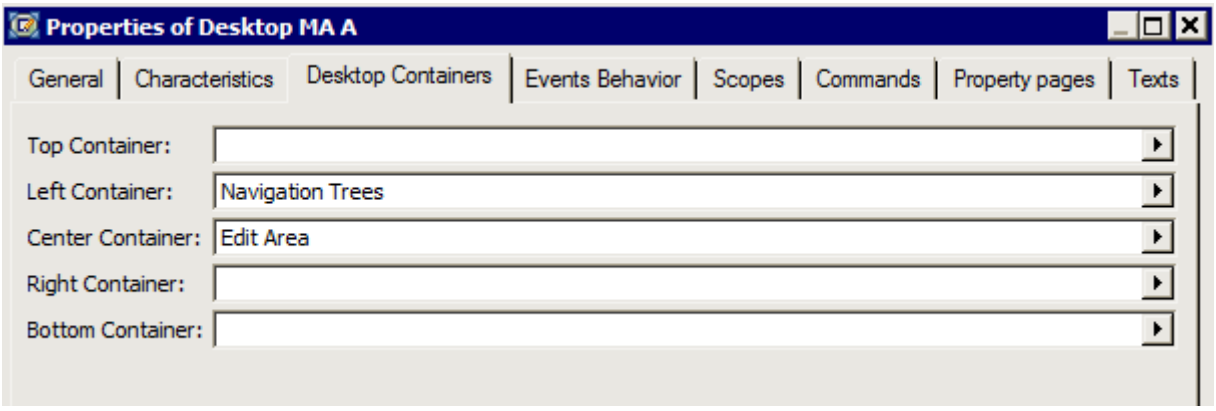
  ➔ see Defining Container characteristics p. 32.

10.1 Adding a Desktop Container to a desktop

To add a Desktop Container to an existing desktop:

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application concerned.
3. Right-click the desktop (example: "Desktop MA A") and select **Properties**.

   ➢ The **Properties of Desktop "Desktop MA A"** dialog box appears.



To add a new **Desktop Container**, see Creating Desktop Containers p. 17 and Step 1, start from the **Properties** of **Desktop** dialog box (instead of from the **Creation of Desktop – Desktop Containers** dialog box).

10.2 Deleting a Desktop Container from a desktop

To delete a Desktop Container from an existing desktop:

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application concerned.
3. Expand the desktop concerned.
4. Right-click the **Desktop Container** you want to delete and select:
   - **Delete** to definitively delete the Desktop Container, or
   - **Disconnect** to retain the **Desktop Container** in the repository.

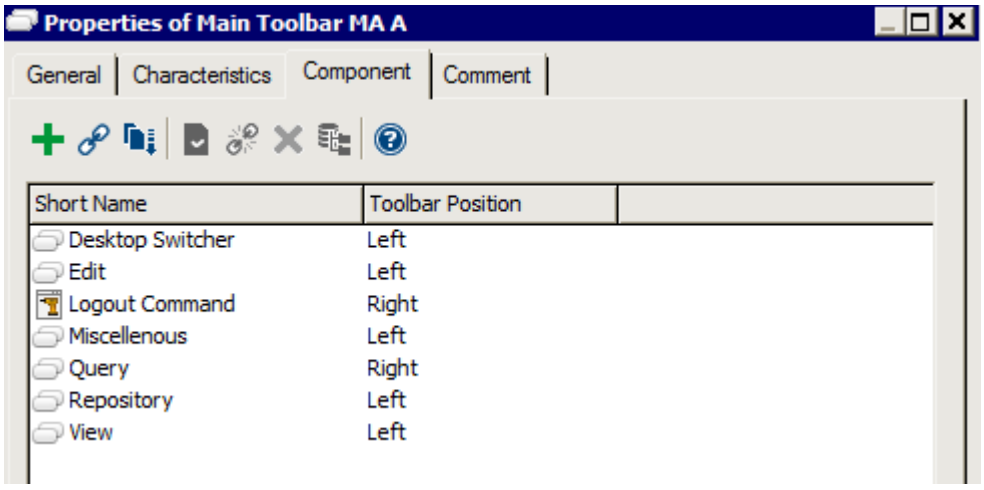
10.3 Modifying position of a tool group in the toolbar

When the toolbar has been created, you can reorganize its tool groups.

The value of the position of an object (left/right) takes priority over its appearance order in the list.

To reorganize the toolbar:

1. In the **MEGA Desktop** workspace, display the **MetaStudio** navigation window.
2. Expand the **MEGA Application** folder, then the application and desktop (example: "Desktop MA A").
3. Right-click the desktop tool group (example: "Main Toolbar MA A") and select **Properties**.
4. Select the **Description** tab.
5. In the **MetaCommand Group** field (example: "Main Toolbar MA A"), click the arrow and select **Properties** of the MetaCommand Group (example: "Main Toolbar MA A").
6. Select the **Component** tab.

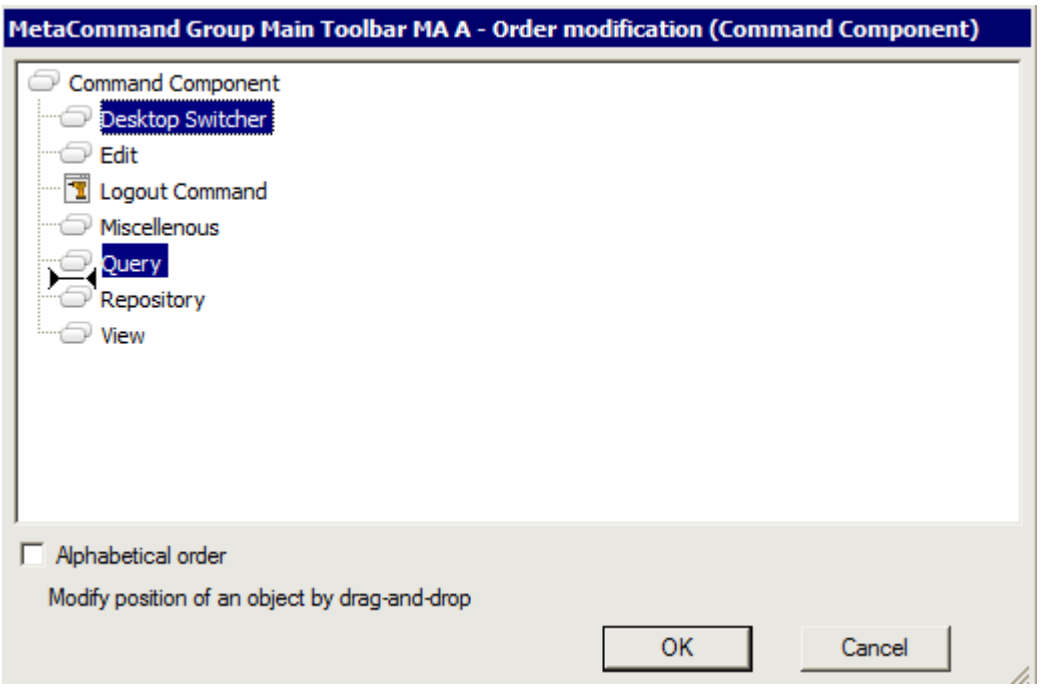


7. Click **Reorder**  .

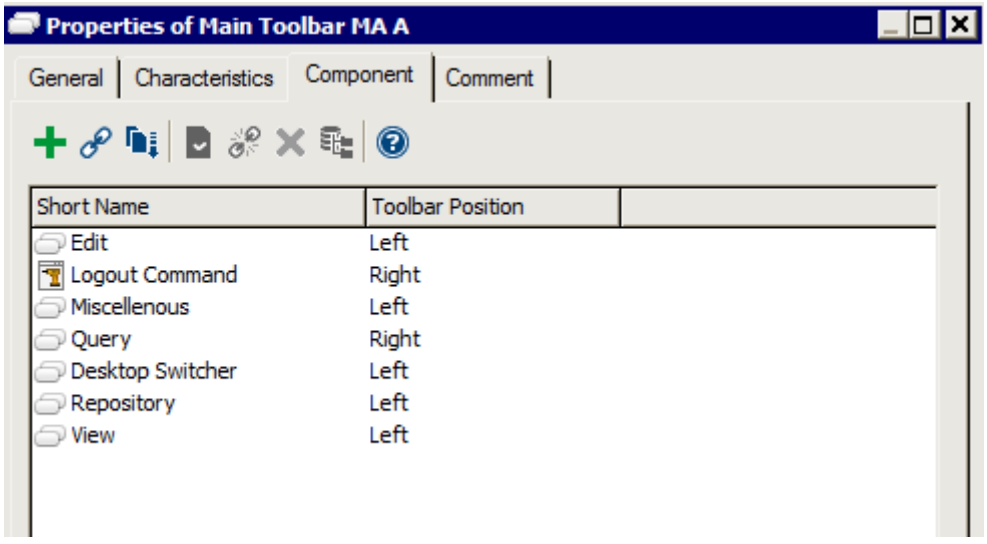
   ➢ The **Order Modification (Command Component)** dialog box appears.

8. Clear **Alphabetic order**.

9. Select the **MetaCommand Group** you want to move and drag it to the required position.



10. Click **OK**.



11 ACTION FOLLOWING EVENT

The desktop can present tools distributed in a given space. Setup of this distribution is not however sufficient for desktop operation: if tools cannot communicate with each other, if there is no interaction between tools, the desktop is unusable.

➢ Tools must be able to communicate between themselves.

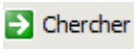
To do this, the desktop introduces two types of events in the application:

- **Current Change**: following an action (click by a user that will change the current) in MEGA, the communication between two tools generates display update of the tool subscriber to the event.

  By default, the current object selected in MEGA is notified to all desktop Containers and Tools. These Containers and Tools update themselves or not depending on this new current.

  Example: When a dialog box is opened, clicking an object generates properties dialog box update as a function of the selected object (unless the action is overridden by the click manager).

- **Interaction**: following an action (click by a user) in MEGA, communication between two tools generates an action of the tool subscriber to the event.

  Example: In the Query tool, clicking 

generates an interaction that requests opening of a dialog box to display search results.

When a (**MEGA Parameterized Tool**) produces an item of information (example: change of state), it can dispatch this event in a Scope (this Scope is saved in the Desktop Container). Only the **Desktop Containers** or **MEGA Parameterized Tools** subscribers to this Scope will take account of this information to update their display as a function of the event (Scope), or execute another action.

11.1 Managing tool update as a function of current

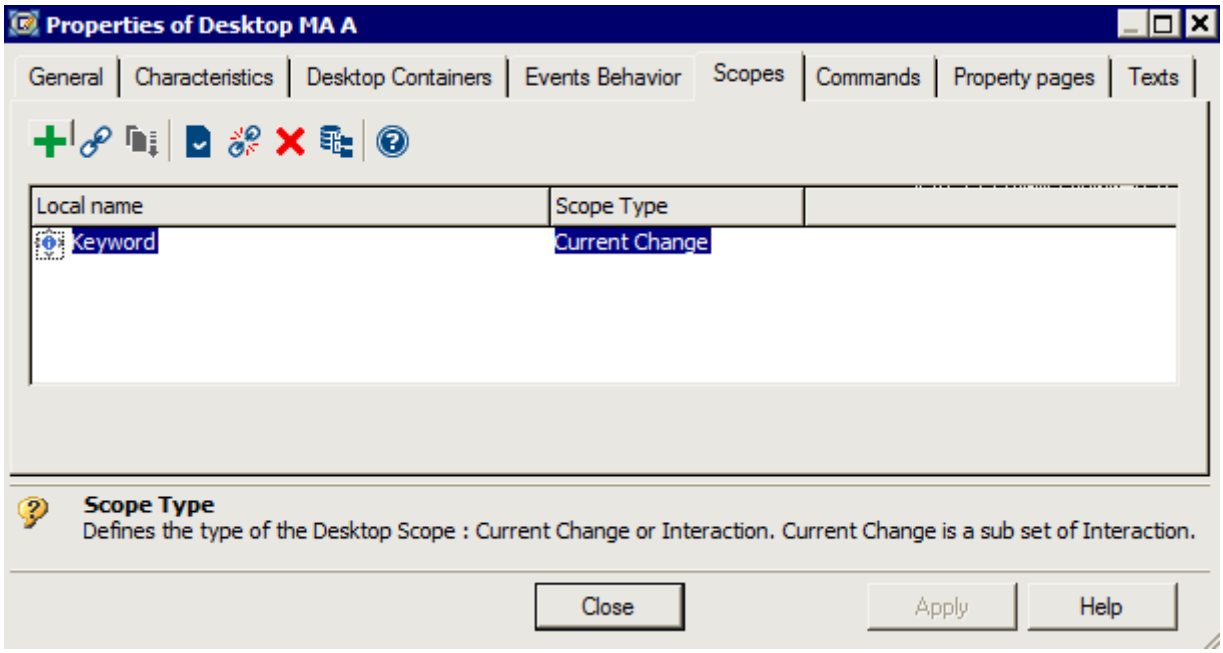
By default the desktop manages global current. This current is automatically sent to all tools, whether they process it or not.

To manage tool update as a function of another current, you must create an event (**Scope**) of **Current Change** type. This event (**Scope**) will be dispatched by a particular tool. Tools sensitive to this current will subscribe to this event (Scope). When the current changes in the source tool, the desktop immediately notifies the target tools , which react if they choose.

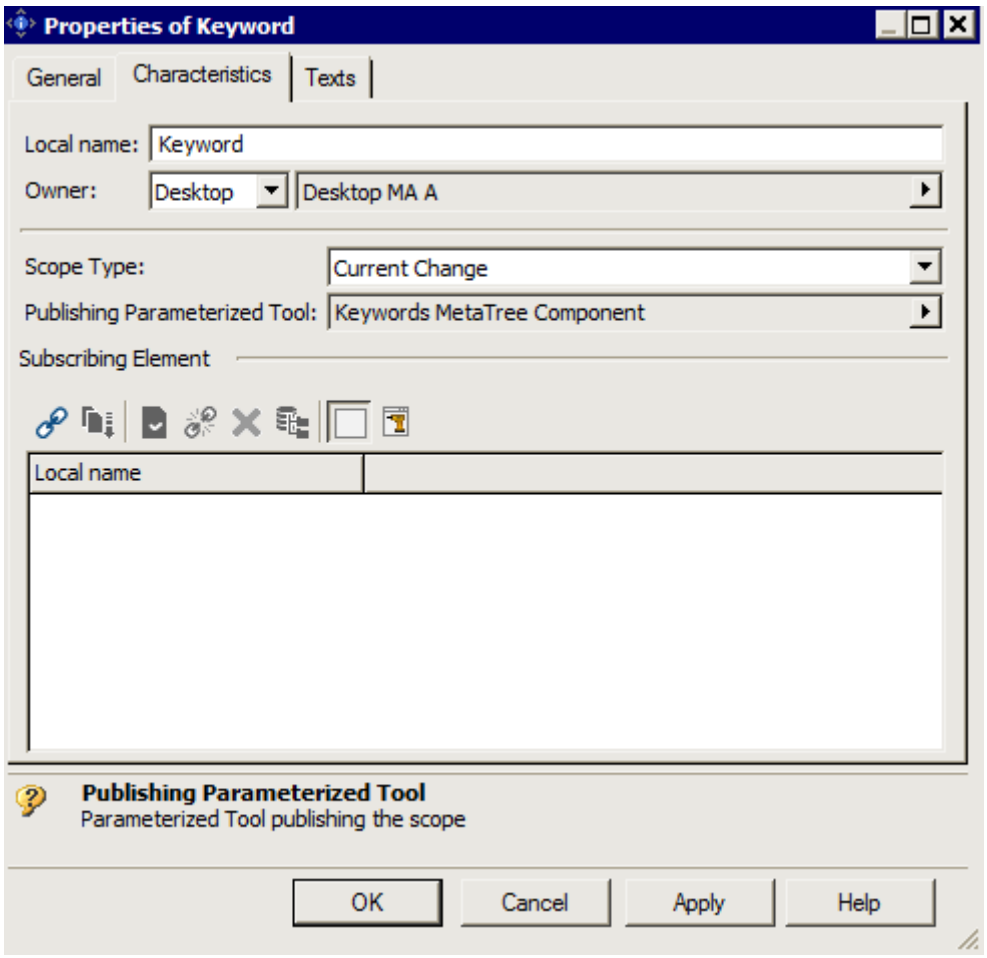
Defining an event (**Scope**) of current is to restrict current change to only two or three tools in the application.

To do this:

1. Open the **Properties** dialog box of the **Desktop Container** (example: "Desktop MA A").
2. Select the **Scopes** tab.
3. Click **New**  .
4. In the **Local Name** field, enter the event name (Scope) (Ex. : "Keyword").
5. In the **Scope Type** field, select the **Current Change** event type.



6. Open the **Properties** dialog box of the new Scope and select the **Characteristics** tab.
7. In the **Publishing Parameterized Tool** field, click the arrow and select the MEGA Parameterized Tool (example: "Keywords MetaTree Component") which will dispatch its modifications.
8. Click **OK**.



9. In the **Subscribing Element** frame, click **Connect** and select the **Containers** and/or **Parameterized Tools** that should be updated as a function of the event (example: modifications of MEGA Parameterized Tool "Keywords MetaTree Component").

10. Click **OK**.

11.2 Managing an action following an interaction

Generating an action following an interaction consists of managing any other event type different from the current. For example so that the action of clicking on a tool generates an action, such as opening a dialog box, you must create an event.

Example: Clicking **Query** generates opening of the query results dialog box and displays the result.

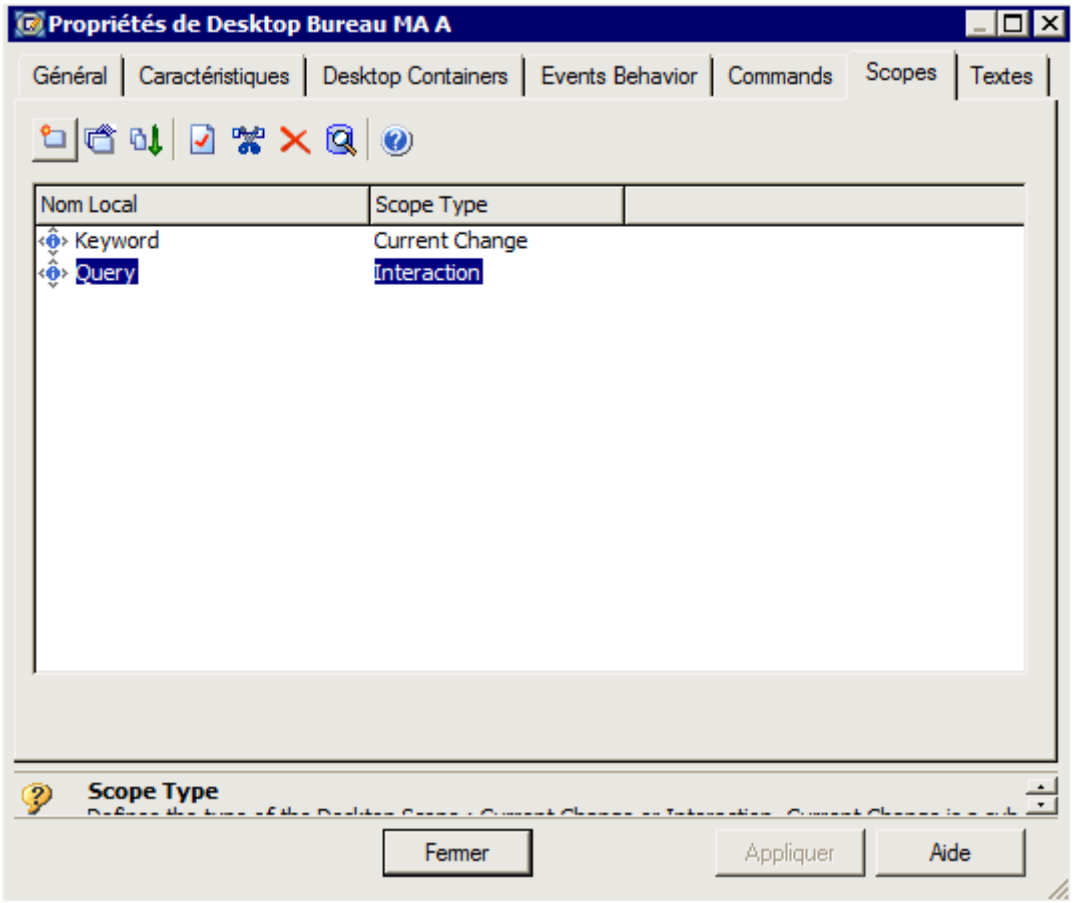
To do this:

1. Open the **Properties** dialog box of the **Desktop Container** (example: "Desktop MA A").

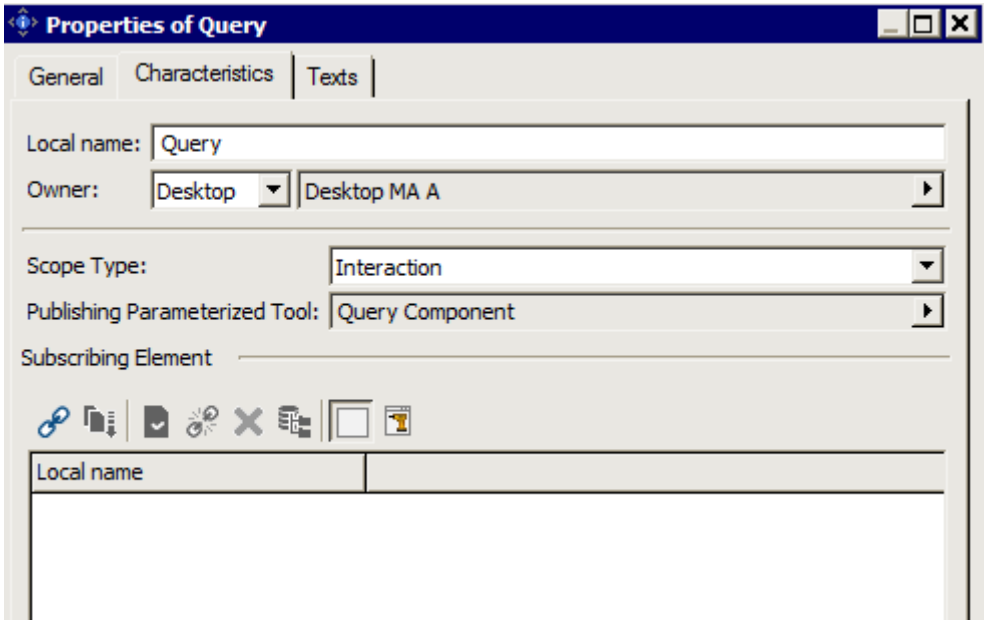
2. Select the **Scopes** tab.

3. Click **New**.

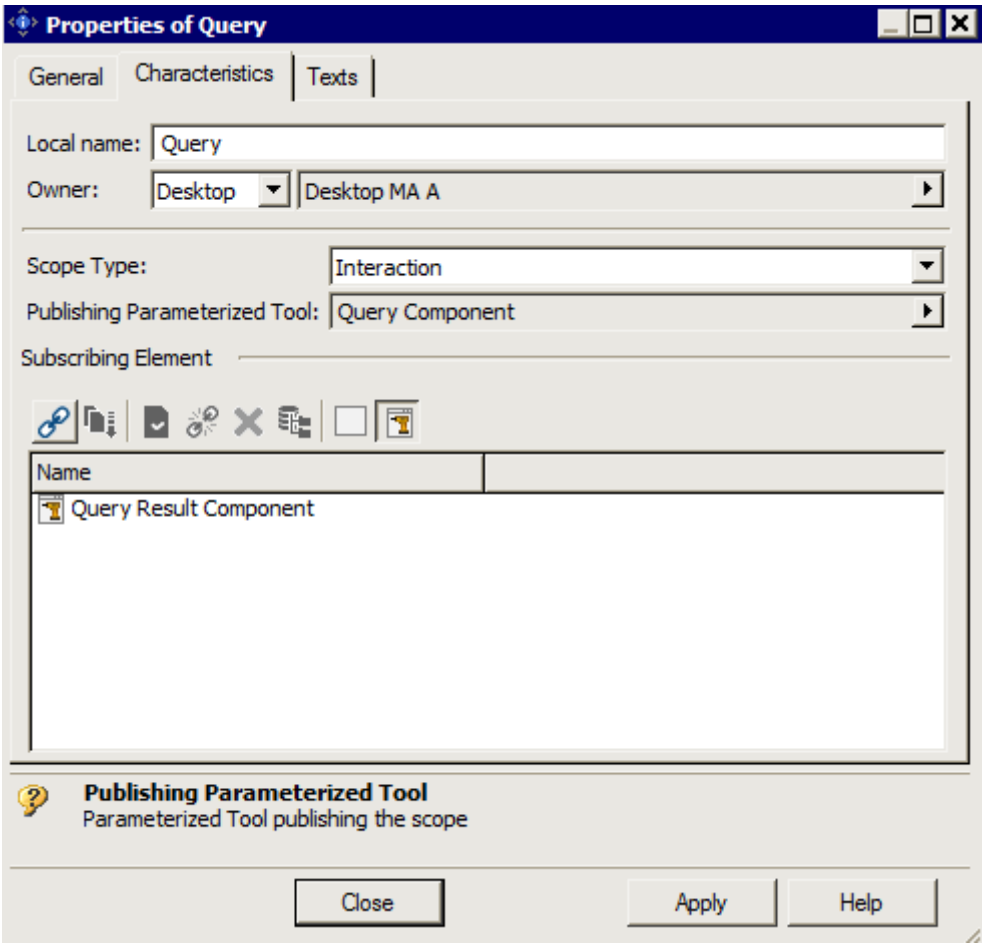
4. In the **Local Name** field, enter the event name (Scope) (Ex. : "Query").
5. In the **Scope Type** field, select the **Interaction** event type.



6. Open the **Properties** dialog box of the new Scope and select the **Characteristics** tab.
7. In the **Publishing Parameterized Tool** field, click the arrow and select the MEGA Parameterized Tool source of the action (example: l'query tool "Query Component").



8. In the **Subscribing Element** frame, click Subscribing **MEGA Parameterized Tool** then **Connect** and select the tool that will present result of the action (example: l'query result tool "Query Result Component").



9. Click **Apply**.

Customizing the Metamodel

Sommaire

QUERY SYNTAX

This section describes syntax used by ERQL (Entity-Relationship Query Language).

Knowledge of this language is useful if you wish to edit repository queries or create your own queries in the **Queries** tab of the query tool.

The following points are covered here:

QUERY GENERAL SYNTAX: SELECT

Query notation

Query uses the following notation:

- An expression within square brackets [ ] is optional.
- An expression within brackets ( ) can be repeated.
- Expressions within braces { } are alternative choices.
- The ERQL operators are in bold characters.

Query structure

The syntax of a query is structured as follows:

```
Select MetaClass from @set Into @Result Where Condition
```

You must use the operators "From", "Into" and "Where" in the order indicated in the example above.

Queries are not case-sensitive. Object types and attributes can be entered with or without accents.

☛ *MetaClasses containing blanks must be entered within square brackets ([ ]).*

Query comments

You can insert comments into commands by starting these with "/*" and finishing with "*/".

```
Example: select Message /* Query for messages */
```

Query settings

You can use settings to specify conditions for a query. When you execute this query, a dialog box asks you to enter these settings, with a box for each setting defined in the query.

Setting names are preceded by &. The setting name is used as the title for the field you are asked to complete when specifying the setting value.

In the syntax description, "&setting" indicates that you can define a setting.

```
select Message where type-message = &Type
```

When the setting is processed, you will be prompted for the value of the "Type" field.

The setting name can contain any character if it is between " ".

```
"&Org-Unit Name"
```

Queries containing several selects

You can use several "Select" clauses in the same query. "Select" clauses follow the same rules as queries that have only one clause. The only difference is that the name specified after the "From" operator can match the result ("Into") of a previous "Select" clause.

This allows you to use intermediate results without having to save them.

Note that the intermediate results are not saved; only the last result is displayed on completion of query execution.

☺ *You can put "Select" clauses on different lines. When entering the query, you can insert a line break by pressing the <Ctrl> and <Enter> keys simultaneously.*

Examples:

- Find the material flows of a project:

```
Select Message into @MaterialFlow where Flow-Type =
"Material Flow"

Select from @MaterialFlow where Diagram.Project = &project
```

- Find the messages of the project diagrams that describe an org-unit:

```
Select Diagram into @Org-UnitDiagram where Described-Org-
Unit and Project = &project

Select Message where Diagram in @ DiagramOrg-Unit
```

List of Query Operators

And: Combines two query criteria.

```
Select Message where Message-Type = "External Data" and
Flow-Type = "Financial Flow"
```

Between: Specifies that a value must be within the range defined by the two given values.

```
Select message Where [Transfer Cost] between 1 And 20
```

Deeply: Performs a recursive query for all objects concerned by a composition link.

```
Select Org-unit where Aggregation-Of deeply = &Org-unit
```

Delete: Deletes a previously saved set.

```
Delete @MainOrg-Units
```

From: Restricts the query to a previously defined set.

```
Select Message from @TradeFlows where Flow-Type = "Material
Flow"
```

Having count: Restricts the query to objects with the indicated number of linked objects.

```
Select Org-Unit where Message-Sent having count > 3
```

In: Queries for values in a set or a sub-query.

```
Select Diagram where Org-Unit in @MainOrgUnits
```

Inherited: All links of the query will include objects inherited via this link.

```
Select [IT Service] Inherited Where [Defining-Application]
= 'Myapplication V2.0"
```

Into: Stores the query result in a set for later use.

```
Select Org-Unit into @MainOrgUnits where not Aggregation-Of
```

Keep: Keeps a set for the whole session, or until it is deleted with the Delete operator.

```
Keep @MainOrgUnits
```

Like: Specifies that the name of the queried objects must match the given value, which can include wild cards such as # and !

```
Select Org-Unit where name like "!!!!!!!!Managem#"
```

Not: Negates a query criterion.

```
Select Message where Flow-Type not = "Financial Flow"
```

Null: The objects queried must not be linked to other objects by the specified link.

```
Select Org-Unit where Message-Sent null
```

Or: One of two query criteria can be true.

```
Select Message where Message-Type = "Instruction" or Flow-
Type = "Financial Flow"
```

Select: Query command.

```
Select Message where Flow-Type = "Financial flow"
```

Unique: Queries for objects linked to only one other object by the specified link.

```
Select Org-Unit where Message-sent unique
```

Where: Specifies the query criteria.

```
Select Message where Flow-Type = "Financial flow"
```

: Wildcard: matches any number of characters.

```
Select Org-Unit where name like "Account#"
```

!: Wildcard: matches any single character (use one "!" for each character).

```
Select Org-Unit where name like "!!!!!!Management"
```

&: Precedes a variable name.

```
Select Diagram where Org-Unit = &Org-Unit
```

@: Precedes a set name.

```
Select Diagram where Org-Unit in @MainOrgUnits
```

: Separation character enabling specification of target MetaClass when browsing a generic link.

```
select Message where [Message Recipient]:[Operation] =
&Operation
```

< > =: Comparison criteria.

```
Select message Where [Transfer Cost] > 5
```

" " : Characters used to enclose values and names of settings or sets that contain blanks.

[ ] : Characters used to enclose the names of object types, links or characteristics that contain blanks.

```
Select [Report template (MS Word)] where Name = &Name
```

/* */ : Characters used to enclose comments.

```
/* Main Org-Units are not components of another Org-Unit */
```

CHECKING SYNTAX AND GETTING HELP

When writing a query, you can:

- click **Analyze** to check syntax of the query without executing it.
- obtain help on the *metamodel*.

To obtain help on the metamodel to find the name of a MetaClass, MetaAssociation or MetaAttribute:

1. Insert a question mark (?) next to the metamodel element for which you require help.
   A help dialog box appears and displays the MetaAssociationEnds, MetaAttributes or MetaClasses you can access at that point in the query, based on its previously defined query path.
2. Double-click an element in the help list box to replace the question mark with the name of the element.

RESULT: INTO

Use the "Into" operator to name a set of results. This name is used as a title for the dialog box displaying the objects found.

```
Select MetaClass Into @set
```

The Into operator is optional: if it is missing, the result has the same name as the target MetaClass (“select Message where ...” is the same as “select Message into @Message where ...”). The name of the set must be preceded by @.

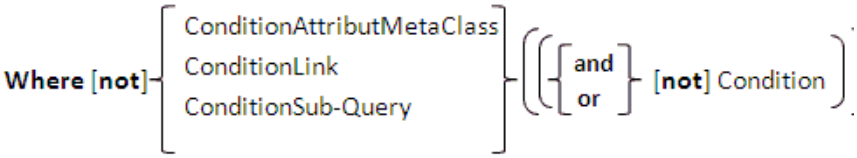
You can save the result using the commands in the **Query** menu of the dialog box displaying the query result. You can apply the "From" operator to this result.

CONDITION: WHERE

Conditions that define query criteria can concern the values of object characteristics or links, or the existence or non-existence of a link. Note that conditions can be grouped.

A condition indicates the query criterion or criteria. A condition is optional: if there is no condition, the result is the set or the combination of sets indicated after the From operator. If no set was specified, the result contains all objects of the type indicated.

A condition can consist of basic conditions connected by the or, and, and not operators. You can use brackets according to the rules indicated for combining sets.



A condition generally consists of:

- a path indicating a link or a characteristic,
- followed by a query operator,
- followed by query parameters.

This restricts the query to target objects fulfilling query conditions.

Sub-query: In

The "in" operator allows you to indicate a condition on members of a saved set or to indicate successive conditions.

```
Query Path [not] in @Set,MetaClass where Condition
```

Grouped conditions also allow you to indicate successive conditions, see "Grouped conditions", page 15.

Examples:

```
select Message where Diagram in @Mod
```

```
select Message where Source-Org-Unit in @Org-Unit where
name like “Department#”
```

Browsing the Metamodel

You can concatenate successive MetaAssociationEnds to define a path in the *metamodel*.

Reminder on MetaAssociationEnd names

In the metamodel, objects are linked as follows:

MetaAssociationEnd names follow this rule: seen from an object type, the link is indicated by the name of the opposite MetaAssociationEnd (in this diagram, seen from "Object Type 1", the link has the name of "MetaAssociationEnd 2").

When the link is:

- explicit (for example, Source-Message between Org-Unit and Message) both MetaAssociationEnds have names (Source-Org-Unit and Message-Sent).
- implicit, the MetaAssociationEnds have the same name as the MetaClass to which they are connected.
- generic, that is when it reaches several different MetaClasses, it is possible to specify a particular MetaClass, separating it with character ':'.

```
select Message where [Message Recipient]:[Operation] =
&Operation
```

The MetaAssociationEnds of reflexive links have their own specific names (for example, Component and Aggregation-Of for Compositions, Previous and Next for Sequences.

The object type attached to the last MetaAssociationEnd is called the "source object".

Example:

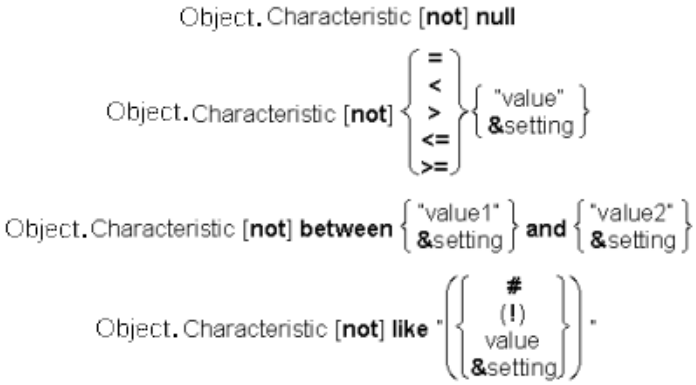
Using the following query, you can obtain the list of Messages Sent by Org-Units in Diagrams of a Project:

```
select Message where Source-Org-Unit.Diagram.Project =
&PROJ
```

The query below gives the same result (see "Conditions on Object Characteristics", page 12):

```
select Message where Source-Org-Unit in Org-Unit where
Diagram in Diagram where Project = &PROJ
```

Conditions on Object Characteristics



☛ *Indicates that the characteristic is not entered.*

Comparisons are done on an alphanumeric basis. For example, 2 is greater than 10 or 02.

Wildcards

In the "like" condition, the "# " symbol matches any number of characters. The "!" is the wildcard that matches one character, and can be repeated. For example:

- #x matches all the values beginning with x.
- #x matches all the values ending with x.
- #x# matches all the values containing x.
- !x# matches all the values that have x as the second character.
- x#x matches all the values beginning and ending with x.
- "!!" matches all the values that have two characters.

Examples:

```
select Org-Unit where Name = “Account Manager”
```

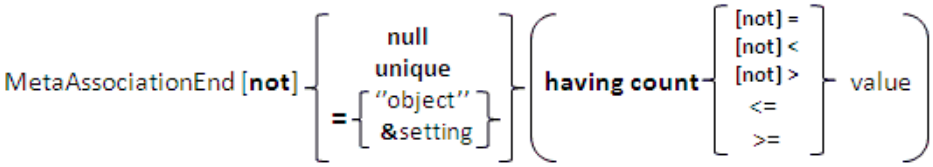
☛ *Here we give an example of a query on a precise name ("Account Manager"). In practice, querying the name of a calculated object can be tedious. It is recommended that a setting be called to write the query. For example: "select Org-Unit where Name = &ManagerName" You then specify the object name when you run the query.*

```
select Org-Unit where Name like "#count#"

select Org-Unit where Name like "!!!!!!! Manager#"

select Org-Unit where Name like "Ac#"

select Diagram where Nature null
```

Conditions on Links

Conditions on links can check link existence and link characteristic values.

Link existence condition: Null, Unique and Having count



- ***Null*** means that the link does not exist. If several MetaAssociationEnds are concatenated, it is the link reached by the last MetaAssociationEnd that is tested.
  For example, "select Project where Diagram.Org-Unit.Message sent null" selects projects where diagrams contain org-units that do not send messages; diagrams with no org-units are ignored.
- ***Unique*** means that the link must exist only once. For concatenated MetaAssociationEnds , the link reached by the last leg is the one counted. (Note: not unique means that at least two links exist.)
- ***Having count*** indicates the number of times the link must exist. For concatenated MetaAssociationEnds , the link reached by the first MetaAssociationEnd is the one counted.

Examples:

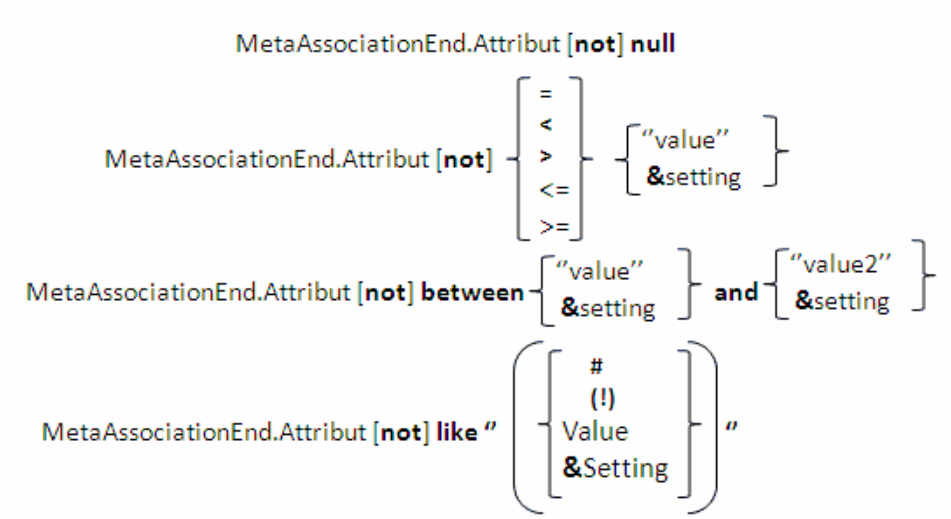
```
select Org-Unit where Diagram null

select Org-Unit into @MainOrg-Units where Component not
null

select Diagram where Org-Unit.Message-Sent not null having
count > 2
```

In this last example, diagrams containing at least two org-units that send messages are found.

Link characteristic condition

A condition on a link characteristic is expressed in the same way as a condition on an object characteristic. The name of the link characteristic is concatenated with the

name of the link. This name and the characteristic name are separated by a period (.).



Example: Query org-units that always send at least one message:

```
select Org-Unit where Message-Sent.Predicate = "always"
```

Source object characteristic condition

You can indicate a condition for a characteristic of the source object in the same way as for link characteristics:

```
select Org-Unit where Message-Sent.Predicate = "always" or
Message-Sent.Flow-Type = "Information Flow"
```

If a characteristic is not specified, the query is based on the name ("Diagram.Name ="zzz"" is equivalent to "Diagram ="zzz""):

```
select Diagram where Org-Unit = &Org
```

Browsing reflexive links Deeply

When the query target MetaClass is identical to the source MetaClass, and the link browsed is a reflexive link (eg. Composition), you can query all levels of the composition using the deeply operator.

```
MetaAssociationEnd deeply Condition
```

Examples:

```
select Org-Unit into @components where Aggregation-Of
deeply = &Org-Unit
```

```
select [Message] where [Org-Unit-Recipient].[Message-Sent]
deeply in [Message]
   where [Org-UnitSender = &"Org-Unit"
```

The first example finds all component org-units of a given org-unit, whether directly or via intermediate org-units.

The second example queries for all Messages produced by target objects when they receive the messages sent by a given org-unit (source).

Restriction concerning Deeply

You cannot combine "Deeply" and "Inherited".

If a link is browsed with the "Deeply" keyword, the Inherited clause enabling inclusion of inherited objects is ignored for this link.

Grouped conditions

When a condition is applied to several characteristics of the source object, it is expressed as a grouped condition. Each of the "and", "or" and "not" operators involves any source object that satisfies the indicated condition. If you want the source object to be the same for different criteria, you must group these criteria with brackets.

As an example, the query:

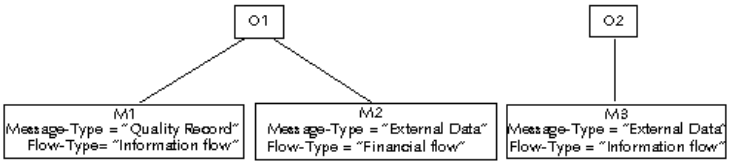
```
select Org-Unit where Message-Sent.Message-Type = "External
Data" and Message-Sent.Flow-Type = "Information flow"
```

finds the org-units that send at least one message of the "External Data" type and at least one message of the "Information flow" type. Nothing in this query indicates that the org-units found must be connected to at least one message of the "Information flow" type which is also of the "External Data" type. To do this, you must rewrite the query as follows:

```
select Org-Unit where Message-Sent.(Message-Type =
"External Data" and Flow-Type = "Information flow")
```

You must place the period before the brackets. You can also group the conditions in a sub-query:

```
select Org-Unit where Message-Sent in Message where
(Message-Type = "External Data" and Flow-Type =
"Information flow")
```



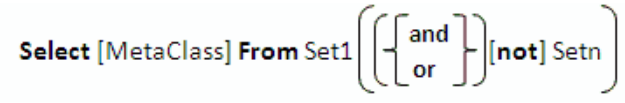
The first query gives O1 and O2, while the second only finds O2.

To combine conditions on object characteristics and link characteristics, you must also use grouped conditions.

For example, to query messages always sent by a given org-unit, that is a query on the name of an org-unit and the value of the predicate characteristic between message and org-unit, you have to write the query as follows:

```
select Message where Source-Org-Unit.(predicate = "always"
and name = &Org-Unit)
```

SETS: FROM



You must use the @ symbol as a prefix for set names used in the query.

☺ *The MetaClass preceding the From operator is optional: a set having been defined for a MetaClass, this MetaClass is used by default.*

You can save the result returned by a query as a set. You can use this result in other queries for the duration of your query session. Sets will restrict your query, which can be useful in optimizing response times, executing lengthy queries only once.

Only saved sets can be used by other queries (except sets used in queries containing several Select clauses; see their description below).

Example:

The following query searches for messages of a diagram that satisfy a condition ("Where" operator) in the set of messages "Mod_Messages".

```
select Message from @Mod_Messages where Condition
```

Sets are kept and named using the **Keep** command of which syntax is keep @set.

You can delete sets with the **Delete** command. The sets are also deleted when you close your work session.

Example:

```
select Org-Unit into @MainOrg-Units where not Aggregation-
Of

keep @MainOrg-Units
```

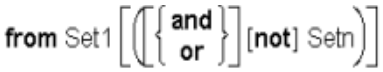
In this query, you build and keep the set of main org-units, that is org-units not aggregated in any other org-unit.

In the next query, you will reuse this set and delete it when you no longer need it.

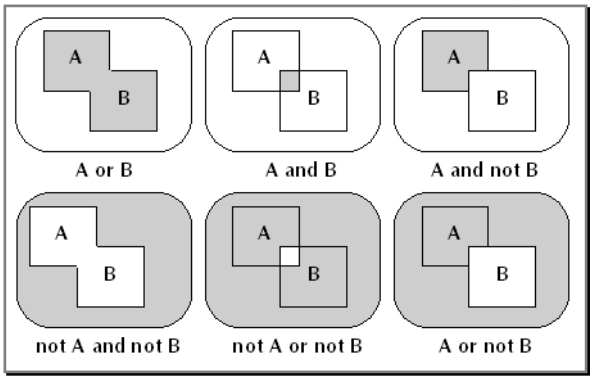
```
select Org-Unit from @MainOrg-Units where Diagram.Project =
&Project

delete @MainOrg-Units
```

This provides you with the list of the main org-units for the project specified by the project setting.

Set Operations



The set used after the "From" operator can be built from a combination of saved sets.



You can use brackets. The rules of distribution and precedence are as follows:

- (A or B) or C = A or (B or C) = A or B or C
- (A and B) and C = A and (B and C) = A and B and C
- (A or (B and C) = A or B and C = (A or B) and (A or C)
- (A and (B or C) = A and B or A and C = (A and B) or (A and C)
- not (A or B) = (not A) and (not B) = not A and not B
- not (A and B) = (not A) or (not B) = not A or not B

Example:

```
select Org-Unit from @MainOrg-Units and not @QualityOrg-
Units
```

QUERY TIPS AND EXAMPLES

Tips on Using Queries

Different queries and the same result

You can obtain the same result using different queries.

Query diagrams of a project that describe an org-unit

- And between two conditions

```
Select Diagram where Described-Org-Unit and Project =
&Project
```

- Query in an intermediate set and restrict to this set:

```
Select Diagram into @Org-UnitDiagram where Described-Org-
Unit
```

```
Select Diagram from @Org-UnitDiagram where Project =
&Project
```

- Query in two sets and find their intersection:

```
Select Diagram into @Org-UnitDiagram where Described-Org-
Unit
```

```
Select Diagram into @ProjectDiagram where Project =
&Project
```

```
Select Diagram from @Org-UnitDiagram and @ProjectDiagram
```

Query messages of a project

- Extended form using a sub-queries

```
Select Message where Diagram in Diagram where Project =
&Project
```

- Browsing the links

```
Select Message Where Diagram.Project = &Project
```

Query org-units without a Message-Sent

```
Select Org-Unit where [Message-Sent] null
```

```
Select Org-Unit where not [Message-Sent]
```

Tips with "and", "not" and "or" operators

It is important to ensure that the query corresponds to the desired search, particularly when using the "and", "not" and "or" operators.

The following query:

```
Select Org-Unit where not [Message-Received] = "Order"
```

gives all the org-units except those that receive the "Order" message. The org-units that do not receive a message are also included in the result.

However, the following query:

```
Select Org-Unit where [Message-Received] not = "Order"
```

selects only the org-units that receive a message except those that receive the "Order" message.

To obtain the same result as above, write:

```
Select Org-Unit where [Message-Received] not = "Order" or
[Message-received] null
```

Other query examples

Query messages always sent by an org-unit

```
Select Message into @obl where Source-Org-Unit = &Org-Unit
and Source-Org-Unit.Predicate = ”always”
```

finds the messages sent by the org-unit entered as the setting value, and connected to any org-unit with the source predicate = ”always”.

```
Select Message into @obl where Source-Org-Unit.(name =
&Org-Unit and Predicate = ”always”)
```

finds the messages sent by the org-unit entered as the setting value, with the source predicate = ”always”.

Check validity of link used

Similarly, when counting links, you should check the link:

```
Select Diagram where Org-Unit.[Message-Sent] not null
having count >= 3
```

finds diagrams having at least three messages sent by the org-units in these diagrams.

Use the following query to find diagrams having org-units sending at least three messages:

```
Select Diagram where Org-Unit in Org-Unit where
Message-Sent not null having count >= 3
```

Examples of queries on reflexive links

- Use the following query to find org-units that are part of the "Sales Management" department (and are therefore aggregations of this department):

```
Select Org-Unit where Aggregation-Of = "Sales Management"
```

- Query to find org-units that are directly or indirectly part of the "Sales Management" department (which are aggregations of the "Sales Management" department or one of its components):

```
Select Org-Unit where Aggregation-Of deeply = "Sales
Management"
```

- Query to find org-units that are part of an org-unit other than the "Sales Management" department:

```
Select Org-Unit where Aggregation-Of deeply not = "Sales
Management"
```

- Query to find org-units that are not part of the "Sales Management" department:

```
Select Org-Unit where not Aggregation-Of deeply = "Sales
Management"
```

- Query to find org-units that are not components of something or are part of the "Sales Management" department:

```
Select Org-Unit where not Aggregation-Of deeply not =
"Sales Management"
```

Examples with diagram, org-unit, message and keyword

- Use the following query to find diagrams connected to at least one org-unit that receives at least one message with a keyword:

  ```
  Select Diagram Where Org-Unit. [Message received]. Keyword
  ```

- Use the following query to find diagrams connected to an org-unit that receives no messages without a keyword:

  ```
  Select Diagram where Org-Unit.(not Message-Received.(not
  Keyword))
  ```

  OR

  ```
  or Select Diagram where Org-Unit.(not Message-
  Received.Keyword null)
  ```

- Use the following query to find diagrams connected to an org-unit that receives messages that all have a keyword:

  ```
  Select Diagram where Org-Unit.(Message-Received and not
  Message-Received.(not Keyword))
  ```

  OR

  ```
  Select Diagram where Org-Unit.(Message-Received and not
  Message-Received. Keyword null)
  ```

- Use the following query to find diagrams that have only org-units that receive messages without a keyword:

  ```
  Select Diagram where not Org-Unit.(not Message-
  Received.(not Keyword))
  ```

- Use the following query to find diagrams whose org-units all receive messages that all have a keyword:

  ```
  Select Diagram where Org-Unit and not Org-Unit.(not
  Message-Received or Message-Received.(not Keyword))
  ```

Examples of Queries for MEGA Process

The following query examples are specific to **MEGA Process**.

- Use the following selector to find all the org-units of an organizational process and its component organizational processes:

  ```
  Select Org-Unit Where Diagram.[Described Organizational
  Process] = &"Organizational Process" Or Diagram.[Described
  ```

```
Organizational Process]].Aggregation-Of = &"Organizational
Process"
```

- Use the following selector to find all the org-units described in diagrams of an organizational process:

```
Select Org-Unit Where Diagram [Described Organizational
Process] =&"Organizational Process"
```

- Use the following selector to find all the messages whose type is "external data" in an organizational process diagram:

```
Select Message where Diagram = &Diagram and Message-Type =
"External data"
```

- Use the following selector to find diagrams describing an organizational process:

```
Select Diagram Where [Described Organizational Process] =
&"Organizational Process"
```

- Use the following selector to find the project organizational charts of the quality department:

```
Select Diagram where Project.(name = &Project and Employ =
"Quality") and Nature = "Organizational Chart"
```

- Use the following selector to find the organizational processes of a project connected to a chapter of the standard:

```
Select Project into @Projects where Aggregation-Of deeply
or name = &Project
```

```
Select [Organizational Process] where ( project in
@Projects) and chapter = &Chapter
```

- Use the following selector to find the event messages of an operation or messages connected to a synchronization of the operation.

```
Select Message into @E1 where Event = &Operation
```

```
Select Message into @E2 where Synchronization.Operation =
&Operation
```

```
Select Message from @E1 or @E2
```

- Use the following selector to find the messages which result from an operation either directly or via a condition.

```
Select Message into @R1 where result = &Operation
```

```
Select Message into @R2 where Source-Condition.Previous-
Operation = &Operation
```

```
Select Message from @R1 or @R2
```

- Use the following selector to find the hierarchical or functional organizational chart of a project.

```
Select Diagram Where project.(name = &Project And Employ =
"B") And Nature = "Organizational Chart"
```

INTRODUCTION TO MEGA STUDIO

Each organization is unique, with its own culture and methods. Ready-to-run software often integrates in the organization to only a limited extent. An activity as strictly regulated as enterprise architecture modeling gains in flexibility when the software tool can adapt to methodological recommendations, to demands of the domain and to graphical preferences. **MEGA Studio** allows you to implement and manage customizations and **MEGA** metamodel extensions as well as to adapt the tool to your own particular use.

MEGA Studio aims at simplifying product customizations using an ergonomic graphical interface. A **MEGA Studio** navigator allows you to precisely configure the multiple aspects of the **MEGA** environment: navigation, object properties pages, diagram types, modeling rules and the graphical interface.

Customizations or extensions that can be made to the metamodel can impact various aspects of **MEGA**:

- the creation of new concepts or new attributes. The metamodel diagram allows you to easily create MetaClasses and to create MetaAssociations between the different MetaClasses.
- the exploitation and use of the newly-created concepts. This means being able to show them in diagrams, use them in navigation trees, and being able to customize their properties pages.

PRESENTATION OF MEGA STUDIO

This guide covers the following points:

- ✓ "Managing the Metamodel", page 13 presents the metamodel diagram, creation of new MetaAttributes, MetaClasses and MetaAssociations. It also presents other tools contributing to metamodel management.
- ✓ "Configuring Diagrams", page 89 presents some basic principles used to configure display in the different diagram types, as well as the object types available in diagrams.
- ✓ "Creating and Editing Shapes", page 101 presents how to use the **Shapes Editor** tool to create or modify shapes.
- ✓ "Configuring Navigation Trees", page 115 presents the navigation window operating principle and navigation tree configuration techniques.

For other aspects of **MEGA Studio** as customization of properties pages, refer to the corresponding technical articles.

1

MANAGING THE METAMODEL

The following points are covered here:

Introduction to Metamodel Management

Founded in 1989, the Object Management Group (OMG) establishes and maintains computer industry specifications and promotes the theory and practice of object technology for interoperable enterprise specifications.

Within the OMG, **MEGA** works in several domains. In particular, **MEGA** performs a reviewer role in the specification of Meta Object Facility (MOF).

The objective of this standard is to ensure interoperability of different modeling tools by common definition of the concepts used.

OMG metamodeling architecture comprises four layers.

An example of use of this four layer architecture:

M3 MetaMetaModel	Basic objects	metaclass	MetaAssociation	metaclass
M2 Metamodel	Metamodel objects	Org-Unit	An Org-Unit sends a Message	Message
M1 Model	MEGA user objects	Client	The Order is sent by the Customer	Order
M0 Objects	End user objects	Mr Smith	Mr Smith issues Order No. COM1727	COM1727

The metamodel enables definition of semantics of models that will be created.

They are located in level 2 of metamodel architecture defined by the OMG.

The **MEGA** metamodel can be modified or extended to suit specific requirements. To do this, certain precautions must be taken to assure maintenance of its extensions over time.

METAMODEL EXTENSIONS AND MODIFICATIONS

☛ *The metamodel defines the language for expressing a model. It defines the structure used to store the data managed in a repository The metamodel contains all the MetaClasses used to model a system, as well as their MetaAttributes and the MetaAssociations available between these MetaClasses. The metamodel is saved in the environment system repository.*

You can create metamodel extensions to manage new object types. Repositories that exchange data (export, import, etc.) must have the same metamodel, otherwise certain data will be rejected or inaccessible.

The metamodel diagram enables modification of repository structure (create object types, links, characteristics, modify object and link typing).

This modification type should be used with extreme care.

☛ *So that a user can modify **MEGA** standard configuration, **Authorize MEGA Data Modification** option (**Repository** options) must be selected. For more details, see **MEGA Administration-Supervisor** guide, chapter "Managing Options".*

You can create metamodel extensions using *command files*, in text format (ASCII).

To take command files into account in the system repository:

1. Open **MEGA Administration**.
2. In the navigation tree, right-click the desired environment and select **Open**.
3. Expand the **Repositories** folder.
4. Right-click **SystemDb** and select **Object Management > Import**.

Whether you have created your extensions using a metamodel diagram or a command file, you must validate them by a translation of the environment metamodel, using the command **Metamodel > Translate and Compile** in the environment pop-up menu.

Warning Concerning Metamodel Modification

You can modify the structure (or *metamodel*) of environment repositories, for example when you want to add a MetaAssociation between two MetaClasses.

Note that these modifications should be carried out with caution because they will have to be maintained by the administrator. The metamodel is saved in the system repository of the environment: all repositories of an environment have the same metamodel.

Do not create extensions to the metamodel without careful consideration. If you create metamodel extensions, encode your extensions (MetaClass, MetaAssociation, MetaAttribute or text) with a specific prefix that identifies your site, so as to avoid

possible conflict with a new MEGA concept when updating your version of MEGA.

☛ *Data exchanges between repositories with different metamodels can result in rejects.*

Extensions to the metamodel are managed differently from modifications to the standard metamodel.

Creating metamodel extensions

After you create new MetaClasses, MetaAssociations or MetaAttributes, back up these metamodel extensions with their absolute identifiers (export as MGR file). You will need them when creating a new environment.

Upgrading a site or environment does not impact extensions you have added to the metamodel.

Managing modifications to the standard metamodel

The MetaAttributes, MetaClasses and MetaAssociations delivered as standard are protected. It is necessary to **Authorize MEGA data modification** in the user options to be able to execute these updates. Do not forget to prohibit this modification later. For more details, see "Managing Options", page 419 **MEGA Administration-Supervisor** guide, chapter "Managing Sites".

When upgrading a site or an environment to a new version, the standard installed concepts will be automatically reassigned the standard values.

You must save the MGR type files that you extracted after having made modifications, in order to reapply these after upgrading your environment.

You should apply your modifications to the environment (and only those changes, not an extraction of the complete metamodel) in order to have both the upgraded standard version and your own modifications.

Metamodel extensions backup

A simple way of backing up your metamodel extensions is to work in a transaction using the metamodel diagram.

You can then make an initial verification in this transaction before dispatching. In general, you must translate and compile the metamodel so that your extensions will be correctly taken into account.

When you are satisfied with operation of your extensions, you can export the logfile of your transaction before you dispatch it. In this way you will obtain a command file that you can then reimport into another environment when using your extensions or at a change of version.

Transferring metamodel extensions

The absolute identifiers of metamodel concepts are indispensable if **MEGA** is to run properly. Metamodel extensions (creation of MetaClass, MetaAssociation, MetaAttribute) should be processed as follows:

- Carry out transaction logfile export if you have been working in a transaction, or a metamodel extensions logical backup in other cases.
- Save the generated file, which you will use to transfer the extensions into other environments.

Compiling the environment

After you create or import extensions into an environment, it must be compiled. This is done in **MEGA Administration**, by requesting translation of the metamodel in the current language.

Restrictions

- Repositories that exchange data through export or import processes must have the same metamodel.
- Extensions to the metamodel (new MetaAssociations and MetaClasses) cannot be graphically handled in standard diagrams. Diagram configuration needs to be customized to take into account such extensions (contact MEGA Professional Services for diagram customization).

Implementation conditions

So that modifications to the metamodel are visible to users:

- the metamodel must be compiled
- a new transaction must be opened. If the user has a transaction open, it must be dispatched or updated so that the user has access to metamodel modifications.

Precautions Concerning Metamodel Extensions

Certain precautions must be taken when customizing the metamodel:

- Always work in a test environment. Remember that modification or deletion of a metamodel concept (MetaClass/MetaAttribute) will affect all data linked to this concept.
- Create a "Super User" with administration rights and rights to modify **MEGA** data.

Concepts

Metamodel definition commands concern:

- MetaClasses (or object types)
- links possible between these MetaClasses (called MetaAssociations)
- the two MetaAssociationEnds of each of these MetaAssociations
- MetaAttributes (characteristics).
  Object comments or texts are assimilated in MetaAttributes.

Concept	Syntax
MetaClass (or object type)	metaclass
MetaAssociation (or link)	MetaAssociation
MetaAssociationEnd	MetaAssociationEnd
MetaAssociationType (or link type)	MetaAssociationType
MetaAttribute (or characteristic)	MetaAttribute
MetaAttributeGroup (or group of MetaAttributes)	MetaAttributeGroup
MetaAttributeValue (or value of MetaAttribute)	MetaAttributeValue

Concept names

The name of a concept is unique, ie. the name for a MetaClass, MetaAssociation, MetaAttribute or text must be used only once.

Names of MetaClasses with standard naming rule contain maximum 255 characters.

Concerning MetaClasses with namespace, the local name is limited to 140 characters, while the complete name is limited to 255 characters.

☛ *For more information on namespaces, see "Managing Namespaces", page 67.*

Name of a concept must begin with a letter (A to Z). The following characters are authorized for use in concept names:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m n o p q r s t u v w x y z
à á â ã ä å ç é è ë ê ì í î ï ñ ò ó ô ö ù ú û ü ÿ ý
0 1 2 3 4 5 6 7 8 9
* ( ) / = + % ? $ _ & €
, ; - :
```

☛ *Hiragana, Katakana and Kanji characters are accepted for object names in the Japanese language.*

You cannot use "& , ; - : " as the first character. There can be spaces in the name, except as first and last character.

☛ *Unlike the other characters, the ampersand (&) is not controlled as the first character:*

- *it can be used in an object name.*
- *it should not be used in the name of a MetaClass, MetaAssociation, MetaAssociationEnd or MetaAttribute.*

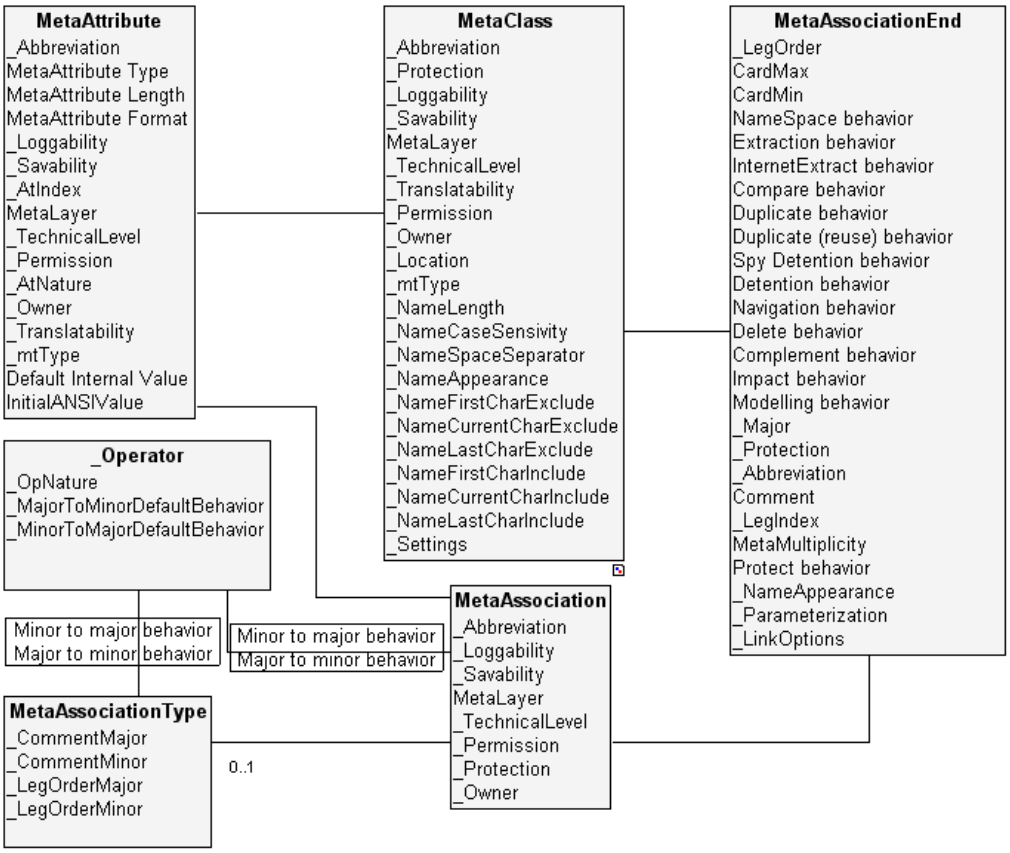
Technical concepts other than repository core concepts, (MetaClass, MetaAssociation and MetaAttribute) have names beginning with an underscore (_).

💣 **It is impossible to define extensions with names beginning with characters "_" or "$".**

💣 **It is possible to modify the list of characters (see "Restricting the names of object types", page 28). Note that this may create problems when executing queries or descriptors.**

Note that you cannot use a name already in use. You must attach a prefix to your extension names. This avoids conflicts that might occur later when the standard metamodel is upgraded. For example, if you create a metamodel extension called "Table" for **MEGA Process** and later install **MEGA Database Builder**, a conflict will occur.

Metamodel diagram



All of these MetaClasses have the following attributes:

- Name
- Absolute Identifier
- Authorization
- Creation Date
- Modification Date
- Creator, Modifier
- Creator Name
- Modifier Name
- Authorization Level
- Creation Version
- Modification Version
- Comment and History

Vocabulary used

To conform to MOF vocabulary, **MEGA** has modified the terminology of its concepts.

Correspondence between versions 5.2 and 5.3

Old Name	New Name
Object Type (or Segment)	metaclass
Link	MetaAssociation
Leg	MetaAssociationEnd
LinkType	MetaAssociationType
Attribute	MetaAttribute
InterfaceDef	MetaAttributeGroup
AttributeValue	MetaAttributeValue

Correspondence between versions 6.1 and 6.1 SP1

Modifying names of attributes

Old Name	New Name
_AbstractionLevel	MetaLayer
_AtFormat	MetaAttribute Type
_AtLength	MetaAttribute Length
_AtExternalFormat	MetaAttribute Format
_IdAbsTextFormat	MetaText Format
_IdAbsSubstitutedAttribute	Substituted MetaAttribute
_ScanMajorType	Minor to major behavior
_ScanMinorType	Major to minor behavior
DefaultAnsiValue	Default Internal Value

Changing values of the MetaLayer attribute (formerly _AbstractionLevel)

Old Name	New Name
MetaData	Meta-MetaModel (MOF)
Data	Metamodel

Modifying values of the MetaAttribute Type attribute (formerly _AtFormat)

Old Name	New Name
Alphanumeric	String
Bool	Boolean
Short	Short
Long	Long
Date	DateTime
Text	VarChar
Binary	VarBinary
QCQ	Binary
HexaLong	Double
DoubleFloat	Float

CREATING METAMODEL EXTENSIONS: METHOD

How to Create Metamodel Extensions

When you create a metamodel extension, it is recommended that you first create a development environment, create the metamodel extension in this environment, then import the extension in a test environment.

Once extensions have been validated in the test environment, you can import them into the production environment.

To create a metamodel extension:

1. Create a development environment and a test environment and initialize the system repository logfiles.
   ☛ *Select a user with rights to modify **MEGA** data.*
2. Create the metamodel extensions in the development environment.
   ☛ *Remember to prefix or suffix the extensions created.*
3. Export the system repository logfile.
4. Import this logfile in the test environment - with a user with rights to modify **MEGA** data - to verify the import result.
5. If there are no rejects, import the logfile again, saving updates.
6. Translate and compile the metamodel in the test environment and verify that extensions function correctly.
7. Import the logfile containing the extensions in the production environment.

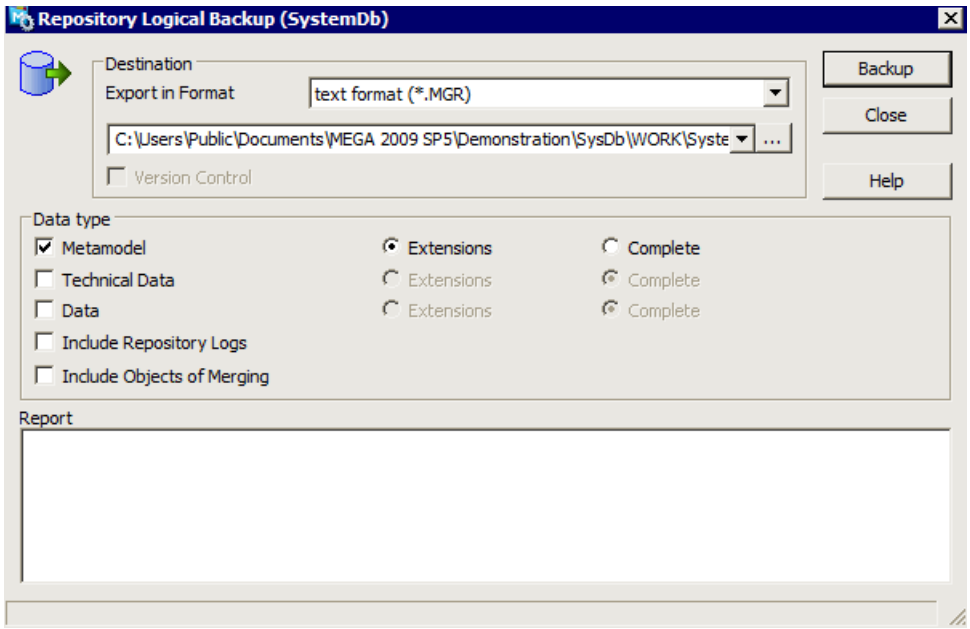
When the extensions have been validated, you can import the logfile with your extensions in the production environment, following the procedure described above.

Extensions Backup

To back up the metamodel extensions you have created:

1. Run **MEGA Administration** and connect to the desired environment.
2. Expand the **Repositories** folder and right-click **SystemDb** and select **Logical Backup**.

3. In the **Repository Logical Backup (SystemDb)**, check only the boxes **Metamodel** and **Extensions**.



4. Click **Backup**.
   The generated file contains all extensions created in the environment.

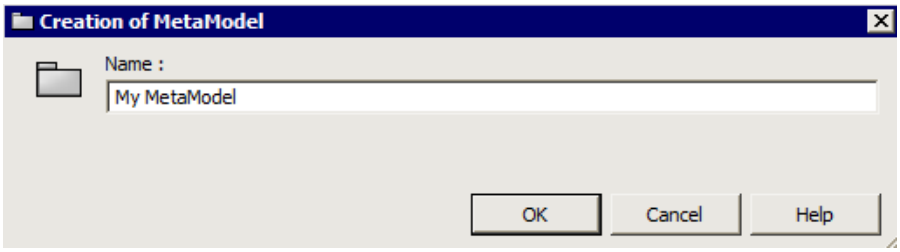
THE METAMODEL DIAGRAM

Creating a Metamodel Diagram

☛ *The metamodel diagram is available with the **MEGA Studio** technical module.*

To create a metamodel diagram:

1. Check that you are in advanced metamodel (**Tools > Options**, **Repository** icon, **Metamodel Access**, "Advanced".
2. In the **MetaStudio** navigation window, right-click the **Metamodel** folder and select **New > Metamodel**.
3. Enter the metamodel **Name** and click **OK**.



4. Right-click the metamodel you have just created and select **New > Diagram**.
   The window that opens proposes selection of diagram type.
5. Select **Metamodel Diagram** and click **Create**.
   The Metamodel diagram opens. It describes a metamodel instance.
   In it you can place MetaAssociations and MetaClasses.

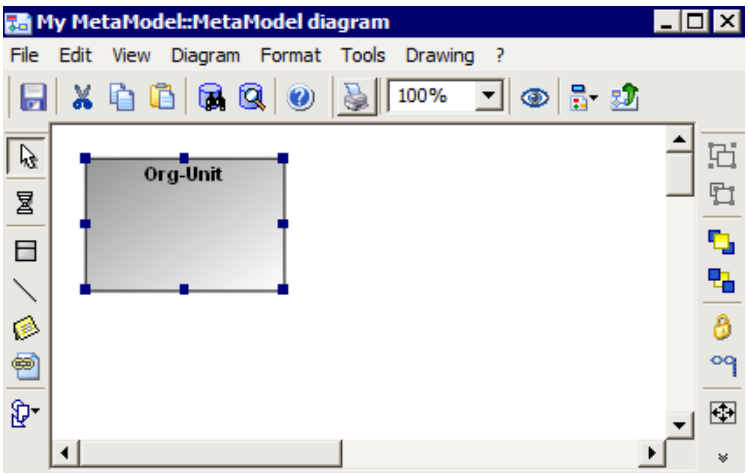
   ☛ *A MetaClass or a MetaAssociation placed in the diagram is automatically connected to the described metamodel, see "Placing a MetaClass in a Metamodel Diagram", page 25 and "Placing a MetaAssociation", page 26.*

Placing a MetaClass in a Metamodel Diagram

To place an existing MetaClass in the metamodel diagram:

1. In the diagram insert toolbar, click **MetaClass** then click in the diagram.
   The **Add MetaClass** dialog box appears.
2. In the **Name** box, click the drop-down menu arrow and select **List**.
3. Select the required MetaClass in the list.
4. Click **OK**.
   The MetaClass name is displayed in the **Add MetaClass** dialog box.

5. Click **Connect**.
The MetaClass is placed in the metamodel diagram.



You can access the attributes of the MetaClass by opening its Properties window.

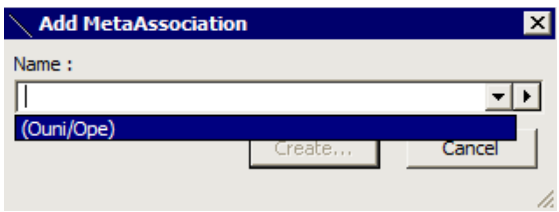
☛ *In the same way you can place several MetaClasses in the metamodel diagram.*

Placing a MetaAssociation

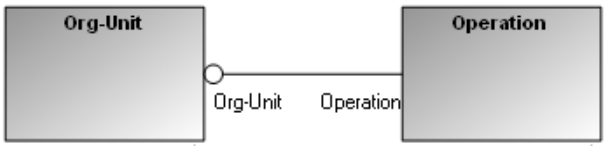
To place a MetaAssociation between two MetaClasses in the diagram:

1. In the metamodel diagram insert toolbar, click **MetaAssociation** , then draw a link from the first MetaClass to the second.
2. In the **Add MetaAssociation** dialog box, click the arrow to select the MetaAssociation that interests you.

☛ *The vertical arrow only appears if the MetaAssociation has already been created. To create a MetaAssociation, click the horizontal arrow and select **New**.*



3. Click **Connect**.
The selected MetaAssociation appears in the diagram.



You can access the attributes of the MetaClass by opening its Properties window.

> ☛ *The major MetaAssociationEnd is indicated by a black diamond (composition), white diamond (aggregation) or a slash (default). See "Creating a MetaAssociation", page 31 et "Specifying MetaAssociation Behavior", page 33 for more details.*

METACLASSES

The following points are covered here:

Creating a MetaClass (Object Type)

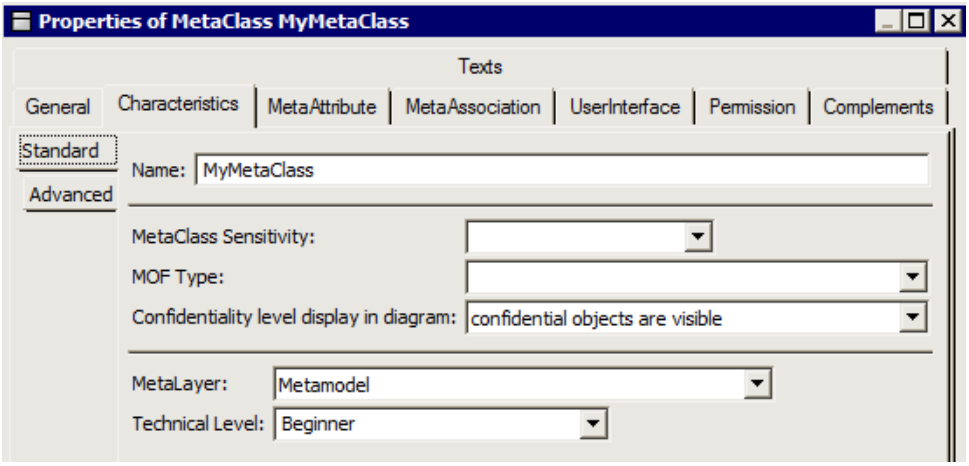
To create a MetaClass in the metamodel diagram:

1. Click **MetaClass**, then click in the diagram.
2. In the dialog box that appears, enter the name of the MetaClass and click **Create**.
   The new MetaClass is created.
   The "Name" and "Comment" attributes are automatically assigned to a newly-created MetaClass.

   ☛ *The "Absolute identifier", "Authorization", "Creator Name", "Creation Date", "Modifier Name", "Modification Date" and "History" characteristics are also created automatically.*

To access MetaClass properties:

- Right-click the MetaClass and select **Properties**.
  The MetaClass Properties window opens.



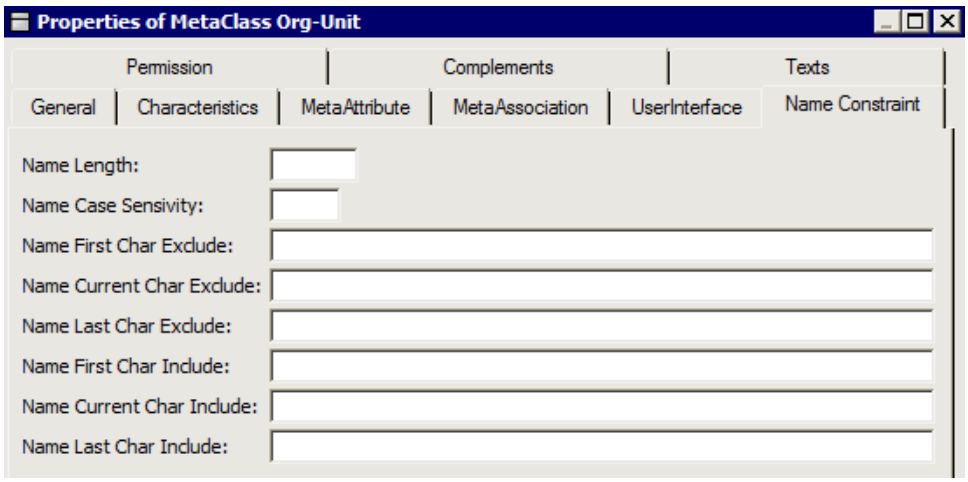
Restricting the names of object types

Characters that can be used in object names are subject to the same restrictions as concept names.

By default, names are in upper/lower case, and are limited to 63 characters (140 for objects included in a namespace).

The names are stored the way they are entered. However, the system verifies that once the name is converted into uppercase and stripped of any accents, it isn't already in use. For instance, for objects of the same type, objects "SMith" and "Smith" are considered to be identical.

To configure characteristics indicated below, you must be in **Extended** metamodel access mode. This configuration is carried out in the MetaClass Properties window (**Name Constraint** tab).



For each metaclass, you can specify:

- Maximum name length: **Name Length**

  ☛ *_NameLength must be equal to or greater than 16.*

- Systematic name conversion: **Name Case Sensitivity**
  - to uppercase: U (Upper)
  - to lowercase: L (Lower)
  - no conversion: I (Ignore)
- Illegal characters:
  - for first character: **Name First Char Exclude**
  - for characters 2 to n-1: **Name Current Char Exclude**
  - for character n: **Name Last Char Exclude**

For each MetaClass, you can add to the list of authorized characters:

- for first character: **Name First Char Include**
- for characters 2 to n-1: **Name Current Char Include**
- for character n: **Name Last Char Include**

☛ *It is recommended that you do not add to authorized characters for MetaClasses other than technical. Correct operation of query and execution of descriptors is not guaranteed for objects with added characters.*

Typing a MetaClass

The MetaAssociation between MetaClass and _Operator is used to modify operation of standard tools for export, protection, etc. for the specified MetaClasses.

This MetaAssociation carries two attributes:

- _ScanInit (default value: no value)
  If _ScanInit has value "S", all objects of this MetaClass must be systematically taken into account by the tool. This is the case for Sort, Keyword, and Language objects when exporting **MEGA** objects.
- _ScanInit (default value: no value)
  If _ScanType has value "D", when one object of this type is extracted, the minor objects linked to it are also extracted, as well as their descendants. This is the case for Information, Rule, Relationship, MetaAssociationEnd and Entity

☛ *For more details, see **MEGA Administration-Supervisor** guide.*

METAASSOCIATIONS

The following points are covered here:

Creating a MetaAssociation

Creating a MetaAssociation (link) implements:

- a major MetaClass via a major MetaAssociationEnd
- a minor MetaClass via a minor MetaAssociationEnd

To create a MetaAssociation, see "Placing a MetaAssociation", page 26.

Semantic rule

The *major* MetaClass is that which is modified at deletion of its MetaAssociation with the minor MetaClass.

The *minor* MetaClass is that which retains its semantic value at deletion of its MetaAssociation with the major MetaClass.

> ☛ *For more details on major and minor objects, see the **MEGA Administration - Supervisor** guide.*

MetaAssociation orientation

When creating a MetaAssociation in the metamodel diagram, it is the direction in which the MetaAssociation was drawn that determines the major MetaAssociationEnd and the minor MetaAssociationEnd.

The major MetaAssociationEnd is on the origin side of the MetaAssociation, the minor MetaAssociationEnd is on the arrival MetaClass.

When a MetaAssociation is created, the "Order" attribute is assigned to it automatically.

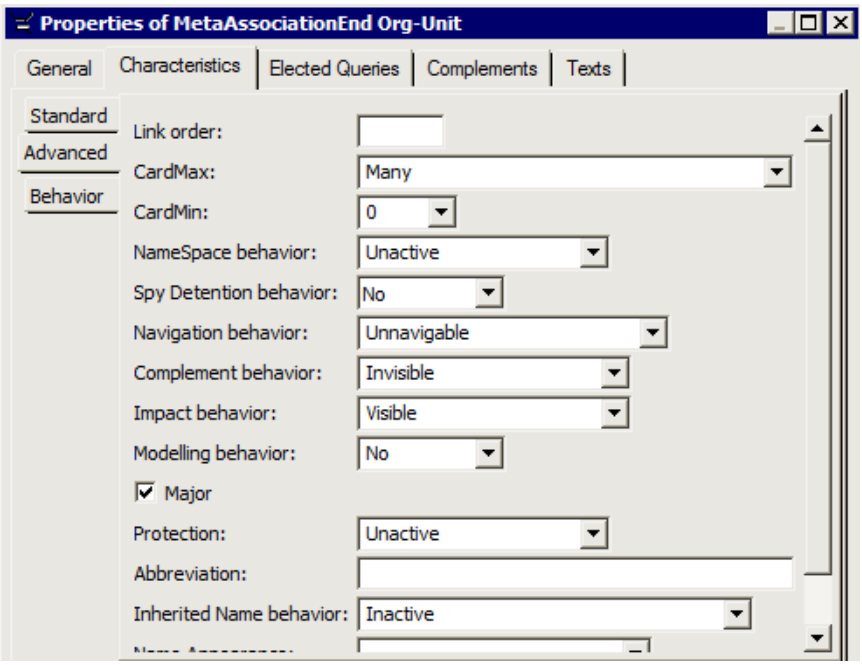
> ☛ *The major MetaAssociationEnd is indicated by a black diamond (composition), white diamond (aggregation) or a slash (default). See "Specifying MetaAssociation Behavior", page 33 for further information.*

Reversing Major/Minor Orientation

After creation of a MetaAssociation, major/minor orientation can be reversed in the metamodel diagram by opening the Properties window of each MetaAssociationEnd.

To modify MetaAssociation orientation:

1. In the Properties window of the major MetaAssociationEnd, select the **Characteristics** tab, then the **Advanced** subtab.
2. Clear the **Major** check box.



3. Open the properties of the minor MetaAssociation End and select the **Major** tab.

Modifying Object Protection

For MetaClasses and MetaAssociationEnds, the ***Protection*** attribute allows you to activate ("A") or deactivate ("U") checks related to user authorization level. By default, for all major MetaClasses and MetaAssociationEnds, this attribute is active.

If no value is specified for ***Protection***, a Major/Minor orientation value is deduced:

- If the MetaAssociationEnd is major, ***Protection*** is "Active" (Value "A")
- If the MetaAssociationEnd is minor, ***Protection*** is "Inactive" (Value "U")

> ☛ *This function is available with the **MEGA Administration - Supervisor** technical module only.*

For MetaClasses and MetaAssociationEnds for which you wish to deactivate this check, assign the value "U" to the ***Protection*** attribute, and there will be no check at updating.

To modify MetaClass and MetaAssociationEnd protection:

1. In the Properties window of a MetaClass or MetaAssociationEnd, select the **Characteristics** tab, then the **Advanced** subtab.
2. For the **Protection** attribute, select value **Inactive**.

Imposing MetaAssociation Uniqueness

This prevents connecting several objects to the same object.

```
Example: To ensure that an org-unit is connected to a single
keyword, you must specify on the MetaAssociationEnd between
Org-Unit and Keyword that the MetaMultiplicity attribute is
0..1 or 1.
```

Uniqueness is applied particularly on sub-type and certain composition links.

*For compatibility with versions earlier than Service Pack 1 of **MEGA** version 6.1, it is also possible to specify the _CardMax attribute between MetaClass and MetaAssociationEnd. This characteristic takes value "U" to impose uniqueness . Other possible values for this characteristic are "1", which means that the uniqueness is only for documentary purposes, and "N", which indicates there is no restriction.*

Only the maximum MetaMultiplicity is taken into account to impose uniqueness of a MetaAssociation. The minimum MetaMultiplicity of a MetaAssociationEnd is purely for documentary purposes. It does not give an obligatory MetaAssociation.

Specifying MetaAssociation Behavior

The MetaAssociationType, associated with its orientation, enables determination of the operation to be executed on each of its MetaAssociationEnds for the following functions: extraction, protection, object deletion, query isolated objects, comparison and impact analysis in the explorer. Main operators are the following:

_Operator	Used for
Extract	Object extraction
Delete (Propagate)	Object deletion
Protect	Object protection
Isolate	Query isolated objects
Compare	Comparison of repositories
Duplicate	Object duplication

Operators act differently depending on major/minor orientation of links and the MetaAssociationTypes associated with the MetaAssociations .

The behavior of an operator related to a MetaAssociation is specified in the "Operator Name Behavior" attributes calculated on each MetaAssociationEnd. This attribute is visible:

- in the **Behavior** tab of the properties of a MetaAssociation.

A MetaAssociation can have only one type. To modify the MetaAssociationType, you must disconnect it from its current type and then connect it to the new type.

This can be done in the metamodel diagram by opening the Properties window of each MetaAssociation.

You can directly modify the behavior of a link for a given operator in the **Behavior** tab of the MetaAssociation:

☛ *Link type processing during extraction:*

The extraction processes the values in the following order:

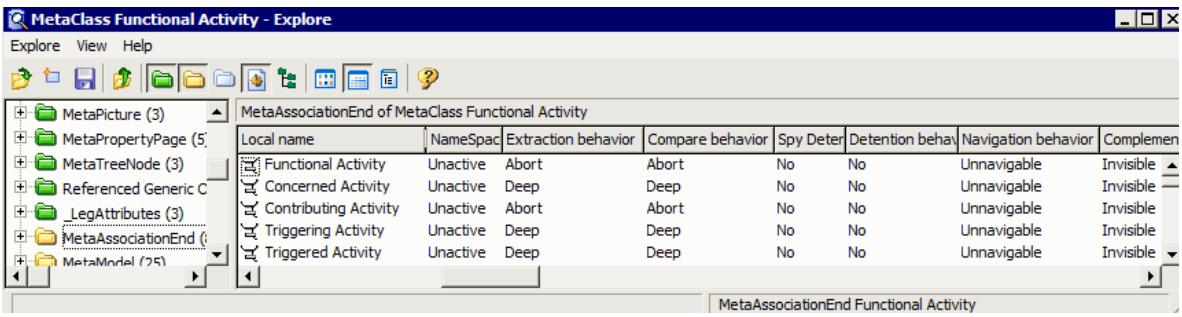
- *The value of the* "Minor to major behavior" *or* "Major to minor behavior" *characteristic (depending on whether the MetaAssociationEnd followed is major or minor) for the link between the link and the operator, if it exists.*
- *The value of the* "Minor to major behavior" *or* "Major to minor behavior" *characteristic of the link between the link type and the operator.*
- *The default value A (Abort) for* "Minor to major behavior" *and D (Deep) for*"Major to minor behavior" *if none of the previous links are specified.*

If the determined value is S (standard), the value of the _Scantype characteristic for the link between the object type (segment) and the operator is also used, if it exists.

This is the case for Entity, Relationship, MetaAssociationEnd , Rule, and Information, where the value is D (Deep).

This means the Entity, etc. is extracted plus the dependencies.

The behavior of an operator related to a link is specified in the calculated attributes on each MetaAssociationEnd.



MetaAssociationTypes

The "Composition" and "Association" MetaAssociationTypes can handle most current situations.

☛ *Confirm that the following option is: **Tools > Options**, select **Repository**, **Definition of path of MetaAssociation**, "From MEGA HOPEX 1.0".*

Composition

This MetaAssociationType indicates a strong dependency between the component and the assembled object. In particular, it indicates that the component has no meaning without this object (example: operations of a procedure).

The component and all its contents are included during extraction, duplication, and protection operations on the assembled object. It should be deleted when the assembled object is deleted.

Association

This MetaAssociation describes the major object, but the two objects are able to exist independently of each other. An example would be procedure and application. The minor object is included in an extraction, but is not copied, protected, or deleted with the major object.

MetaAssociation Types Kept to Maintain Compatibility

This MetaAssociationType (link type) was called "Description" in versions earlier than MEGA 2009 SP2.

To access these MetaAssociationTypes:

- Set option to : **Tools > Options > Repository**, **Definition of path of MetaAssociation**, "Compatibility up to MEGA 2009".

MetaAssociationTypes around a diagram

The MetaAssociation representing graphic content of a diagram is generally of composition type. For extraction of diagrams however, two additional MetaAssociationTypes are used:

Modeling

This MetaAssociationType indicates that one object is modeled by another (Description-Procedure MetaAssociations, etc.). This means that when a diagram is extracted, the object it describes will also be extracted.

Citation

This MetaAssociationType indicates a reference in a diagram to an object which is not intrinsically part of the diagram. For example, processes in a Flowchart need to be extracted with the objects describing them (Messages), while Org-Units should be extracted without their contents (such as their Organizational Chart). The MetaAssociation connecting these org-units to the diagram is citation type.

Other MetaAssociationTypes

alias

This MetaAssociationType is an alias of a generic link and inherits its behavior.

In this case, behavior need only be defined on the generic MetaAssociation. This being so, certain MetaAssociationTypes are no longer used. This is the case for example of "Documentation" and "Annotation" MetaAssociationTypes.

Aggregation

The dependency between the objects concerned is also strong in this case, but the component can exist without the aggregated object, for example: org-units of a flowchart, which can exist in an organizational chart even after the flowchart no longer exists.

These links are processed as for a composition link, except for deletion (the component should not be deleted since it may be used elsewhere) and copying (the component is linked to the copy without itself being copied).

Definition

The link is indispensable in understanding the defined object, for example: synchronization to message. Adding specifications to Message has no effect on synchronization. However, deleting the Message makes Synchronization null and void.

The MetaAssociation to the minor object is processed as for a description link (see below). The major object should be deleted when the minor object is deleted.

Sequence

This MetaAssociationType represents time sequencing of two independent phenomena. Modification or deletion of one of the objects has no impact on the other. Example: Next operation.

Flow

These MetaAssociations express exchange of a flow or message between two objects. Example: Message exchanged between two org-units.

Classification

These MetaAssociations enable classification of objects using keywords or other criteria.

Disappearance of the classification has no effect on the objects concerned.

Check

These MetaAssociations express a check to be carried out on an object. Example: MetaAssociation between constraint and operation.

Transformation

These MetaAssociations express transformation of an object from one type to another. Example: A class produces a table in a database.

Sub-Typing

These MetaAssociations express that one of the objects is a particular case of the other. The sub-type is the major object since it inherits all modifications to the super-type.

Extraction of the sub-type also necessitates extraction of the super-type.

Bivalent UML

These MetaAssociations express a strong relationship between two objects. Neither one has any meaning without the other. Example: link between role and association. The two objects are always extracted together.

Hierarchy

These MetaAssociations express a strong dependency of one object on another. Example: MetaAssociation between package and class or between database and table. Deletion of the major object also deletes the minor.

Variance

This MetaAssociationType enables connection of a variant to the object of which it is a variant.

Processing MetaAssociationTypes

Key:

- **-**: the link is not processed.
- ***D***: downward processing, includes the linked object, then follows the associations from that object. The objects at the end of the links may be included depending on the link type.
- ***S***: standard processing, the object at the end of the link is included.
- ***L***: only the link to the object is included (not the object itself)

Link type	Example	Major Minor	Extraction Compar.	Protect	Dupli	Delete	Iso-late
COMPOSITION	Procedure/Operation; Application/Tool; Project/Diagram, Tool, Function, Document ; ...	Aggregation-of Component	- D	- D	D	- -	- S
AGGREGATION	Operation/Rule, Timer ; Information/Entity ; Diagram/Entity, Relationship, MetaAssociationEnd, Constraint, Rule, Message, Operation, Condition, Record, Set, Sequencing, ... ; Record/Attribute; Identifier; RecordKey; Index; Diagram/Attribute (View-of-Diagram); ...	Aggregation-of Parameter	- D	- D	L	- -	- S
DESCRIPTION	Operation/Service; Org-Unit/Operation; Project/Application, Procedure; InstalledDiagram; Site/OrgUnit; Record/Language; ...	Described Description	- S	- -	L	- -	- S

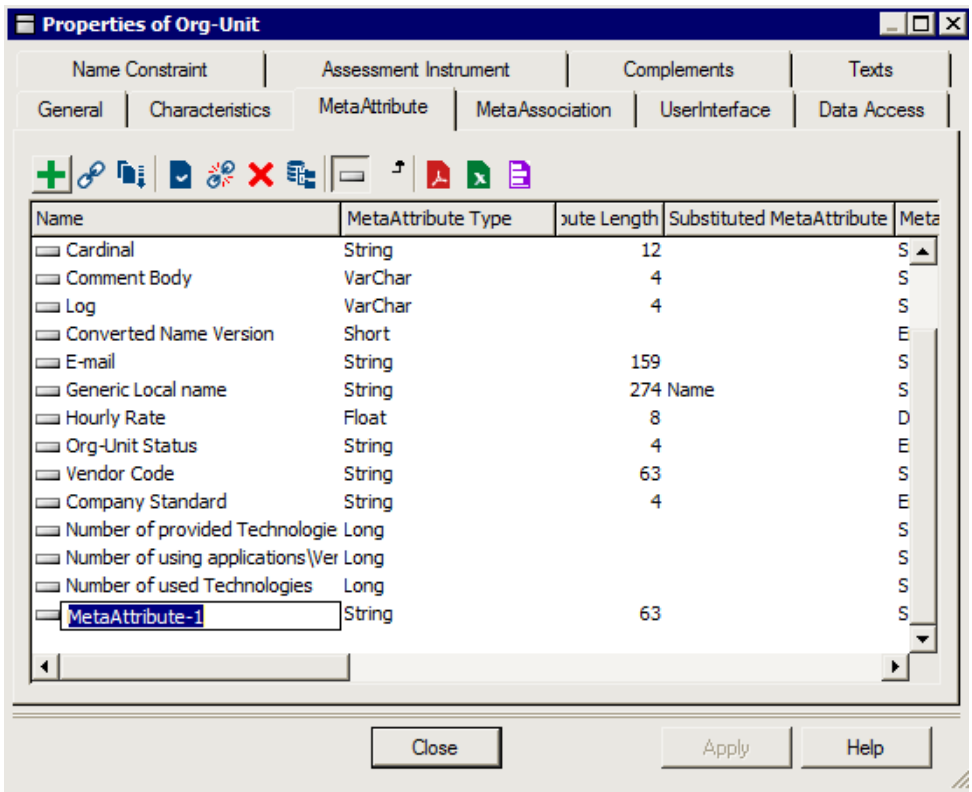
Link type	Example	Major Minor	Extraction Compar.	Protect	Dupli	Delete	Iso-late
DEFINITION	Logical Group/ Entity; MetaAssociationEnd/Entity; Record-Owner, Record-Member; Synchronization/ Message ...	Defined Defining	- S	- -	L	- -	- S
CLASSIFICATION	Keyword	Characterized Used	- S	- -	L	- -	- S
FLOW	Source-... ; Target-...	SrcTarget Source Target	- -	- -		- -	- -
TRANSFORMATION	Relationship/ Record; Organizational-Operation; ...	Derived Origin	- S	- -	L	- -	- S
SEQUENCE	Next; Timer/Message; ...	Next Previous	- -	- -		- -	- -
MODELING	Descriptor-.. ; Program/Tool; Analysis-Function	Modeled Diagram	S D	- D	L D	- -	- -
CITATION	Diagram/Procedure, Org-Unit, Site, Application, Tool, Information, Function, State, ...	Citing Cited	- S	- S	L	- -	- S
SUB-TYPING	Sub-Type; Equivalence, ...	Sub-Type Super-Type	- D	- -	L	- -	- -
BIVALENT	Output, MetaAssociationEnd/Relationship	Component Component	D D	D D	D D	- -	- -S
HIERARCHY	Database/Table; Table/Column, Key, Index		- D	- D	D	- D	- S
CHECK	Constraint/Entity, Information, Relationship	Checking Checked	D -	- -	L	- -	- -

METAATTRIBUTES

Creating MetaAttributes (Characteristics)

To create a MetaAttribute:

1. Open the Properties window of a MetaClass or MetaAssociationEnd.
2. In the **MetaAttributes** tab, click **New** .
   The new MetaAttribute is created and appears in the list.



☛ *When a MetaClass is created, the "Name" and "Comment" attributes are assigned to it automatically.*

☛ *The "Absolute identifier", "Authorization", "Creator Name", "Creation Date", "Modifier Name", and "Modification Date" attributes are also available automatically when an object type is created.*

☛ *The names of the attributes that can be used in queries are limited to 32 characters.*

The attribute format (MetaAttribute Type) must have a value, which can be one of the following:

Ab	MetaAttribute Type	Meaning	MetaAttribute Length
X	String	Alphanumeric	1 to 159
D	DateTime	Date	N/A
To	VarChar	ASCII text	N/A
B	VarBinary	Binary text (reserved)	N/A
1	Boolean	Boolean (0 or 1)	N/A
S	Short	Integer (0-65535)	N/A
L	Long	Integer (0 - 4294967295)	N/A
Q	Binary	Binary (reserved)	N/A
H	Double	Integer (0 - 18446744073709551616)	N/A
F	Float	Floating number	N/A

The "X" and "9" formats must have an indicated length (MetaAttribute Length).

With the Date format ("D"), the dates are entered in the format "YYYY/MM/DD HH:MM:SS". In windows where dates can be entered, the format used is the one defined in the Windows configuration.

MetaAttribute Format enables indication that external values are stored in a table (value T).

☛ *Tabulated characteristic values are saved with the technical data.*

Text type characteristic

By default, each new object type is connected to the "Comment" characteristic.

Characteristic MetaText Format specifies ANSI, RTF or HTML text format.

Tabulated characteristic

A tabulated characteristic is a characteristic associated with a series of tabulated values, such as Information-Class, Org-Unit-Type, and Message-Type.

You can change a standard characteristic to a tabulated characteristic after creation by setting MetaAttribute Format to "Enumeration" (F) or "Enumeration (opened)" (T).

To delete this characteristic, define its MetaAttribute Format as "S".

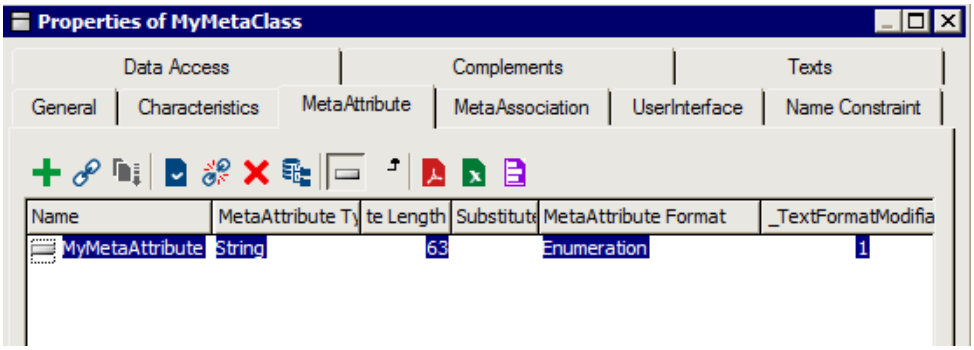
There are other possible types:

- "Duration" (D): for a date
- "Percent" (P): for a percentage
- "Double" (E): for a number
- "Object" (O): for an object
- "Signed Number" (Z): for a signed number.

To make a characteristic tabulated:

1. In the Properties window of a MetaClass, select the **MetaAttribute** tab.
2. In the **MetaAttribute Format** column of the desired MetaAttribute, select value "Enumeration" or "Enumeration opened".

> ☛ *The tabulated values associated with the tabulated characteristics are included in the technical data.: They can be modified using specialized data entry accessible in the **Characteristics** tab of the properties of the MetaAttribute.*



When you create a tabulated value, the Administration application automatically generates a name for this value. It should not be modified.

You must indicate the internal value actually stored in the attribute, and the external value to be listed during data entry, in the current language.

> ☛ *The attribute update version must be at least "16643" for it to be modifiable using this data entry.*

Specialized data entry is also accessible by selecting **Metamodel > Tabulated Values** in the pop-up menu of the environment in the Administration application.

> ☛ *For more information on tabulated values data entry, see "Editing Attribute Tabulated Values", page 43.*

MetaAttribute with unique index

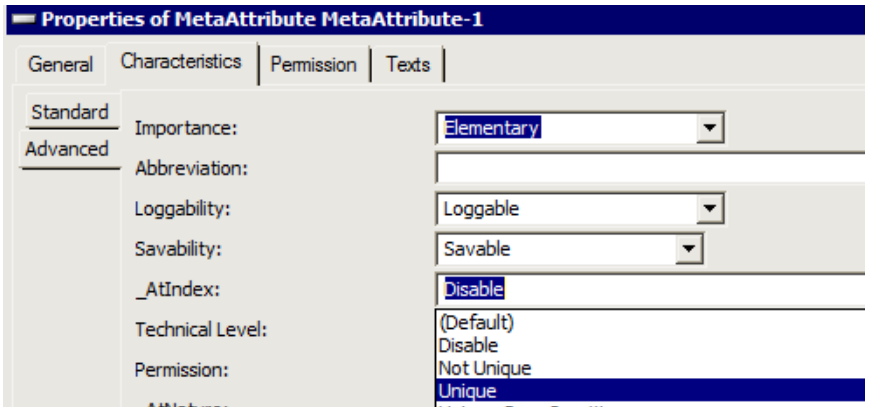
A unique index prevents the same value being assigned to two different attributes.

The **_AtIndex** attribute is accessible in the Properties window of a MetaAttribute in the **Advanced** subtab of the **Characteristics** tab.

To attribute a unique index to a MetaAttribute:

1. In the Properties window of the MetaAttribute, select the **Characteristics** tab, then the **Advanced** subtab.

2. For the **_AtIndex** attribute, select value **Unique**.



☛ *Attribute uniqueness is only effective in repositories created after uniqueness is declared. It is necessary to reorganize existing repositories.*

_AtIndex	Comment
0	No index
U	Unique index
N	Non-unique index

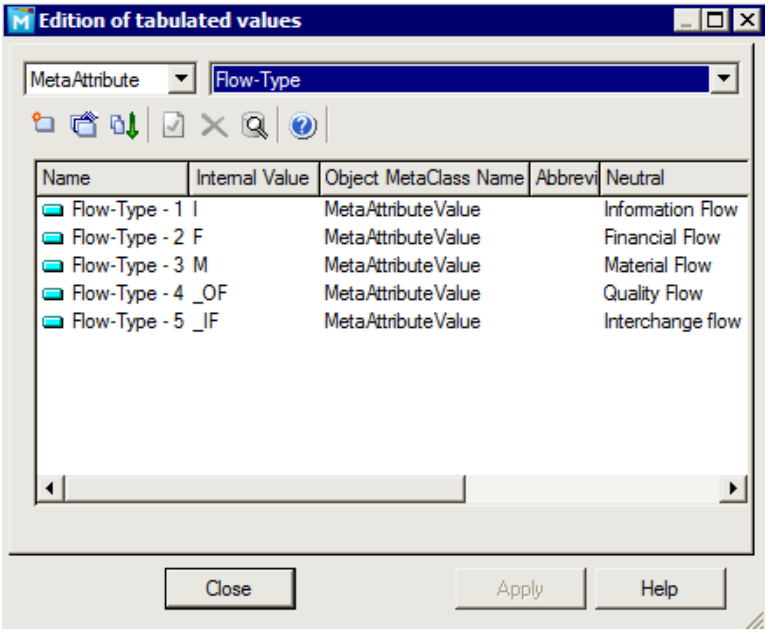
Editing Attribute Tabulated Values

A tabulated value of a characteristic is the set of predefined values that this characteristic can have, for example the values of the "Flow-Type" characteristic of messages. You can modify these values in the properties of the MetaAttribute or by using the **MEGA Administration** application.

To modify tabulated values of a MetaAttribute:

1. In **MEGA Administration**, log on to the desired environment.
2. Right-click the environment and select **Metamodel > Tabulated Values**.
   The dialog box that appears allows you to update characteristics of which values are saved in a table.

The **MetaAttribute** list box presents attributes of this type. Select an attribute to display the internal names and internal and external values in the table.



☛ *If you want to modify the value of a characteristic, do not modify the **Name** field. Modify the **English** field which contains the external value of the attribute.*

To modify the value of an element in the table, select it and enter its new value and corresponding internal value (the value stored in the repository).

To add a new value to the table, click **Create** 

Using VB Scripts to Calculate Characteristics

You can:

- define new attributes or parameters for a **MEGA** object.
- determine value read and save modes for this parameter using **MEGA** macros.

To calculate an attribute by creating a macro:

1. Open the explorer on the new attribute or new parameter.
2. Right-click the attribute (or parameter) and select **New > Macro**
   The macro creation wizard appears.
3. Select **Create Macro (VB)Script**.
4. Click **Next**.

5. (Optional) Modify the default **Name** ("AttributeName".Macro) of your macro.

   A macro is an object containing a VB Script code sequence interpreted at execution.

6. Click **Finish**.

Edit the "AttributeName".Macro macro and note that in particular the VB Script contains the following functions:

☛ *If they are not present, standard implementation is selected.*

- ***GetAttributeValue(ByVal Object, ByVal AttributeID, ByRef Value)***
  Define attribute access mode. The parameters are:
  - Object: corresponds to the object of which attribute value is requested.
  - AttributeID: absolute identifier of the attribute (or taggedValue).
  - Value: the function returns the attribute value for this object.
- ***SetAttributeValue(ByVal Object, ByVal AttributeID, ByVal Value)***
  Define attribute save mode. The parameters are:
  - Object: corresponds to the object of which the attribute value must be updated.
  - AttributeID: absolute identifier of the attribute (or taggedValue).
  - Value: the function saves the attribute value for this object.

☛ *The attribute nature (**_AtNature**) should be **Virtual**.*

For both of these functions, attribute change mode is a character string.

Conversion must be carried out to change text format to the internal format of the attribute.

Example:

```
Sub GetAttributeValue (ByVal object, ByVal AttID, Value)
   ' internal value reading in integer format.
   numValue = CInt(objet.GetProp(AttID, "Physical"))
      if numValue < 20 then
         Value = "Young"
      elseif numValue < 35 then
         Value = "Youthful"
      elseif numValue < 55 then
         Value = "Mature"
      else
         Value = "Elderly"
   end if
End  Sub
```

You can directly implement read-only and read/write access in the attribute format (without passing via standard conversion).

In this case, you must implement the following two functions, of which prototypes are similar to those above:

- GetExtendedAttributeValue(ByVal Format as LONG, ByVal Object, ByVal AttributeID, ByRef Value)
- SetExtendedAttributeValue(ByVal Format as LONG, ByVal Object, ByVal AttributeID, ByVal Value)

The difference is in the additional parameter: Format. The possible values are:

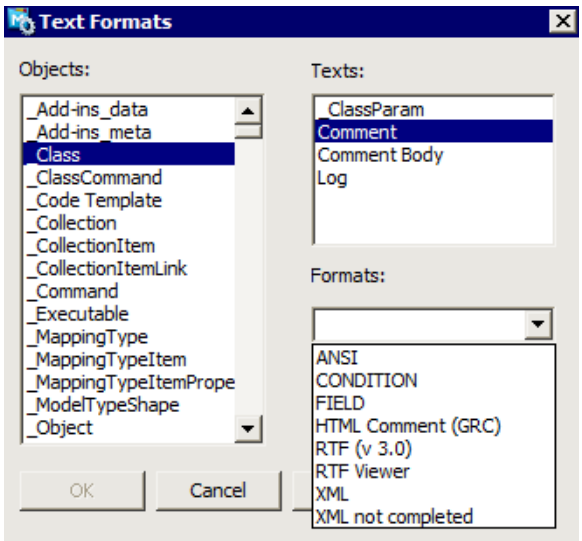
- 0 internal: value in internal format (binary, integer,...)
- 1 external: value in external format, but before display processing (certain objects have external form that is textual with the addition of index identifiers, as for class attributes or association roles).
- 3 Display: value in external format used in Web sites or Word documents (expurgated when identifiers in external format occur).

If one of the two extended functions is implemented, call by GetProp with "Physical" format on the same attribute is prohibited since it would lead to an infinite recursion.

Modifying Text Types

To indicate for each attribute how its comment is to be managed:

1. In **MEGA Administration**, log on to the desired environment.
2. Right-click the environment and select **Metamodel > Text Format**. The **Text Formats** dialog box opens.



You can indicate for each **Comment** whether the text should be ANSI, XML or RTF. You can also do this for other text types defined for the MetaClass. Note that RTF text saves all formatting such as bold, italic, etc.

☛ *Do not apply RTF format to technical text, such as text including supporting specifications.*

☛ *If the text format is specified as ANSI, the text is converted to ANSI format. If the comment of an object is in RTF while the text format is not specified, the comment of the object is read-only. The type of text must be specified in order to unlock the comment of an object.*

Connecting Attributes to a MetaClass

An attribute can be connected to one or several diagrams. The "Name" attribute is connected to all MetaClasses.

To connect an attribute to a MetaClass:

1. In the Properties window of the MetaClass, select the **MetaAttribute** tab.
2. Click **Connect** .
   The dialog box that opens asks you to execute a query to find the MetaAttribute you want to connect.
3. Execute the query.
4. Select the MetaAttribute.
   ☛ *You can select several MetaAttributes.*
5. click **Connect**.
   The selected MetaAttributes appear in the list of MetaAttributes of the MetaClass.

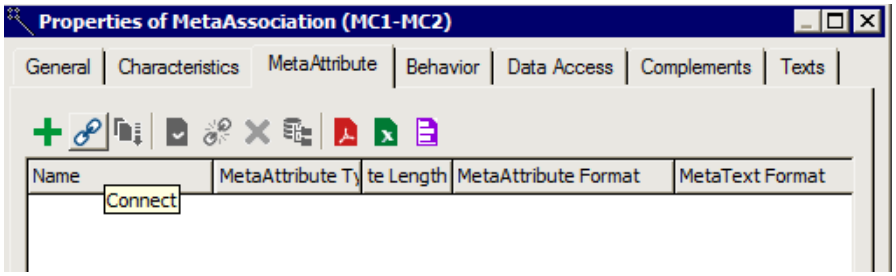
Connecting Attributes to MetaAssociations

A MetaAssociation cannot be connected to a text type attribute. An attribute can be connected to one or several MetaAssociations. The "Order" attribute is connected to all MetaAssociations.

☛ *A "Text" type attribute on a link is only accessible with certain tools such as the explorer, and not by others (diagrams, etc.).*

To connect an attribute to a MetaAssociation:

1. In the Properties window of a MetaAssociation, select the **MetaAttribute** tab.



2. Click **Connect** .
   The dialog box that opens asks you to execute a query to find the MetaAttribute you want to connect.
3. Execute the query.
4. Select the MetaAttribute.
   ☛ *You can select several MetaAttributes.*

5. click **Connect**.
   The selected MetaAttributes appear in the list of MetaAttributes of the MetaAssociation.

Abbreviations

Definition of abbreviations for the names of MetaClasses, attributes and text is optional. As standard, the metamodel is delivered with attribute abbreviations.

Standard Attributes

All MetaClasses have standard attributes:

- Name:
  - limited to 63 characters for MetaClasses without namespace.
  - limited to 255 characters for MetaClasses with namespace, of which 159 characters are for the local name.
- Internal identifier (absolute identifier calculated from the object creation date)
- Creation date
- Creator name (absolute identifier of the user who created the object)
- Last modification date
- Last modifier name (absolute identifier of the user who last modified the object)
- Comment
- Log

ABSTRACT METAMODEL

Basic Concepts

The abstract metamodel of the **MEGA** platform offers the possibility of managing the notion of inheritance for MetaClasses and MetaAssociations. This enables a significant reduction in the number of MetaAssociations.

Available from version **MEGA 2009**.

MetaClass types are:

- ***Concrete MetaClasses***: MetaClasses for which instances exist
- ***Abstract MetaClasse***s: generic MetaClasses used only to describe inheritance.

Abstract metamodel

The abstract metamodel on which the **MEGA** platform is based is described using metamodel diagrams. Graphical rules provide a view of basic concepts.

Abstract MetaClass

An abstract MetaClass is a MetaClass that does not have a concrete occurrence. It enables definition of common attributes to MetaClasses that inherit these.

When a concrete MetaClass inherits an abstract MetaClass, it inherits:

- MetaAttributes

  Example: MetaAttributes of the"BPMN Activity" MetaClass are: "Predicate", "Loop", "Ad hoc", "Multiple", etc.

- MetaAssociations,

  Example: the MetaAssociation inherited from the "Element with Note" MetaClass is: (Note Element/note) enabling linking of a Note object with an "Element with Note".

- Properties pages (MetaPropertyPage).
- Menu commands (MetaCommand).

Generic MetaAssociations

A generic MetaAssociation is a MetaAssociation common to a set of concrete MetaClasses. It is defined between an abstract and another MetaClass. All MetaClasses inheriting the abstract MetaClass inherit this "Generic" MetaAssociation by default.

For example, the generic MetaAssociation (Element with note/Note) defined between the "Element with Note" abstract MetaClass and the "Note" MetaClass exists for all concrete MetaClasses inherited from the "Element with Note" abstract MetaClass.

Managing Abstract MetaClasses

Creating an abstract MetaClass

To create an abstract MetaClass:

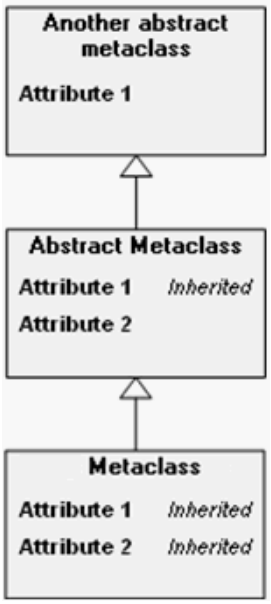
1. Check that you are in Advanced metamodel (from **Tools** > **Options** > **Repository**, **Metamodel Access**: "Advanced").
2. Create a new MetaClass (for more details, see "Placing a MetaClass in a Metamodel Diagram", page 25)
3. Right-click the MetaClass and open its Properties.
4. Select the **Characteristics** tab and select the **Standard** subtab.
5. In the **MetaClass Layer** field, select **Abstract**.
   Note that the MetaClass changes color in the metamodel diagram and that its icon appears transparent in the navigation tree.

You must not change an existing concrete MetaClass into an abstract MetaClass. Occurrences of this MetaClass will no longer be accessible. See "Abstract Metamodel Extension Recommendations", page 57.

Inheritance relationships

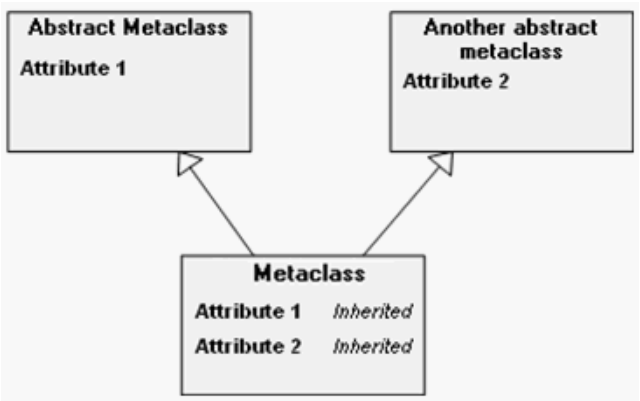
MEGA inheritance is hierarchical and multiple:

- Hierarchical, since each MetaClass hierarchically inherits all characteristics of the abstract MetaClass from which it inherits.



Inheritances of several levels are not recommended.

- Multiple, since a MetaClass can inherit several MetaClasses.



A MetaClass of the "System" repository cannot inherit an abstract MetaClass of the "Data" repository.

The list of MetaClasses from which a MetaClass inherits is accessible from the Properties window of the MetaClass, in the **Characteristics** tab, **Standard** subtab. The **SuperMetaClass** field allows you to view and update this list.

To view with the explorer the list of MetaClasses from which a MetaClass inherits (for example: "Element with Note"):

1. Right-click the MetaClass and select **Explore**.
   An explorer window opens.
2. Expand the "SuperMetaClass" folder.

   ☛ *Similarly, by expanding the "SubMetaClass" folder, you obtain the list of all MetaClasses that inherit the current MetaClass.*

Managing Generic MetaAssociations

A generic MetaAssociation is a MetaAssociation defined between an abstract MetaClass and another MetaClass.

All MetaClasses inheriting the abstract MetaClass inherit this "Generic" MetaAssociation by default.

You can specify that a MetaAssociation already existing between two concrete MetaClasses is in fact an alias of a generic MetaAssociation. In this case, this MetaAssociation inherits all the attributes of the generic MetaAssociation. These attributes cannot be redefined.

```
For example, the MetaAssociation "Source Activity" / "Sent
Message" is an alias of the generic MetaAssociation
"Message Source" / "Sent Message" inherited from the
abstract MetaClass "Messaging Participant" by the concrete
MetaClass "Functional activity".
```

If you use aliases, you must remember to convert your repositories and respect recommendations relating to aliases. For more details, see "Abstract Metamodel Extension Recommendations", page 57.

A MetaAssociation can be defined as an alias of a generic association for two reasons:

- Because the MetaAssociation existed before creation of the generic MetaAssociation representing the same link. This MetaAssociation should be kept for compatibility reasons.
- Because the MetaAssociation represents reduction of the generic MetaAssociation to a concrete MetaClass.

Managing MetaAssociation inheritance

To define that an alias on a MetaAssociation is the alias of a generic MetaAssociation:

1. Check that you are in Advanced metamodel (from **Tools** > **Options** > **Repository**, **Metamodel Access**: "Advanced").
2. Open the properties of the MetaAssociation you want to modify.
3. Select the **Characteristics** tab then the **Advanced** subtab.
4. In the **Super MetaAssociation** field, select the generic MetaAssociation that interests you.

5. In the **Meta Specialization Type** field, select:

- **Depreciated** if the alias is defined to assure compatibility.
- **Restrictive** if the alias is defined to reduce the number of MetaClasses concerned by the generic MetaAssociation.

Accessing inherited MetaAssociations

To view the list of MetaAssociations that are aliases of a generic MetaAssociation:

1. Right-click the MetaAssociation and select **Explore**.
2. Expand the "Restricting MetaAssociation" folder to obtain lists of restrictions of this MetaAssociation.

> ☛ *Similarly, expand the folder "Restricted MetaAssociation" of a MetaAssociation from which it inherits and you obtain the generic MetaAssociation.*

Viewing types in a navigation tree

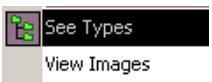
A generic MetaAssociation can connect objects of different MetaClasses.

In a navigation tree, a generic MetaAssociation appears as a branch to a list of objects of heterogeneous MetaClasses.

```
For example, if you use the explorer on an organizational
process described by an organizational process diagram, the
navigation tree will display an "Owned Element" branch.

If you expand this branch, you will see a list of objects of
different MetaClasses.
```

To store these objects in the explorer as a function of their MetaClass:

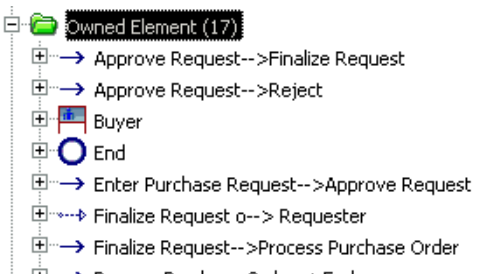
1. Click **View > Option**.
   The list of options appears.
2. Select **View types**.



If you expand the "Owned element" folder, you will see the sub-folders corresponding to each type.

To view icons associated with MetaClasses:

- In the same way, select **View > Option > View Images**.



> ☛ *To view repository objects by type, in the **Home** navigation tree, right-click the repository name and select **Regroup by Type**.*

Adapting Development Tools to the Abstract Metamodel

The use of abstract MetaClasses and generic MetaAssociations modifies tools of the **MEGA** platform.

Properties pages

To access the list of MetaAttributes of an abstract MetaClass:

1. Open the Properties window of the MetaClass.
2. Select the **MetaAttribute** tab.
   Note the three subtabs:
   - **MetaAttribute**, groups the list of MetaAttributes specific to the current MetaClass.
   - **SuperMetaAttribute**, groups the list of inherited MetaAttributes.
   - **Standard**, groups the MetaAttributes automatically inherited by any new MetaClass. The format of these attributes is defined as"Standard" in the **MetaAttribute Format** field of the **Characteristics** tab. For more details, see "MetaAttributes", page 40.

To access the list of MetaAssociations inherited from an abstract MetaClass:

1. Open the Properties window of the MetaClass.
2. Select the **MetaAssociation** tab.
   Note the two subtabs:
   - **MetaOppositeAssociationEnd**, groups the list of MetaAssociations specific to the current MetaClass
   - **SuperMetaOppositeAssociationEnd**, groups the list of inherited MetaAssociations.

Abstract metamodel diagrams

In an abstract metamodel diagram, graphic functionalities enable visual differentiation of abstract metaclasses, inheritance relationships and generic links.

To view metamodel diagrams proposed in the **MEGA** platform:

- In the **MetaStudio** navigation window, expand the **Metamodel** folder, then the **MEGA Modeling** folder.

For example, to access the metamodel diagram concerning libraries:

1. Confirm that you are in Advanced metamodel (**Tools > Options** > **Repository** > **Metamodel Access** > "Advanced").
2. In the **MetaStudio** navigation window, expand the **Metamodel** folder, then the **MEGA Modeling** folder.
3. Open the metamodel diagram "Library & Packaging Model".

Note that:

- The "Library" MetaClass is grayed
- The abstract MetaClasses "Owner Packager", "Packaged Element" and "Library Element" appear in color
- A directional link indicates that the "Library" MetaClass inherits the "Container" MetaClass
- A generic MetaAssociation is defined between the "Container" abstract MetaClass and the "Containable Element" abstract MetaClass.

Query Tool

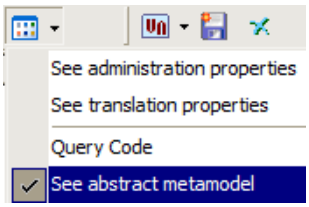
Querying from an abstract MetaClass

The query tool enables access to an occurrence via an abstract MetaClass from which it inherits.

```
For example: Select [Library element]
```

To access abstract MetaClasses in the query tool:

1. Open the query tool.
2. Click **Show**  and select **See abstract metamodel**.



You then have access to abstract MetaClasses.

Querying from a generic MetaAssociation

The query language of **MEGA** has evolved to take account of generic MetaAssociations. Operator "**:**" enables targeting of a query result on a specific MetaClass.

```
For example, the generic MetaAssociation"Message sender"/
"Message sent" connects the MetaClass "Message" to the
abstract MetaClass "Message participant" from which the
"Operation" MetaClass inherits. To access the list of
messages the recipient of which is an operation, it is
possible to use the MetaOppositeAssociationEnd "Message
recipient" and restrict the result using operator ":".
Syntax of the query is as follows:

Select [Message] where [Message Recipient]:[Operation]
```

☛ *When the concrete MetaClass has been specified, you can add to the selection MetaAttributes and MetaAssociations belonging to the concrete MetaClass.*

Evolution of APIs

A consequence of the abstract metamodel is that conventional MetaAssociations, MetaAssociationEnds and MetaAttributes are no longer systematically accessible from a concrete MetaClass.

New APIs, using the compiled metamodel are proposed, to enable optimized and precise access to abstract metamodel concepts. These are generally based on "MegaObject" and "MegaCollection" concepts.

Accessing abstract MetaClasses

Objects handled in **MEGA** are attached to a concrete MetaClass inherited from one or several abstract MetaClasses.

The **GetType** function allows an object or collection to be considered as an instance of a given MetaClass.

Used from a ***MegaObject***, the **GetType** function enables consideration of the object as a function of the MetaClass given as parameter.

```
Example: MyOrgProc.GetType("BPMN Owner Element").explore
runs the explorer from an object of the Organizational
Process concrete MetaClass, considering it as an object of
the "BPMN Owner Element" MetaClass.
```

☛ *Used without a parameter, operator **GetType** enables consideration of the current object as an element of the concrete MetaClass to which it belongs.*

Used from a ***MegaCollection*** of objects of different concrete MetaClasses, operator **GetType** allows you to obtain a collection restricted to instances of the MetaClass specified as parameter.

```
Example: oCollection.GetType("Sequence flow").explore runs
the explorer on objects of the collection belonging to the
"Sequence flow" MetaClass.
```

If no object of the collection inherits the MetaClass given as parameter, operator **GetType** returns nothing. No error is generated.

Notions of class and collection description

APIs enabling browsing of the abstract metamodel are accessible from two main entry points:

- "ClassDescription" which finds the MetaClass of a "MegaObject" from the function **GetClassObject** .

  `Example:` myMegaObject.GetClassObject.Explore
- "CollectionDescription" which enables discovery of the interface of a "MegaCollection" from the **GetTypeObject** function.

  `Example:` myMegaCollection.GetTypeObject.Explore

API s available from a class description

Among available APIs, APIs relating to the abstract metamodel are:

- **UpperClasses**: returns the list of inherited MetaClasses, viewed as ClassDescription
- **LowerClasses**: returns the list of MetaClasses, viewed as ClassDescription, inheriting the given MetaClass..

☛ *These APIs return a collection accessible by **GetCollection***

APIs available from a collection description

The main APIs available are:

- **TargetClassID**: returns identifier of the target MetaClass (of the collection)
- **SourceClassID**: returns identifier of the source MetaClass (of the collection)
- **TargetTypeID**: returns identifier of the target MetaClass, abstract in the case of a generic MetaAssociation
- **SourceTypeID**: returns identifier of the source MetaClass, abstract in the case of a generic MetaAssociation
- **AliasID**: returns identifier of the alias of the collection if this exists
- **RootID**: returns identifier of the generic association if we are on an alias

☛ *These APIs return a property accessible by **GetProp***

- **IsSuperClassOf ()**: returns a positive boolean if the MetaClass passed as argument inherits the MetaClass from which the collection is built
- **IsSubClassOf ()**: returns a positive boolean on elements of the collection that inherits the MetaClass passed as argument
- **IsClassAvailable()**: tests if an object of the MetaClass passed as argument can be inserted in the collection

☛ *These APIs are functions with argument*

Abstract Metamodel Extension Recommendations

Given that an extension of the abstract metamodel impacts MetaClasses and occurrences of the repository, and that modifications can be lost when installing a new version of **MEGA**, you must respect certain rules before intervening on the abstract metamodel.

Recommendations to be taken into account are the following:

- You can add abstract MetaClasses.
- Inheritances of several levels are not recommended.
- You must not modify status of MetaClasses of **MEGA**, change parameters, or add MetaAttributes.
- You can add generic MetaAssociations, but they cannot relate to abstract MetaClasses of **MEGA**. They can relate to a concrete MetaClass of **MEGA**.
- You can add MetaAttributes, on condition that they do not relate to abstract MetaClasses of **MEGA**. They can be connected to a generic MetaAssociation of **MEGA**.
- You should avoid creating an abstract MetaClass simply because a MetaPropertyPage is common to several MetaClasses. It is preferable to configure different MetaPropertyPages on the adapted MetaClasses.
- You should avoid creating an abstract MetaClass simply because a MetaPropertyPage is common to several MetaClasses. It is preferable to configure different MetaCommands on the adapted MetaClasses.
- When you define a MetaAssociation that already exists as an alias of a generic MetaAssociation, you must convert all repositories that use the MetaAssociation defined as alias.
- If a MetaAssociation **MEGA** is defined as alias, this alias cannot be removed.
- If you remove an alias on a MetaAssociation that is not **MEGA**, you risk loss of consistency of your repositories and loss of links.
- The MetaAttributes of the MetaAssociation defined as alias must be identical to the MetaAttributes of the generic MetaAssociation.
- If a MetaAssociation defined as alias has additional MetaAttributes compared with the generic MetaAssociation, these MetaAttributes must be defined at the level of the generic MetaAssociation.

A perimeter enables building a set of objects and links from a root object.

PERIMETERS

Introduction to Perimeters

Perimeters enable configuration of the propagation mechanism which enables building of a set of objects from one or several objects called root objects.

This mechanism is used by certain tools to apply processing to this set.

`Examples: Export, Deletion, Duplication.`

☛ *See use of perimeters with MetaTools **Export** or **Compare and Align** in the guide **MEGA Administration-Supervisor**.*

MetaTool

A MetaTool is an object that enables definition of perimeters that can be used with a tool.

There is a MetaTool for each tool that uses the propagation mechanism.

To build this set of objects, the MetaTool applies a perimeter to the root object.

☛ *You can assign a default perimeter for the MetaTool (see "Modifying MetaTool Default Perimeter", page 66).*
The aim of this perimeter is to provide a standard behavior if no customized perimeter exists in this context.

Propagation

Propagation is the platform mechanism which enables building of a set of objects from one or several objects called root objects.

The propagation principle is to include in the resultant set all the objects connected to an object, then the objects connected to the connected objects, and so on. The resultant set comprises all objects directly or indirectly linked to root objects.

Certain links are taken into account and others not: for a given object, inclusion or non-inclusion in the resultant set of objects linked to the initial object according to link type is called propagation behavior for this link type. Perimeters enable definition of propagation behavior of link types.

Perimeters are associated with MetaTools. MetaTools represent tools that use perimeters to indicate link type propagation behaviors when using a given tool.

Table: Description of propagation behaviors

Value	Icon	Propagation description
Deep		***Recursive complete propagation***: Takes into account this link and the opposite object only. Propagation continues.
Standard		***Simple propagation***: Takes into account this link and the opposite object only. Propagation stops.
Link		***Limited propagation:*** Takes into account this link but not the opposite object. Propagation stops.
Abort		***No propagation:*** Does not take into into account this link or the opposite object. No propagation:
Computed		***Propagation dependent on context*** Link type does not enable determination of perimeter behavior of this object. Propagation depends on context; it is defined by a macro and can take values "Deep", "Standard", "Link" or "Abort".See. See "Creating a New Perimeter", page 64.

Scope

Propagation behavior is defined by the perimeter.

Each MetaAssociation is linked to a major MetaClass and a minor MetaClass. The perimeter determines the value (Deep, Standard, Link, Abort, Computed) of its MetaAttributes MajorToMinor and MinorToMajor.

☛ *The value of MetaAttributes can also be defined by the MetaAssociationType associated with the MetaAssociation.*

In this example:

- if MetaclassA is taken as root object, propagation behavior to be taken into account is the value of the MetaAttribute MajorToMinor.

  ☛ *Depending on this value, the link and the opposite object MetaClassB are added or not to the set of objects and links.*

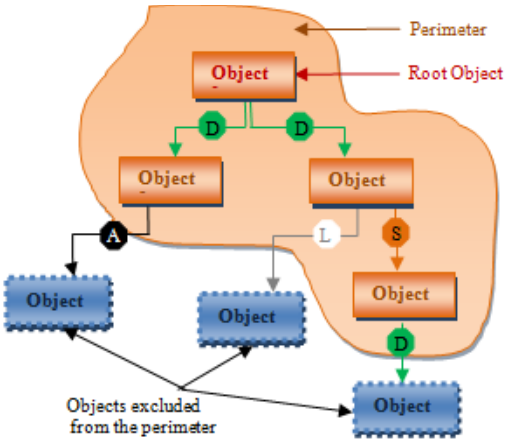
- if MetaclassB is taken as root object, propagation behavior to be taken into account is the value of the MetaAttribute MinorToMajor.

  ☛ *Depending on this value, the link and the opposite object MetaClassA are added or not to the set of objects and links.*

Propagation example

The following example presents object-to-object propagation, from root object "Object 1":

- The perimeter includes object 2 (link ***Deep*** with object 1).
  Propagation continues and excludes object 4 (link ***Abort*** with object 2).
- The perimeter includes object 3 (link ***Deep*** with object 1).
  Propagation continues and:
  - excludes object 5 (link ***Link*** with object 3).
  - includes object 6 (link ***Standard*** with object 3).
    Propagation stops, object 7 is not processed and is excluded despite its ***Deep*** link with object 6.



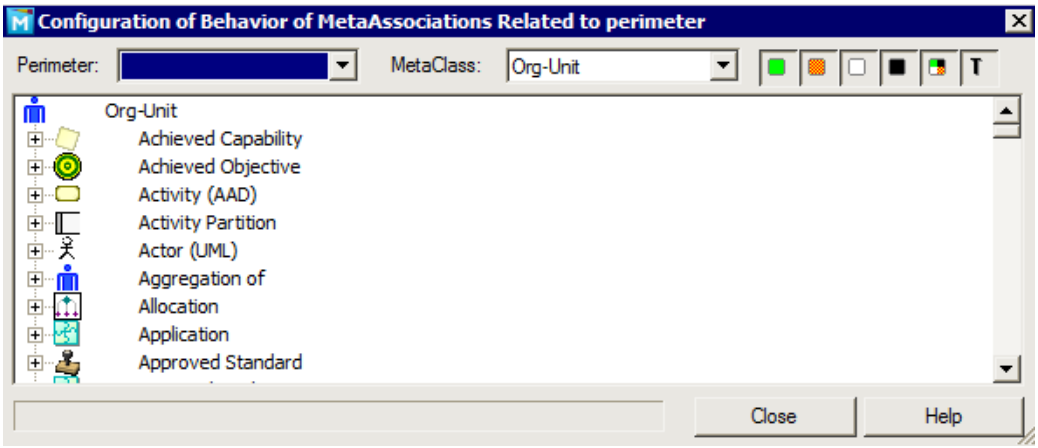
Viewing MetaAssociation Behavior Related to a Perimeter

To view MetaAssociation behavior related to a perimeter:

1. In the **MetaStudio** navigation window, expand the **MetaClass** folder, then **MetaModel**.

2. Right-click the MetaClass concerned and select **Manage > Configure**. The **Configuration of Behavior of MetaAssociations Related to Perimeter** dialog box appears.

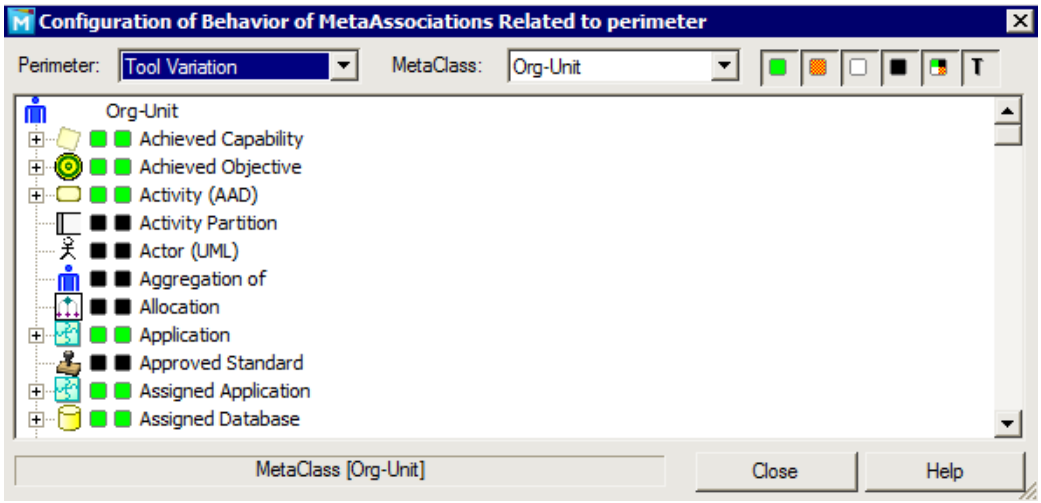
   ☛ *In the **MetaClass** box, the MetaClass concerned is already selected. You can modify this selection via the drop-down menu.*



3. In the **Perimeter** box drop-down menu, select the perimeter you wish to study.

   ☛ *Only those perimeters with _OpPreview characteristic value "Active" are visible.*

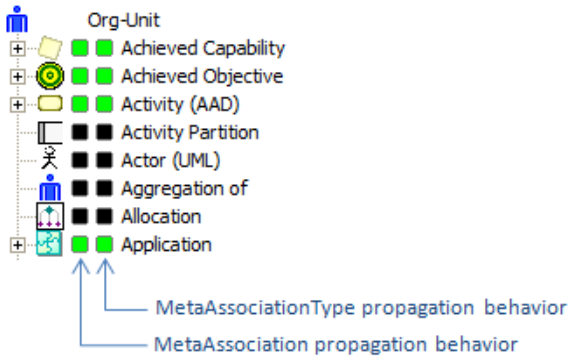
   The **Configuration of Behavior of MetaAssociations Related to Perimeter** dialog box is updated.



The **Configuration of Behavior of MetaAssociations Related to Perimeter** dialog box details for the selected MetaClass (here "Org-Unit") the behavior of the selected perimeter (here "Tool Variation") for each MetaClass concerned:

- propagation behavior of each MetaAssociation (link)
- propagation behavior of each MetaAssociationType (link type)

> ☛ *For description of icons, see "MetaTool", page 59.*



You can filter display:

- To show only certain propagation types, click **Deep**, **Standard**, **Link**, **Abort** and/or **Computed**.
- To show/hide behavior of MetaAssociationTypes (link default behavior before customization), click .
  A second column appears/disappears at the right of the propagation types column.

> ☛ *indicates that propagation of the MetaAssociation is different from the MetaAssociationType.*

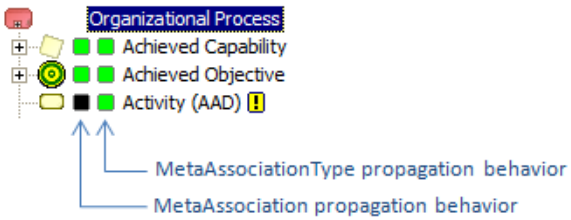
Modifying MetaAssociation Behavior Related to a Perimeter

You can modify MetaAssociation behavior related to a perimeter.

To modify MetaAssociation behavior related to a perimeter:

1. Open the **Configuration of Behavior of MetaAssociations Related to Perimeter** dialog box, see "Viewing MetaAssociation Behavior Related to a Perimeter", page 61.
2. Right-click the MetaAssociationEnd.

3. In the **Behavior** section, select **Propagation**, then the propagation behavior.
   - The icon on the left, representing the behavior of the perimeter on the MetaAssociation, is updated according to the selected propagation behavior.
   - The ! icon indicates that propagation of the MetaAssociationEnd is different from propagation of the MetaAssociationType.



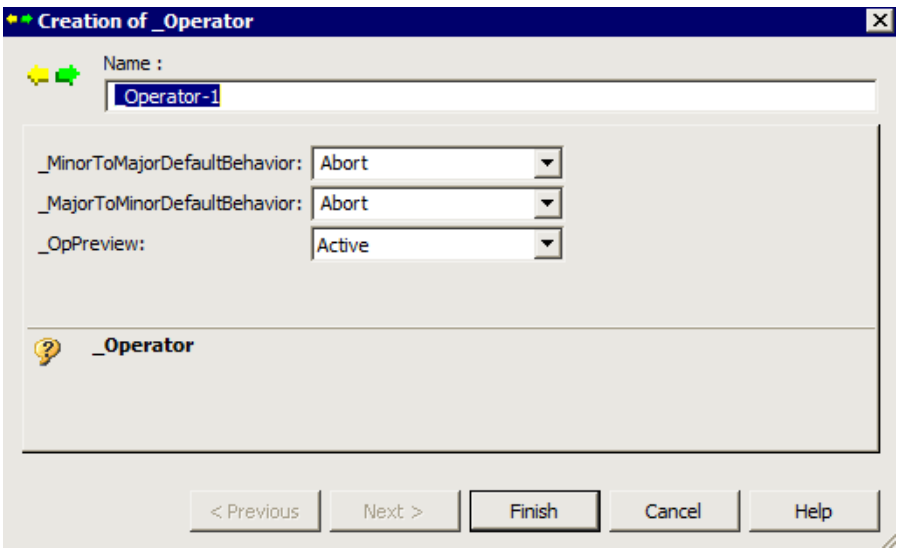
The new behavior is active from next use, except for "Computed" behavior, which required additional configuration.

☛ *"Computed" behavior uses a macro (_BehaviorMacro which must be previously created and linked to the current link and perimeter, see "Using VB Scripts to Calculate Characteristics", page 44.*

Creating a New Perimeter

To create a new perimeter:

1. In the **MetaStudio** navigation window, expand the **Perimeters** folder.
2. Right-click the **Customized Perimeters** folder and select **New > Customized Perimeters**.



3. Enter the **Name** of the perimeter.

4. Define characteristics of the perimeter:

   ☛ *For information on how to configure propagation, see "Introduction to Perimeters", page 59*

   - **_MinorToMajorDefaultBehavior** represents the default behavior of links of major objects to minor objects.

     ☛ *The default value "Abort" (links not propagated) avoids generating large volumes of objects.*

   - **_MinorToMajorDefaultBehavior** represents the default behavior of links of minor objects to major objects.

     ☛ *The default value "Abort" (links not propagated) avoids generating large volumes of objects.*

   - **_OpPreview** enables display of the perimeter in configuration of the MetaTool (see "Viewing MetaAssociation Behavior Related to a Perimeter", page 61).

     ☛ *Default value "Active"*

5. Click **Finish**.
   The new server appears in the list of **Customized Perimeters**.

   ☛ *To use this perimeter with a MetaTool, see "Using a Perimeter in a MetaTool", page 65 then "Modifying MetaTool Default Perimeter", page 66.*

Using a Perimeter in a MetaTool

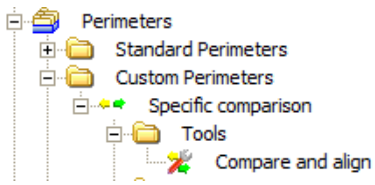
To use a perimeter in a MetaTool, you must connect this perimeter to the MetaTool.

☛ *You can connect a perimeter to several MetaTools, enabling reuse of the same perimeter in different contexts.*

To connect a perimeter to a MetaTool:

1. In the **MetaStudio** navigation window, expand the **Perimeters** folder.
2. Expand the **Standard Perimeters** or **Customized Perimeters** folder concerned (see "Creating a New Perimeter", page 64).
3. Expand the perimeter concerned.
4. Right-click the **Tools** folder and select **Connect > Tool**.
5. Using the **Query** tool, select the MetaTool concerned.
   The MetaTool is added to the **Tools** folder.

```
In the following example, the "Compare and Align" MetaTool
is connected to the "Specific comparison" perimeter
```



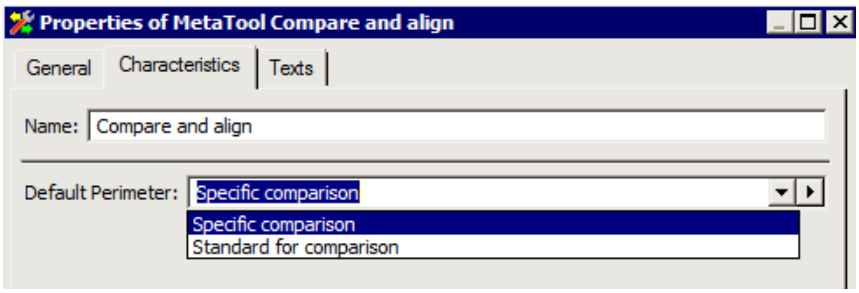
Modifying MetaTool Default Perimeter

☛ *The default perimeter you wish to define for the MetaTool must previously be connected to the MetaTool, see "Using a Perimeter in a MetaTool", page 65.*

A MetaTool can be connected to several perimeters, which enables building different sets of objects and links from the same root object.

To modify MetaTool default perimeter:

1. In the **MetaStudio** navigation window, expand the **Perimeters** folder.
2. Expand the **Perimeters by Tools** folder.
3. Open the properties of the MetaTool concerned.
4. In the **Characteristics** tab, modify the value of **Default Perimeter** via the drop-down menu.



The default perimeter of the MetaTool is modified.

NAMESPACES

The following points are covered here:

Managing Namespaces

Uniqueness of object name is normally checked against all objects of the same MetaClass in the repository. It is however possible to limit uniqueness check of a name to a particular context known as the namespace.

You can for example use two different conditions with the same name in two different procedures.

```
The condition "If approved" in the "Purchasing Processing"
procedure has complete name "Purchasing Processing::If
approved".
```

```
The condition "If approved" in the "Order Processing"
procedure has complete name "Order Processing::If
approved".
```

These two conditions are two different objects in the repository and can be connected to different operations.

When an object is included in a namespace, two names are presented:

- Local name defined as below of which length is limited to 140 characters.
- Name comprising the local name preceded by the name of the owner object (namespace). Display of this name is limited to 255 characters. If length to be displayed exceeds this limit, "..." are displayed in the middle of the name to replace missing characters.

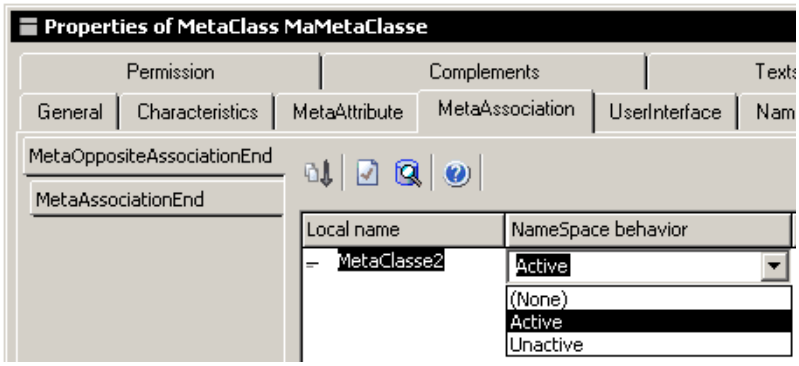
> ☛ *You can define namespaces in cascade. Example: A message owned by a collaboration itself owned by a business area. In this case, all successive namespaces are displayed in the object name.*

Defining Namespaces

To include a MetaClass in a namespace:

1. Check that you are in "extended metamodel" mode.
2. Open the Properties window of the MetaClass concerned.
3. In the **MetaAttribute** tab, click **Connect**  to connect the **Generic Local Name** MetaAttribute.
   The dialog box that appears asks you to execute a query to find the desired MetaAttribute.

4. Execute the query and connect the **Generic Local Name** MetaAttribute.
5. Select the **MetaAssociation** tab, then the **MetaOppositeAssociationEnd** subtab.
6. Select **Active** for the **Local Name Behavior** property of the MetaAssociationEnd displayed in the tab.

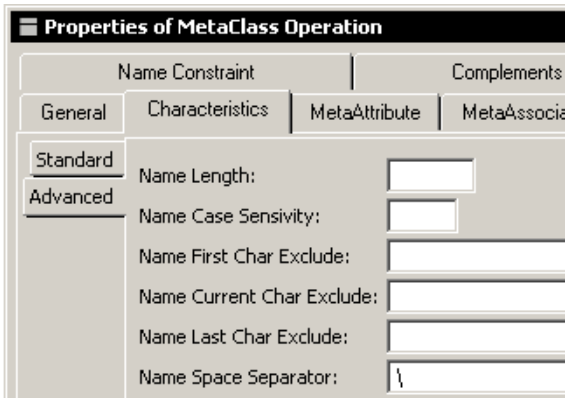


The MetaClass is now in a namespace, with "::" as default separator.

Customizing the separator

To customize the separator, you must modify **NameSpaceSeparator** in the MetaClass Properties window. You can use:

- a simple string to replace "::".



- a string containing the expressions "%M" and "%S".
  - "%M" indicating the namespace (the parent)
  - "%S" indicating the object in the namespace (the child)

☛ *Respect character case.*

When the form of separator with expressions "%M" and "%S" is used, certain query by name functions cannot be used.

Complementing specification of the MetaAssociation used for the namespace

So that deletion of the namespace also deletes the objects owned by this namespace:

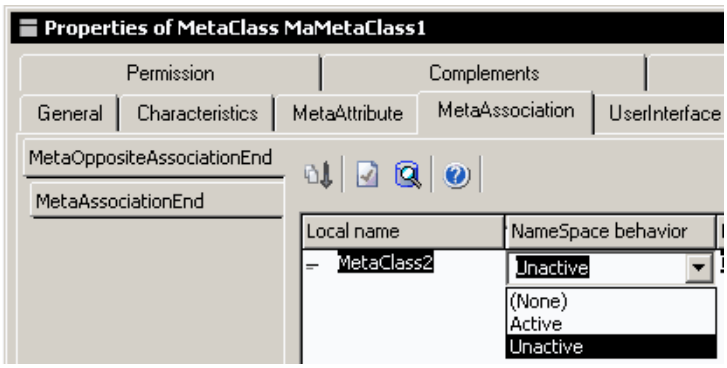
- Connect the MetaAssociation between the MetaClass and its namespace using a "Hierarchy" link type.
  - ☛ *This option is recommended.*

Canceling Namespaces

MetaClasses connected to GenericLocalName

To cancel a namespace on a MetaClass (when the namespace has been created using the method described in "Defining Namespaces", page 67):

1. Open the Properties window of the MetaClass concerned.
2. From the **MetaAssociation** tab select the **MetaOppositeAssociationEnd** subtab.
3. Deactivate local name behavior.



Other MetaClasses

To cancel a namespace on a MetaClass:

1. In the explorer, open the MetaClass for which you want to cancel the namespace.
2. Select the MetaAttribute that has "0000000040000002" as a value for the "Substituted MetaAttribute" attribute.
3. Remove value "0000000040000002" of the Substituted MetaAttribute attribute from the link between the MetaAttribute and the MetaClass.
4. Disconnect the MetaAttribute defined as local name for the MetaClass from "GBML NameSpace Server".

Ownership and Use Links

An ownership link indicates that an object (example: an activity) belongs to an owner object (example: a process).

However, in certain cases we may wish to reuse the same object (an activity) in another context (in another business process). The object (the activity) will then be connected to the other context by a use link. An object can have only one owner.

So as to avoid exploring the two links to find all objects (example: the activities), a spy controller automatically connects via the use link all objects connected by the ownership link.

This mechanism exists between Business Process and Activity, Procedure and Operation, etc.

PROBLEMS AT IMPORT

Problems Encountered At Import

Extensions may not be imported correctly for the following reasons:

Text format

The command file can be created with any word processor, such as Word, but it must be saved in Text Only format. If it is in Word format, it will not be recognized by the command interpreter.

☺ *Use Notepad, Write, or Wordpad and avoid formatted text.*

Open transactions

Metamodel extensions will be visible in a transaction only if you dispatch or refresh your transaction.

Dash at the beginning of commands

A dash (-) indicates a comment. Lines beginning with a dash are ignored.

End-of-file character within the commands

Certain file concatenation operations can insert an end-of-file character within the command file. In this case, processing stops at this character.

TRANSLATING THE METAMODEL

You can translate the metamodel of a system repository into another language. MetaClasses, MetaAssociations, MetaAttributes and texts can have a different name depending on the language in which the user is working: this allows a project team to have a single repository for sites working in different languages.

Translating a repository involves the following operations:

- Defining the names in the new language, in a *command file*.
- Importing the command file.
- Translation by the translation utility.

Translating and Compiling Environments

You can translate the metamodel of an environment if several languages are defined in the metamodel. When the metamodel has been modified, it must be compiled. To do this, translate it in the current language.

☛ *For more information on translating and compiling the metamodel, see **MEGA Administration-Supervisor** guide, chapter "Managing Environments".*

RENAMING MEGA CONCEPTS

Some standards (e.g.: NAF, DoDAF, Archimate) use their own terminology. **MEGA** concepts can be renamed according to the context in which they are used.

> For example, the MetaClass called "Application System" in **MEGA Architecture** standard product is called "Application Collaboration" MetaClass in **Archimate** Terminology.

The renaming mechanism implemented in **MEGA** enables definition of different names carried by the same concept in its different contexts of use. Each user, depending on his/her profile and the context in which he/she is working, uses terminology with which he/she is familiar.

Functionalities proposed here are based on the **Terminology** notion.

The following points are covered here:

Defining a Terminology

A new context of use of **MEGA** concepts is defined by all the set of new terms to be used. These terms can be defined in several languages.

> 🕮 *A **Terminology** is a set of terms used in a specific context instead of the names used in basic configuration. This context can be a modeling standard (e.g.: Archimate, TOGAF), the vocabulary used in a specific field (e.g.: audit, internal control) or specific to the company.*
>
> ☛ *For more details on the list of concepts that you can rename, see "Renaming Concepts", page 79.*

Some terms can be specific to a population of users. You can connect a user profile to one or several Terminologies ordered by priority.

> ☛ *For more details regarding profiles connected to several Terminologies, see "Managing profiles associated with several Terminologies", page 77.*

Some standards define pictures associated with their concepts. The renaming mechanism implemented in **MEGA** enables association of MetaPictures with a terminology.

> ☛ *For more details on the use of MetaPictures, see "Defining a shape for the new MetaClass", page 98.*
>
> ☛ *For more details on how to associate MetaPictures with a terminology, see "Connecting MetaPictures of concepts", page 77.*

To define a Terminology:

1. Connect to **MEGA** with the **MEGA Customizer** profile.
2. Create a Terminology.

> ☛ *See "Creating a Terminology", page 74.*

3. Create all the languages you need for the Terminology and associate each of them with the Terminology.

   ☛ *See "Creating a language for a Terminology", page 75.*

4. Configure the Terminology:
   - specify the profiles associated with the Terminology

     ☛ *See "Specifying all the profiles associated with a Terminology", page 76.*

   - (optional) specify the concept MetaPictures for the Terminology

     ☛ *See "Connecting MetaPictures of concepts", page 77.*

5. From **MEGA Administration**, translate and compile the Metamodel (you do not need to compile the technical data).

   ☛ *For details on the translation and compilation, see **MEGA Administration - Supervisor** guide, chapter "Compiling an environment".*

6. Define the terms associated with your Terminology.

   ☛ *See "Renaming Concepts", page 79*

7. From **MEGA Administration**, translate and compile the Metamodel (Technical Data and Metamodel).

   ☛ *For details on the translation and compilation, see **MEGA Administration - Supervisor** guide, chapter Compiling an environment.*

Creating a Terminology

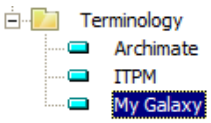
To create a Terminology:

1. Connect to **MEGA** with the **MEGA Customizer** profile.
2. Check that you are in Expert metamodel (**Tools > Options > Repository > Metamodel Access >** "Expert").
3. In the **MetaStudio** navigation window, right-click the **Terminology** folder and select **New > Terminology**.
4. Enter the terminology **Name**.

   ```
   For example: My Galaxy.
   ```

5. Click **OK**.
   The Terminology is created.

   ☛ *If you want you Terminology to inherits from another Terminology see "Inheriting a Terminology", page 75.*



Inheriting a Terminology

A Terminology can inherits from another one. Connected to the inherited Terminology, the inheriting Terminology inherits:

- languages

  ☛ *See "Creating a language for a Terminology", page 75.*

- MetaPictures

  ☛ *See "Connecting MetaPictures of concepts", page 77.*

- terms given to the different concepts in the different languages

  ☛ *See "Renaming Concepts", page 79.*

To connect an inherited Terminology:

1. Explore the Terminology you want to be inheriting from another Terminology.

   ☛ *For example explore "My Galaxy" Terminology.*

2. In the **Explore** tool, click **Empty Collections** .
3. Right-click the **Inherited Terminology** folder and select **Connect**.
4. In the **Terminology** list, select the Terminology you want your terminology to inherit from and click **Connect**.

   For example if you select "Archimate" Terminology, "My Galaxy" Terminology inherits languages, MetaPictures, and terms from "Archimate" Terminology.

Creating a language for a Terminology

To configure your Terminology you need to create the languages you want to be available for your Terminology and associate each language with the Terminology.

☛ *You can create as many languages as you need for your Terminology.*

To create a language for a Terminology:

1. In the **MetaStudio** navigation window, expand the **Language** folder.
2. Expand the language you want to be available for the Terminology.

   For example **English**.

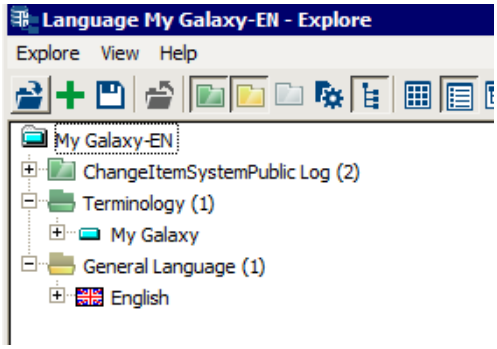
3. Right-click the **Specialized Language** folder and select **New > Specialized Language**.
   The **Creation of Language** dialog box opens.
4. In the **Name** field enter your language name.

   For example "My Galaxy-EN".

5. Right-click the language and select **Explore**.
6. In the **Explore** tool, click **Empty Collections** .
7. Right-click the **Terminology** folder and select **Connect**.

8. In the **Terminology** list, select your Terminology and click **Connect**.

   For example "My Galaxy".



Specifying all the profiles associated with a Terminology

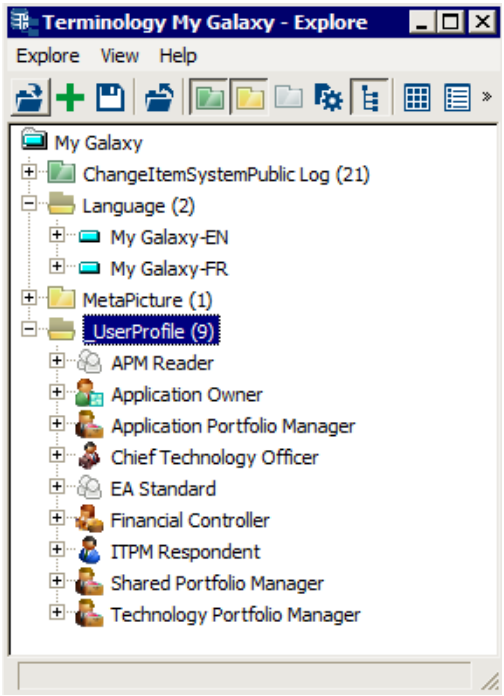
☛ *For more details on definition of users and profiles, see **MEGA Administration - Supervisor**, chapter "Managing Users".*

To associate profiles with a Terminology:

1. From the **MetaStudio** navigation window, in the **Terminology** folder right-click the Terminology and select **Explore**.
   An **Explore** window opens.
2. Right-click the **_UserProfile** folder and select **Connect**.

3. From the **Connecting** window, select all the profiles you want to associate with the Terminology and click **Connect**.
   All the selected profiles are associated with the Terminology.

   ☛ *To add a profile to a Terminology see also "Adding a Terminology to a profile", page 78*



Connecting MetaPictures of concepts

Prerequisite: diagrams specific to the context to which the terminology relates are described and MetaPictures exist.

☛ *For more details on the use of MetaPictures, see "Defining a shape for the new MetaClass", page 98.*

To connect a concept MetaPictures to a Terminology:

1. From the **MetaStudio** navigation window, in the **Terminology** folder right-click the Terminology and select **Explore**.
   An **Explore** window opens.
2. Right-click the **MetaPicture** folder and select **Connect**.
3. From the **Connecting** window, in the MetaPicture list, select all the MetaPictures you want to connect to the Terminology and click **Connect**.

Managing profiles associated with several Terminologies

☛ *For details on profiles see **MEGA Administration - Supervisor** guide, "Managing users" chapter.*

You can associate a profile with several terminologies.

☛ *See "Adding a Terminology to a profile", page 78.*

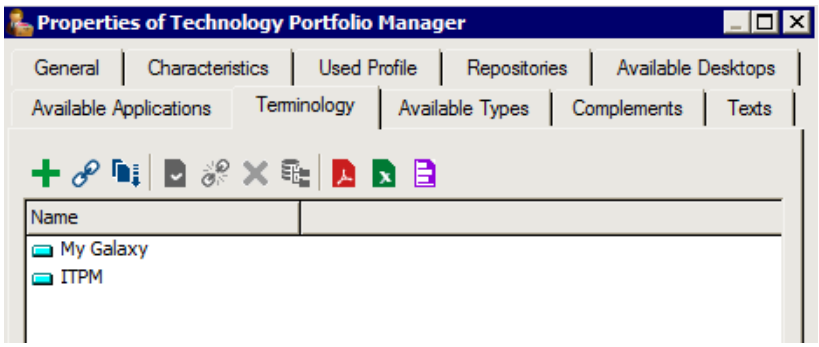
When a profile is associated with several Terminologies, you must order the Terminologies to define which of them must be displayed as priority over the other ones.

☛ *See "Defining the priority Terminology", page 78).*

Adding a Terminology to a profile

To associate a profile with a terminology:

1. Access the profile Properties.
2. Select the **Terminology** tab.
3. Click **Connect** to connect the terminology to the profile.

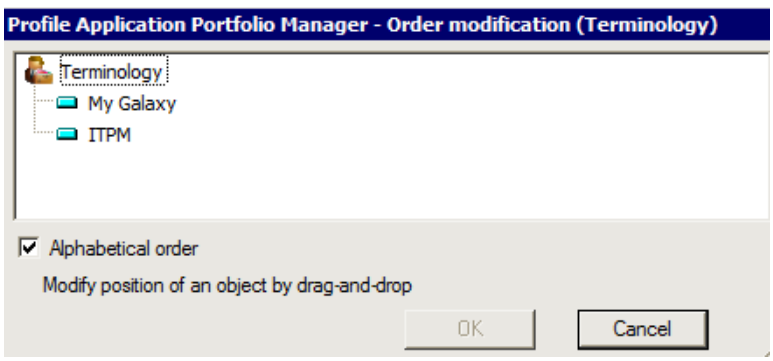


Defining the priority Terminology

By default Terminologies are listed in alphabetical order, so that the default Terminology displayed is the top level one.

To define the priority Terminology for a profile:

1. Access the profile properties.
2. Select the **Terminology** tab.
3. Click **Reorganize** to define the priority order of appearance of terms.



4. Drag and drop the Terminologies to order them with the priority Terminology at the top.

   ```
   If My Galaxy Terminology is above ITPM Terminology, ITPM
   terms are displayed when Galaxy terms have not been
   specified.
   ```

Renaming Concepts

For each of the **MEGA** concepts that can be renamed, you can redefine:

- the concept name

  ☛ *See "Changing translatable fields of a concept", page 80,*

- other translatable fields

  ☛ *See "Changing translatable fields of a concept", page 80.*

Changing a concept name for a specific Terminology language

☛ *For details on the list of concepts that can be renamed and the attribute that corresponds to the concept name, see "Concepts that can be renamed", page 81.*

Prerequisite: To use the new terminology, you must first translate and compile the Metamodel (you do not need to compile the technical data).

☛ *For more details on translate and compile, see **MEGA Administration - Supervisor** guide, chapter "Compiling an environment".*

To change a concept name:

1. Access the concept Properties.

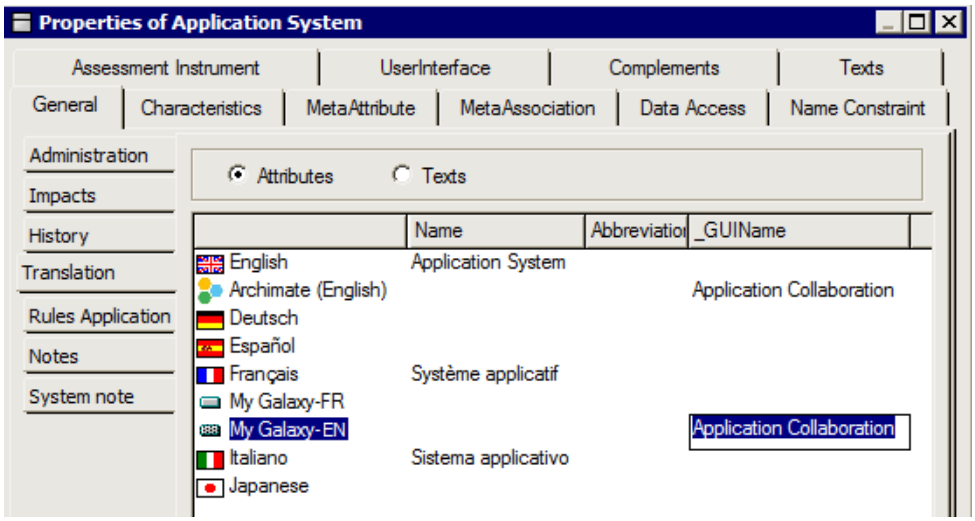
   ```
   E.g.: the Application System MetaClass.
   ```

2. In the **General** tab select **Translation** sub-tab.
3. Select **Attributes**.

4. In the language line, in the **_GuiName** field enter the concept name corresponding to the application context.

   For example you can modify the ***Application System*** Metaclass in the context of "My Galaxy" terminology for English.

   For "My Galaxy-EN" language, in the **_GUIName** field enter "Application Collaboration".



5. Click **OK** to finish.

   ☛ *For these modifications to be taken into account you need to compile the Metamodel.*

Changing translatable fields of a concept

The majority of concepts that can be renamed have a "Comment" field, which can be modified to correspond to a context-specific terminology.

Some concepts, such as "_Code Template" have other fields that can be renamed (e.g.: "Comment", "Translatable Code template").

To modify a Code Template field:

For example a Code Template in the context of Galaxy Terminology and for English language.

1. Access the _Code Template properties.
2. Select the **General > Translation** tab.
3. Select **Texts**.
4. Select the field you want to modify.

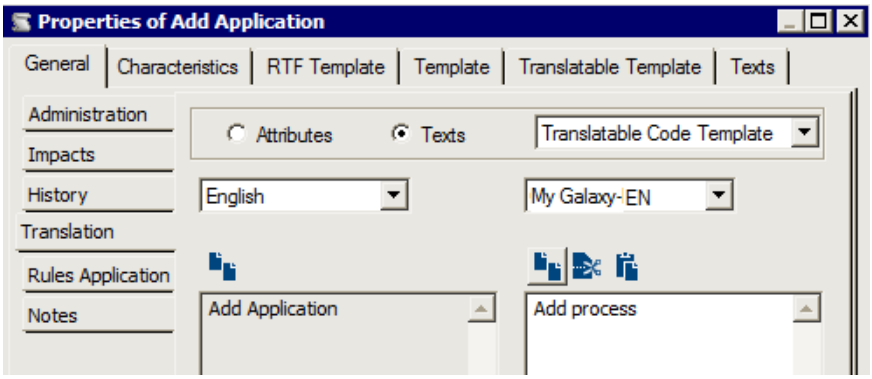
   For example: "Translatable Code template"

5. In the left language field, select the language from which you want to make the modification.

   For example: English.

6. In the right language field, select the language you want to specify.

   `For example: Galaxy-US.`



7. In the right pane, enter the modified string.
8. Click **OK**.

   ☛ *For these modifications to be taken into account you need to compile the Technical Data.*

Concepts that can be renamed

The following table lists the **MEGA** concepts that can be renamed. For each of these:

- the "Main attribute" column indicates the field in which the name of the concept for a terminology and a given language should be specified
- the "Secondary attribute" column lists the other fields that can be modified for a terminology and a given language.

Concept	Main attribute	Secondary attribute
_Code Template	Translatable Code Template	
Business Role	Abbreviation, External Value	Comment
Chapter		Comment
Desktop	_GUIName	
Desktop Container	_GUIName	
DiagramTypeView	DiagramTypeviewName	
DiagramTypeField	_GUIName	

Concept	Main attribute	Secondary attribute
MetaAssociation		Comment
MetaAssociationEnd	_GUIName	Comment
MetaAttribute	_GUIName	Comment
MetaAttributeGroup	_GUIName	Comment
MetaAttributeValue	ExternalValue	ExternalAbbreviation
metaclass	_GUIName	Comment
MetaCommand Group	_GUIName	
MetaCommand Item	_GUIName	
MetaCommand Manager	_GUIName	
MetaField	_GUIName	
MetaListType	_GUIName	
MetaPattern	_GUIName	
MetaPropertyPage	_GUIName	Comment
MetaTree	_GUIName	Comment
MetaTreeBranch	_GUIName	
MetaTreeNode	_GUIName	
MetaWizard	_GUIName	Comment
Modeling Rule	Diagnosis	RuleDescription
Modeling Regulation	_GUIName	Comment
Profile	_GUIName	Comment
Report Chapter	_GUIName	Comment
Report Parameter	_GUIName	Comment

Concept	Main attribute	Secondary attribute
Report Template	_GUIName	Comment
TaggedValue	_GUIName	Comment
Workflow Status	_GUIName	Comment
Workflow Transition	_GUIName	Comment

METAMODEL SYNTAX IN COMMAND FILES

This section contains examples of syntax enabling understanding of metamodel management operations contained in command files.

Creating MetaClasses

Syntax	.Create MetaClass "Object Type"
Example	.Create .MetaClass "Company Object Type"

For each MetaClass, you can specify name maximum length, systematic conversion and a list of prohibited characters.

Syntax	.Update .MetaClass "MetaClass" ._NameLength Length - ._NameCaseSensivity "Conversion Code" - ._NameFirstCharExclude "Chars To Exclude" - ._NameCurrentCharExclude "Chars To Exclude" - ._NameLastCharExclude "Chars To Exclude"
Example	.Modifier .MetaClass "Programme" ._NameLength 16 - ._NameCaseSensivity "U" - ._NameFirstCharExclude "&é(-è_çà)=^$ù*¤!:;,<>?}êîûôöïüÿ" - ._NameCurrentCharExclude "0123456789*?.:,+-"- ._NameLastCharExclude "&é(-è_çà)=^$ù*¤!:;,~#{[|`\^@]}êîûôöïüÿ"

Typing MetaClasses

Syntax	Connect .MetaClass "MetaClass" ."_Operator" "Operator" ._ScanInit - "Extraction condition" ._ScanType "Processing type"
Example 1	.Connect .MetaClass "Keyword" ."_Operator" "Extract" ._ScanInit "S"
Example 2	.Connect .MetaClass "Table" ."_Operator" "Extract" ._ScanType "D"

The first command indicates that all repository keywords will be extracted systematically.
The second indicates that when a table is extracted, all the minor objects describing it (columns, keys, indexes) are also extracted.

Creating MetaAssociations

In a command file, the major MetaAssociationEnd is the first MetaAssociationEnd indicated in the MetaAssociation creation command.

Syntax	.Create .MetaAssociation "NameMetaAssociation" .MetaClass "MetaClassMajor" "MetaAssociationEndMajor" N -"MetaClassMinor" "MetaAssociationEndMinor" N
Example 1	.Create .MetaAssociation "(OU/S)" .MetaClass "Org-Unit" "Org-Unit" N - "Keyword" "Keyword" N
Example 2	.Create .MetaAssociation "Org-Unit-Sending" .MetaClass "Org-Unit" "Org-Unit-Sending" N - "Message" "Message-Sent" N
Example 3	.Create .MetaAssociation "Org-Unit-Composition".MetaClass "Org-Unit" - "Composition" N "Org-Unit" "Component" N

Reversing MetaAssociation Orientation

MetaAssociation orientation can be reversed in a command file using the following syntax:

Syntax	.Update .MetaAssociationEnd <MetaClass Name> <MetaAssociationEnd Name> ._Major <Major>
Example 1	.Update .MetaAssociationEnd "Tool" "User-Application" ._Major "0"
Example 2	.Update .MetaAssociationEnd "Application" "User-Tool" ._Major "1"

Modifying Link Type

Link type can be modified in a command file using the following syntax:

Syntax	.Disconnect .MetaAssociation "LinkName" ."MetaAssociationType" "Old Link Type" .Connect .MetaAssociation "NameOfLink" ."MetaAssociationType" "New Link Type"
Example	.Disconnect .MetaAssociation "Org-Unit-Composition" ."MetaAssociationType" "Composition" .Connect .MetaAssociation "Org-Unit-Composition" ."MetaAssociationType" "Hierarchy"

Modifying Object Protection

Syntax	.Update .MetaAssociationEnd "Object Type" "MetaAssociationEnd" ._Protection "U"
Example 1	.Update .MetaAssociationEnd "Operation" "Org-Unit" ._Protection "U" .Modify .MetaAssociationEnd "Org-Unit" "Operation" ._Protection "A"
Example 2	.Update .MetaAssociationEnd "Operation" "Org-Unit" ._Protection "U" .Update .MetaAssociationEnd "Org-Unit" "Operation" ._Protection "U"

Example 1 shows reversal of protections.

Example 2 shows deactivation of protections.

Modifying Link Behavior for a Given Operator

Syntax	.Disconnect .MetaAssociation "LinkName" ."Operator" "Operator"."Minor to major behavior" "MajorProcessing" ."Major to minor behavior" "MinorProcessing"
Example	.Connect .MetaAssociation "Org-Unit-Sending" ."_Operator" "Extract" -."Minor to major behavior" "A" ."Major to minor behavior" "S"

With this command, all the messages sent by an org-unit are extracted, no matter what diagram they are in.

In the standard configuration, the message must be in the diagram to be extracted.

Creating Characteristics

You can also create a MetaAttribute by means of an MGE command file, using the following syntax:

Syntax	.Create .MetaAttribute "Characteristic" ."MetaAttribute Type" "X" ."MetaAttribute Length" "5"
Example 1	.Create .MetaAttribute "Message-Type" ."MetaAttribute Type" "X" ."MetaAttribute Length" "1"
Example 2	.Create .MetaAttribute "Value" ."MetaAttribute Type" "A"
Example 3	.Create .MetaAttribute "ShortName" ."MetaAttribute Type" "X " ."MetaAttribute Length" "8" ."_AtIndex" "U"

Characteristic values may be declared unique for a given object type. To do this, specify the value "U" for _AtIndex in the characteristic definition.

☛ *The characteristic uniqueness is only effective in repositories created after uniqueness has been declared.*

Creating a text type characteristic

Syntax	.Create .MetaAttribute "Characteristic Name" ."MetaAttribute Type" "A"
Example	.Create .MetaAttribute "Ext Objective" .MetaAttribute Type "A"

Creating a tabulated MetaAttribute

Syntax	.Create .MetaAttribute "Characteristic Name" ."MetaAttribute Type" "Format" - ."Update Version" "16643" - .MetaAttribute Length "Length" ."MetaAttribute Format" "T"
Example	Create .MetaAttribute "Ext Role" ."MetaAttribute Type" "X" - "Update Version" "16643" - ."MetaAttribute Length" "20" ."MetaAttribute Format" "T"

Creating a unique index MetaAttribute

Syntax	.Create .MetaAttribute "Characteristic Name" ."MetaAttribute Type" "Format" - .MetaAttribute Length "Length" ._AtIndex "U"
Example	.Create .MetaAttribute "Ext Code" ."MetaAttribute Type" "X" ."MetaAttribute Length" "5" ."_AtIndex" "U"

Connecting a MetaAttribute to a MetaClass

Syntax	.Create MetaClass "Object Type"
Example 1	.Connect .MetaClass "Procedure" ."MetaAttribute" "Procedure-Type"
Example 2	.Connect .MetaClass "Procedure" ."MetaAttribute" "Object"

Example 2 illustrates attachment of a text to an object type.

☛ *The "Comment" text is linked to all object types when they are created. This command is used to connect additional text to a segment.*

Connecting a MetaAttribute to a MetaAssociationEnd

Syntax	.Connect .MetaAssociation "Link" ."MetaAttribute" "Characteristic"
Example	.Connect .MetaAssociation "Operation-Sending" ."MetaAttribute" "Predicate"

Creating an abbreviation

Syntax	.Abbreviate .MetaClass "Long Name" "Short Name"
Example	.Abbreviate .MetaAttribute "Creation Date" "Creation"

CREATING CONSISTENCY CHECKS

Repository build is subject to modeling rules. Depending on the product or products you have, **MEGA** provides a certain number of rules you can apply to objects you create so as to check their consistency. You can also create new rules.

Each user can run a check on an object. As for rule or regulation modification functions, these require the **MEGA Supervisor** or **MEGA Studio** technical modules.

REMINDER: RULES OPERATION PRINCIPLE

Rules apply to **MEGA** repository objects. They define checks on these objects and are implemented by:

- tests
- macros.

Rules provided by default are accessible in the **MetaStudio** navigation window, available only with the **MEGA Supervisor** and **MEGA Studio** technical modules.

You can view rules as a function of the regulations or objects to which they apply.

Each regulation enables classification of rules according to a specific context or domain.

You can apply:

- a regulation or a set of regulations by default.
- a regulation to repository objects as and when required.

MEGA presents a list of regulations that varies depending on products you have available. In **MEGA Process**, you have for example an organizational type regulation that you can apply by default to check correct sequencing of objects you create.

Displaying check results

Check results are displayed:

- in the properties dialog boxes of objects concerned by the rules currently applied.
- in the diagram drawings that you wish to check. For more information, see the **MEGA Common Features** guide.

Rules and check descriptors

Use of rules does not replace the check descriptor mechanism.

☛ *For more details about this mechanism, see "Checking Descriptor Validity", page 75.*

Both tools coexist and complement each other.

In fact, in a check descriptor it is possible to request insertion of the rules application report of a regulation on the current object.

The check descriptor:

- determines the set of objects to be included in the check
- defines global formatting of the report
- then for each object browsed, inserts the report or reports on rules application of one or several regulations.

We therefore separate repository browsing (which concern the check descriptor) from the definition of modeling rules.

REGULATIONS

A regulation enables classification of rules according to a specific context or domain. It can contain not only a set of rules but also sub-regulations.

Displaying regulations

To view regulations provided in **MEGA**:

1. Select the **MetaStudio** navigation window.

   ☛ *This window is available only with the **MEGA Supervisor** technical module.*
2. Successively expand the "Repository Consistency" and "Modeling Regulation" folders.

The list of available regulations appears.

Activating a regulation

To activate a regulation:

1. In **MEGA** menu bar, select **Tools** > **Options**.
2. In the dialog box that appears, select **Modeling and Methods Regulations**.
3. In the **Active modeling regulation** field, click ....
4. In the dialog box that opens, select the regulation to be applied and click **OK**.

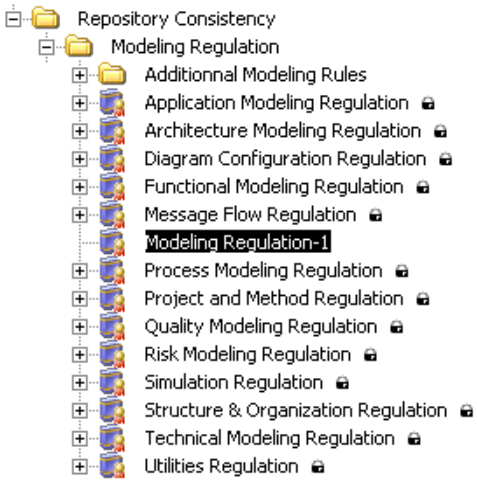
If you have applied several regulations, you must assemble these under a single complex regulation.

Creating a complex regulation

To create a complex regulation:

1. In the **MetaStudio** navigation window, expand the "Repository Consistency" folder.
2. Right-click the **Modeling Regulation** folder and select **New** > **Modeling Regulation**.
3. Name the new regulation and click **OK**.
   This then appears in the navigator.

4. Drag-and-drop the desired regulations onto this regulation.



5. To take this new regulation into account, select it in the options dialog box as indicated above.

Defining rules of a regulation

Rules provided by default are accessible in the **MetaStudio** navigation window. You can view them as a function of the regulations or objects to which they apply. You can also display all existing rules. They are preceded by the following icon: .

When you run a check on an object, you do this via a regulation. So that a rule can be taken into account in the check, it must be connected to the applied regulation.

To connect a rule to a regulation:

1. Expand the **Repository Consistency** folder.
2. Find the desired rule in the set of rules or from the objects to which it applies.
3. Select the rule and drag it onto the regulation to which you wish to connect it.

Regulation application scope

Regulation application scope corresponds to the MetaClasses concerned by the regulation. Defining MetaClasses for each regulation simplifies selection of regulations to be applied on an object. In fact, when you run a check on an object, only the list of regulations that concern the object will appear.

You can extend application scope of a regulation.

```
You can for example connect the "Organizational Process"
MetaClass to a regulation concerning operations so that
this regulation can apply to operations contained in an
organizational process.
```

To add a MetaClass to regulation application scope :

1. In the **MetaStudio** navigation window, select the regulation.

2. Open its properties.
3. Select the **Scope of Application** tab.
   The first frame lists the MetaClasses concerned.
4. In the second frame, click the **Connect** button  .
   The Query dialog box appears.
5. Select the MetaClass to be connected, for example "Organizational Process" and click **OK**.

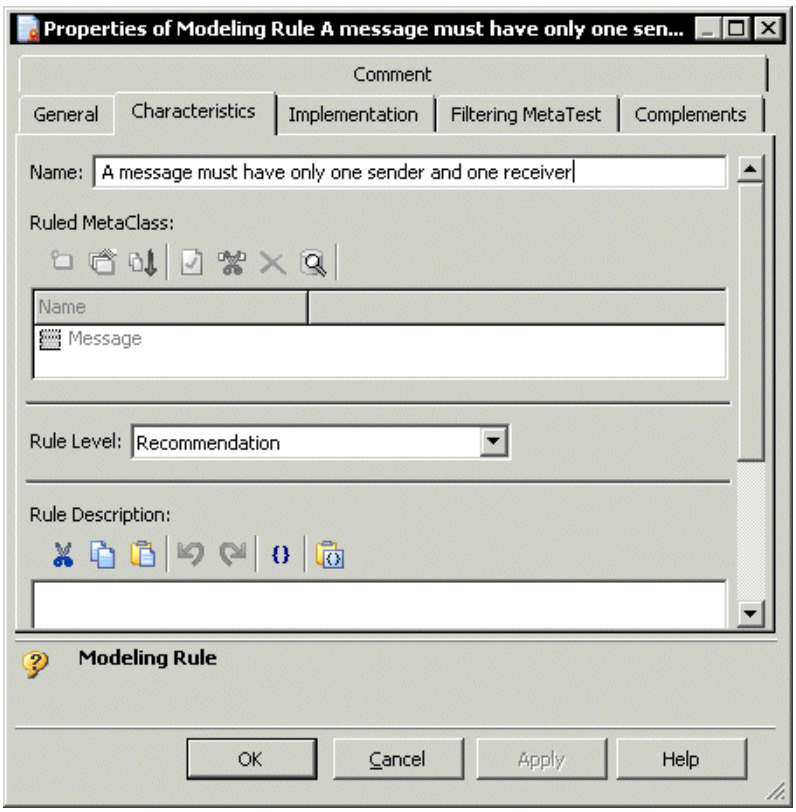
RULE PROPERTIES

To display rule properties:

1. In **MEGA**, select the **MetaStudio** navigation window.

   ☛ *This window appears only if you have the **MEGA Supervisor** technical module.*

2. In the "Repository Consistency" folder, right-click the rule and select **Properties**.
3. Select the **Characteristics** tab.
   The dialog box that appears indicates:
   - the name of the rule
   - the MetaClass(es) checked by this rule.



Rule level

This box defines importance of the rule. The possible options are:

- **Suggestion**: rule it is advisable to observe, but not a requirement. Non-respect of the rule is indicated in the rules application report but does not produce a warning.
- **Recommendation** important rule. Non-respect of the rule produces a warning.
- **Requirement**: obligatory rule. Non-respect of this rule produces an error and is a source of blocking.

The properties dialog box also displays the test or the macro used for implementing the rule. See "Rule Application Scope", page 192.

RULE APPLICATION SCOPE

A rule is applied to a MetaClass. By default, it can be applied to all objects of this MetaClass. However, it is sometimes necessary to define the scope of application of a rule more precisely.

Restricting application scope

Consider the example of the following rule:

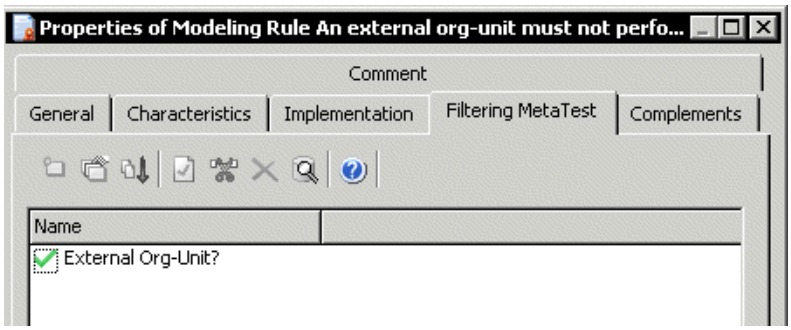
```
"An external org-unit cannot execute an operation".
```

This rule is only meaningful for external org-units. It is therefore appropriate to restrict application of this rule on the org-unit MetaClass, so that it executes for external org-units only.

To restrict scope of application of the rule:

- Create a test filtering the scope of application of the rule.

```
For example, in the rule you will define a filter so that
the rule applies not to all org-units but only to external
org-units.
```



If the filtering test is true, in other words if the org-unit is external, the implementing test executes.

Several filtering tests can be used. In this case, the rule applies if all filtering tests are true.

☛ *See "Defining an Implementation Test", page 198.*

Extending application scope

Certain highly specific rules can apply to several MetaClasses.

Consider the example of rule:

```
"Important objects must have a comment".
```

```
MetaClasses considered as "important" are business process,
org-unit, organizational process, application, etc. For
each of these MetaClasses, the condition applied is: "The
comment should not be empty".
```

Creating a rule for several MetaClasses is of interest only if the condition applied is valid whatever the MetaClass. In our example, the condition refers to the comment which is a property of all MetaClasses, and consequently of all those concerned by the rule.

Conditioning application of a rule

In the modeling regulation around the business process, two rules have been defined for the "Operation" MetaClass:

- An operation must be performed by an org-unit
- An operation must be performed by only one org-unit

For pedagogical reasons, these two constraints have not been grouped in a single constraint (an operation must be performed by an org-unit and by only one org-unit).

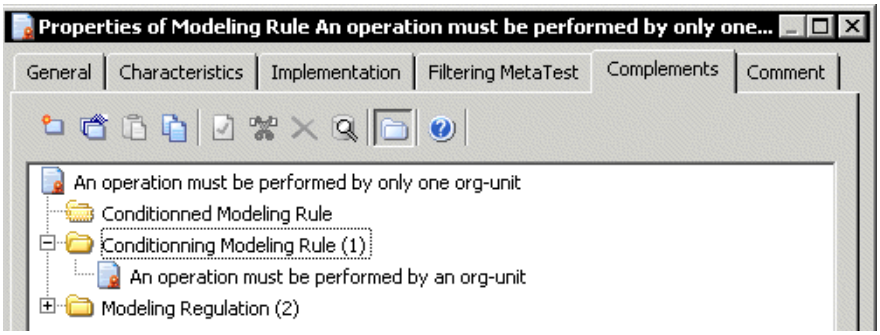
However, it is obvious that if the first rule is not verified, the second is of no interest.

This is why application of the second rule depends on verification of the first. In other words, the second rule is only applied only if the first is verified.

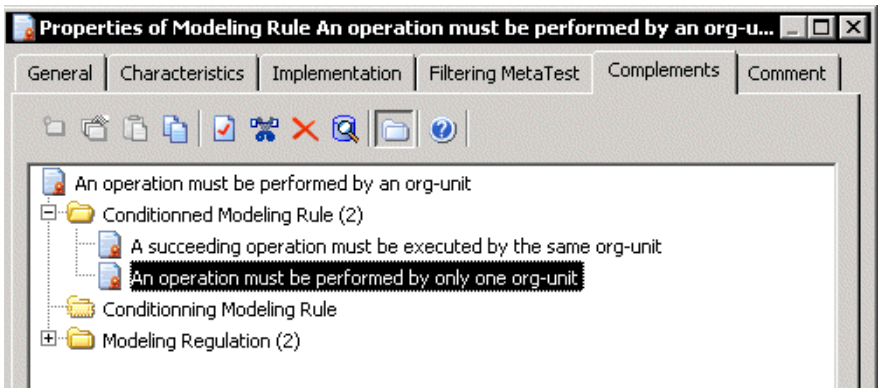
1. Open the dialog boxes of these two rules in turn and select the **Complements** tab.

You will see that:

- The rule "An operation must be performed by only one org-unit" is conditioned by the rule "An operation must be performed by an org-unit".



- The rule "An operation must be performed by an org-unit" conditions the rule "An operation must be performed by only one org-unit".



RULE IMPLEMENTATION

A rule can be implemented:

- by a test
- by a macro

> ☛ *The use of implementation tests is recommended, test by macros generally being used in complex cases only.*

While rules can be used by all users, creation of an implementation test requires advanced knowledge and assumes that you are in "Advanced" access mode.

To pass to "Advanced" access mode:

1. In **MEGA** menu bar, select **Tools** > **Options**.
2. In the dialog box that opens, select **Repository**.
3. In the **Metamodel Access** option, select "Advanced" mode.

Implementing a Rule by a Test or Tests

A test expresses a condition in the form of an expression.

Implementing a modeling rule using a test consists of connecting this test to the rule. For a given object, the modeling rule is verified if application of the test that implements it returns the value "true".

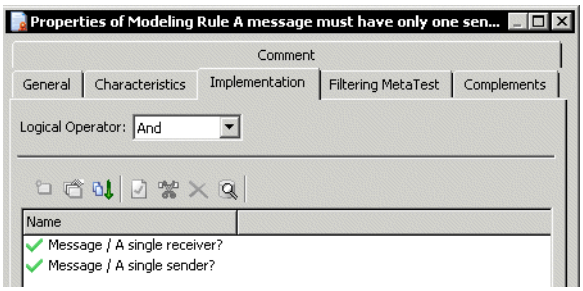
You can connect several tests to a rule. In this case, the **Logical Operator** offers two possible options:

- **And**: the rule is verified only if all tests are positive.
- **Or**: a single test is sufficient to verify the rule.

For example, the rule "A message must have a sender and a receiver" is implemented by two tests :

- One test to verify origin of the message
- One test to verify destination of the message

The logical operator of the test is "And", which indicates that both tests must be positive for the rule to be verified.



A test can also be used in several rules.

☛ *To create an implementation test, see "Defining an Implementation Test", page 198.*

Implementing a Rule by a Macro

The macro is written in VBScript and uses **MEGA** APIs.

To define a macro implementing a rule:

1. Right-click the rule.
2. Select **New** > **Implementing Macro**.

A macro is created and functions to be implemented are automatically declared.
The following is the detail of these functions:

```
Sub RuleAppliableIs (oToBeRuled as MegaObject, oRule as
MegaObject, sParameter as String, bRuleAppliableIs as
Boolean)

Sub RuleApply (oToBeRuled as MegaObject, oRule as
MegaObject, sParameter as String, bRuleResult as Boolean)
```

☛ *"oRule" is the modeling rule implemented.*

- The RuleAppliableIs function returns "true" in the "bRuleAppliableIs" variable if "oToBeRuled" is a root project. If not, it returns "false".
- The RuleApply function returns "true" in the "bRuleResult" variable if "oToBeRuled" is connected to a project type.
- The "RuleAppliableIs" function is optional. If it is not defined, the rule is implicitly applicable to all instances of the MetaClass (which is the case for the majority of rules).

Another function can be implemented if we wish to control the content of text inserted in the detailed report when the rule is not verified:

```
Sub RuleWithReportApply (oToBeRuled, oRule, sParameters,
bRuleResult, sErrorReport)
```

This function is called on building the detailed report. The last parameter enables sending of a precise error report adapted to the processing carried out.

Implementing Test or Macro?

Consider the example of rule:

```
"A root project has no project type".
```

This rule is applied to the "Project" MetaClass, and more particularly to root projects.
To implement this rule, you must:

- firstly reduce the application scope of the rule.
- carry out the test "This project is not connected to a project type".

Using tests

If we use expression tests to define this rule, we must create two expression tests.

- One to test if the project is a root project, enabling filtering of the rule application scope.
- Another to test if the project is connected to a project type serving to implement the rule.

Using macros

If we use a macro, we have to implement two functions:

- RuleAppliableIs => this function enables restriction of rule application scope.
- RuleApply => this function enables definition of rule implementation.

> **If a rule is implemented by a macro, tests and filters are ignored. The macro code must manage implementation tests and filters on the application area.**

DEFINING AN IMPLEMENTATION TEST

A test can be implemented by an expression or a macro. The use of expressions is recommended, test by macro generally being used in complex cases only.

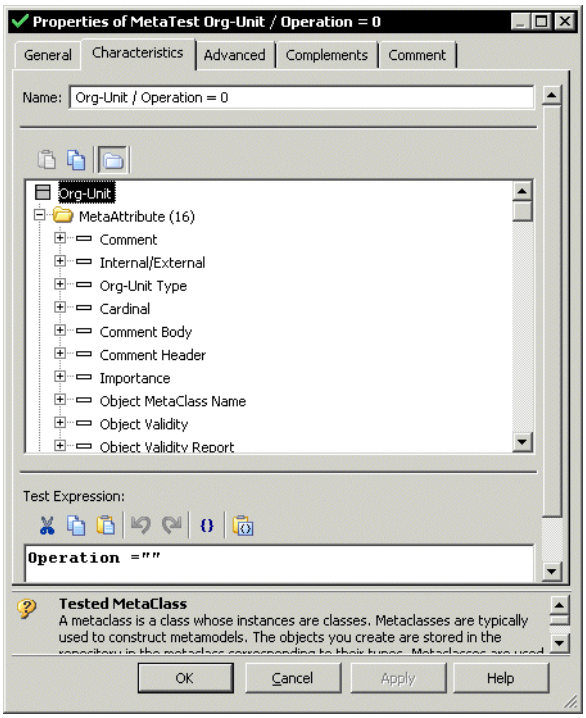
While rules can be used by all users, creation of an implementation test requires advanced knowledge and assumes that you are in "Expert" access mode.

To pass to "Expert" access mode:

1. In **MEGA** menu bar, select **Tools** > **Options**.
2. In the dialog box that opens, select **Repository**.
3. In the **Metamodel access** option, select "Expert" mode.

Defining an Implementation Test By an Expression

In the test properties dialog box, the **Characteristics** tab presents all MetaAssociationEnds and MetaAttributes of the tested MetaClass.



By drag-and-drop, you can place in the test expression the repository information to be tested.

☛ *Test expression syntax is the same as that used to define conditions in properties dialog box configuration. For more details, see the technical article on properties dialog boxes.*

You can:

- describe a test by expressions and by the functions described below.
- taking account of inheritance in tests (see "Taking account of inheritance in tests", page 202)

Expression and logical operator

The expression of a test comprises expressions. These expressions can be logically combined using brackets " () " as well as logical operators.

Logical operators available are:

```
And and Xor
```

Xor is the exclusive "Or", that is "true when one and only one of the combined expressions is true".

It is also possible to use **not** to obtain the negative of an expression

Example:

(Expression1 **and** Expression2) or **not** Expression3

Comparison attribute and operator

The expression of a test can be defined using various functions and comparison operators.

Comparison operators are:

```
= <> > < >= <=
```

These comparison operators can be used to define an expression from the value of one of the MetaAttributes of the tested MetaClass.

```
Attribute operator Value
```

Example

For an org-unit, the expression:

```
Internal/External = "X"
```

Returns "true" if the **Internal/External** attribute is "X" and false if not.

Reminder: here we use a tabulated value attribute and we therefore test its internal value).

For an operation, the expression:

```
Duration >= 40
```

Returns "true" if the **Duration** attribute is greater than or equal to 40.

> ☛ *Operators >, <, <=, >= are of course only meaningful for attributes of a type enabling such comparison.*

Warning: By proceeding in this way, we can test only attributes of the tested MetaClass. To be able to handle the attribute of a link from the MetaClass, the TrueForEach() and TrueForOne() functions subsequently presented should be used.

Function ItemCount() and comparison operator

Comparison operators can also be used to define an expression from the ItemCount function.

This function returns the number of objects found at the end of the MetaAssociationEnd or returned by the selector:

```
ItemCount(LegSel)
```

Where **LegSel** is a field representing a MetaAssociationEnd or a query.

Example

For an org-unit, the expression:

```
ItemCount(Operation)>2
```

Returns "true" if the org-unit is connected to strictly more than 2 operations.

It is possible to use a shortcut to test if at least one object is connected by a MetaAssociationEnd.

Therefore, for an org-unit, the expression:

```
Operation=""
```

Returns "true" if no operation is connected.

```
Operation<>""
```

Returns "true" if at least one operation is connected.

> 🕮 *Use of a query in the ItemCount() function of course requires that the query starts from the currently tested object, or that this does not require an input object.*

Functions TrueForEach() and TrueForOne()

These functions enable definition of an expression applying to an object or objects at the end of a MetaAssociationEnd or of a query.

TrueForEach returns "true" if the expression is true for all objects found at the end of the MetaAssociationEnd or returned by the query.

If no object is found or returned, this function returns "true".

TrueForEach returns "true" if the expression is true for at least one of the objects found at the end of the MetaAssociationEnd or returned by the query.

If no object is found or returned, this function returns "true".

```
TrueForEach(LegSel, Expression)
```

```
TrueForOne(LegSel, Expression)
```

Where **LegSel** is a field representing a MetaAssociationEnd or a query.

Example

For an org-unit, the expression:

```
TrueForEach (Operation , Duration >= 40)
```

Returns "true" if the org-unit is connected only to operations of duration strictly greater than 40, or if the org-unit is not connected to any operation.

For an organizational process, the expression:

```
TrueForOne (Org-Unit, RACI = "R" or RACI = "E")
```

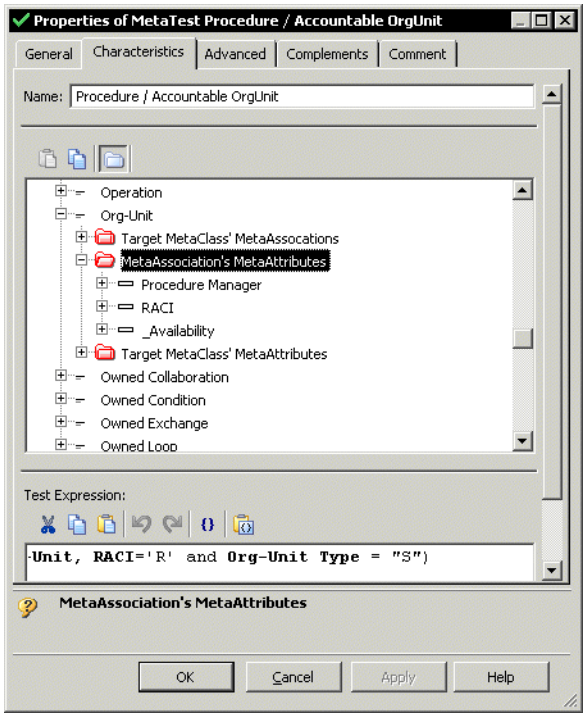
Returns "true" if the organizational process is connected to at least one org-unit with the RACI link attribute internal value "R" or "E".

Note here that when we use the TrueForEach and TrueForOne functions with a MetaAssociationEnd, it is possible to use in the expression :

- The link attributes
  and/or
- The attributes of the object at the end of the link

In the expression used by a TrueForEach or TrueForOne function, it is of course possible to use one of these two functions again so as to take a new MetaAssociationEnd or to trigger a new query.

In the implementation test properties dialog box, the tree presenting the MetaAssociationEnds and MetaAttributes enables navigation in depth of the metamodel. It is therefore possible to drag-and-drop one of the attributes of an object at the end of a MetaAssociationEnd.



Other available functions

```
ItemExist(LegSel, Object)
```

Where **LegSel** is a field representing a MetaAssociationEnd or a query.

This function returns "true" if the specified object is connected via the identified MetaAssociationEnd or is present in the set of objects returned by the query.

```
Available(Object)
```

This function returns "true" if the specified object exists.

Taking account of inheritance in tests

To take account of inheritance in tests on a MetaAssociation, you can suffix the field by:

- @INHERITING: displays objects that are inheriting
- @INHERITED: displays inherited objects
- @INHERITANCE: @INHERITING + @INHERITED

Example

From a MetaTest that relates to the Application MetaClass, you can define the following test:

```
ItemCount(~msUikEB5iGM3[Component]@INHERITED) > 2
```

Tests if the number of component applications of the tested application is more than 2 (including inherited component applications)

☛ *You can also use inheritance with ItemExist.*

Defining an Implementation Test By a Macro

The macro is written in VBScript and uses **MEGA** APIs.

To define a macro in a test:

1. Open the properties dialog box of the test.
2. Select the **Advanced** tab.
3. Connect the macro to the test.

This macro should show the function:

```
Sub TestApply(oToBeTested as MegaObject, oMetaTest as
MegaObject, sParameter as string, bTestResult as boolean)
```

or:

- "oToBeTested" is the **MEGA** object.
- oMetaTest is the test implemented.
- "sParameter" is the value of the chain attribute of the link between the test and the macro.
- "bRuleResult" is the result of application of the test.

☛ *"sParameter" is a parameter supplied to the macro. It is the value of the "Macro Parameter" attribute, which can be specified on the link between test and macro. This parameter can enable definition of generic macros, used to implement several tests.*

How to make a concept variable

1 Summary

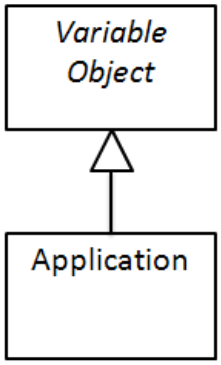
This document is a technical article for people wanting to extend the MEGA repository capability. Technical knowledge about the MEGA metamodel is a prerequisite.

The document details how to make a MEGA concept variable. A variable concept allows the instances to be varied. That means new instances can be created from the previous one inheriting some of the existing connections. Furthermore, inherited objects can be removed and or replaced.

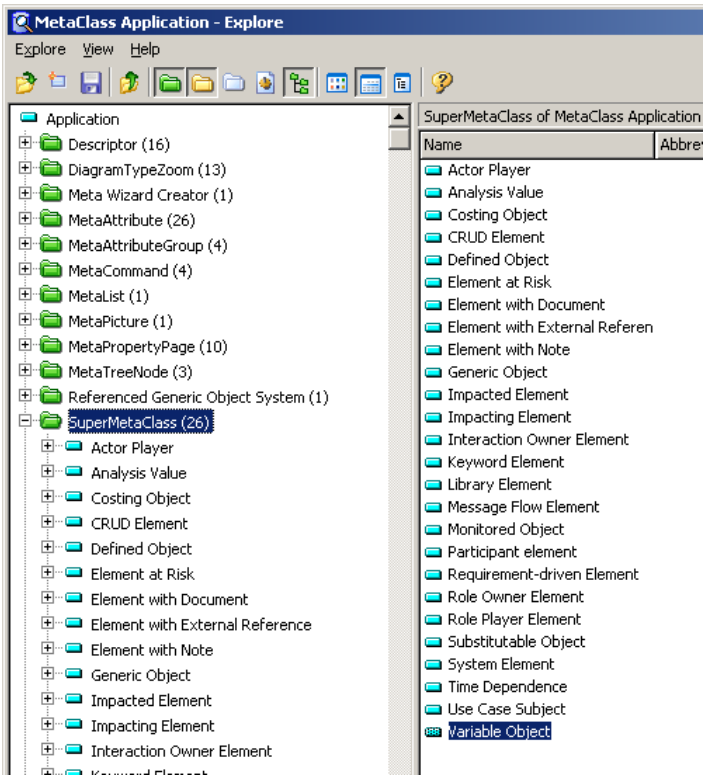
2 Variability

The first step to make an object variable is to inherit from the *Variable Object* Metaclass. The *Variable Object* metaclass is an abstract metaclass; it is not possible to create an instance from it. But, any concrete metaclasses inheriting from it becomes variable.

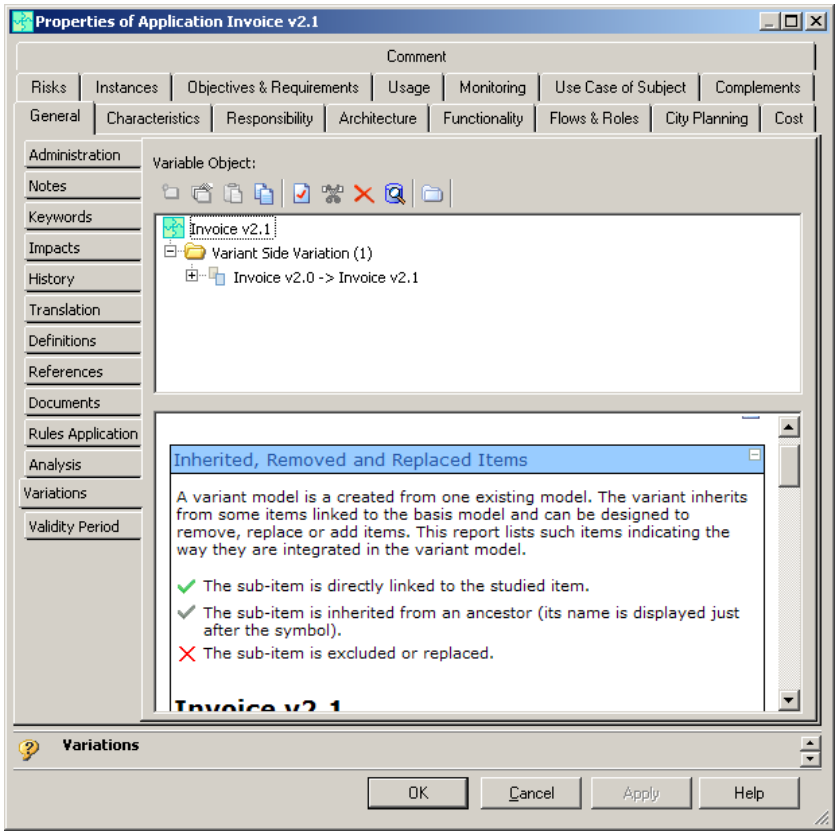
For example, the *Application* metaclass inherits from *Variable Object*.



The inheritance link is performed either tracing the link in a metamodel diagram or adding the *Variable Object* metaclass in the superMetaClass relationship of the concerned concept.

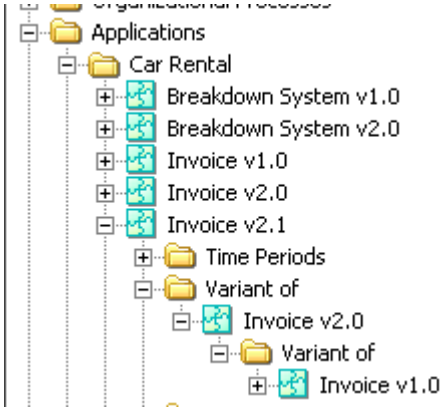


Inheriting from *Variable Object* will add a *General/Variations* tab in the property pages of the extended concept. This tab shows information related to the variation lineage.



Furthermore, a *New / Variant* command is also added in the popup menu of the variable concept so that variant can be created.

For varied object, a *Variant Of* folder will be displayed in the navigation trees showing the variable concept instances.

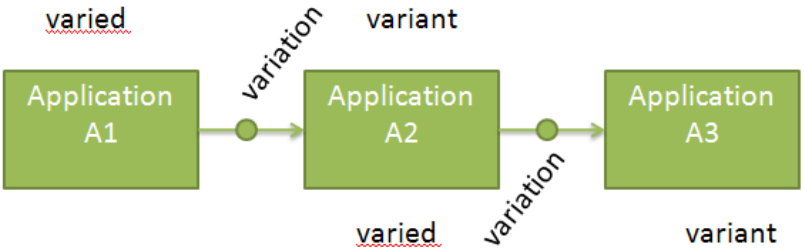


3 Inheritance

The variation mechanism is used to inherit from varied objects. To understand the inheritance we first define the vocabulary:

- **Variation**: the mechanism allowing varying instances. This mechanism is embodied by a MEGA object of type *Variation*.
- **Variant**: any instance created from another existing instance.
- **Varied**: any instance used to create a variant.

Based on those definitions, a variant object of A1 can be a varied of A3.

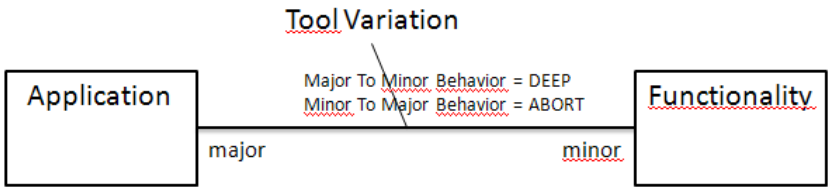


3.1 Inheritance Setting

To be inherited a sub-object must be connected to an inheriting association. Such association is set thanks to the *Tool Variation* operator. This operator is linked to the selected association and the following values must be set:

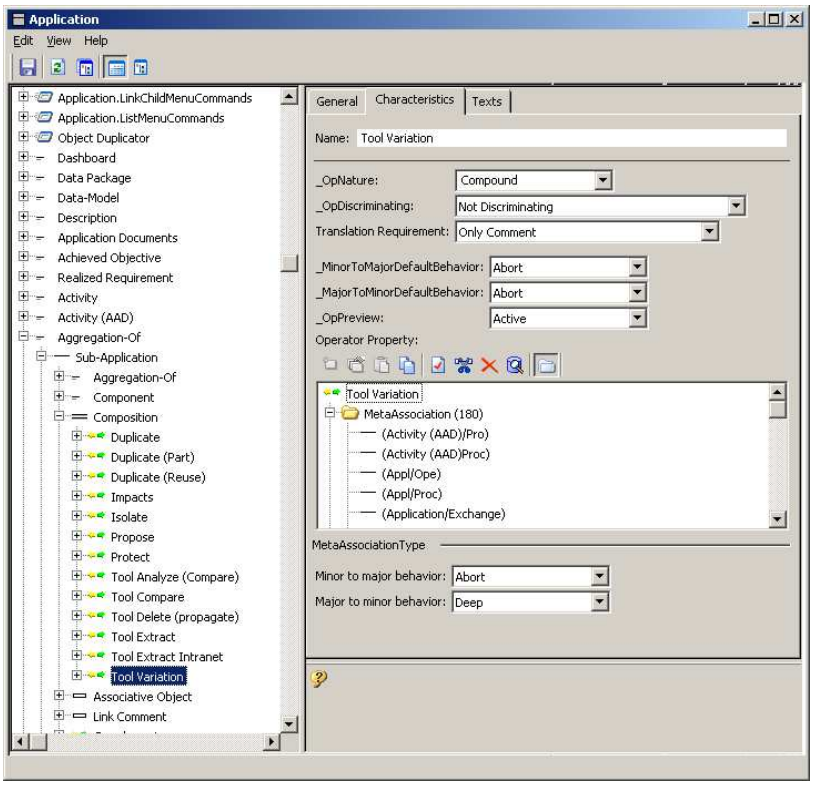
- ***Minor to Major Behavior***
  - o **Abort**: there is no inheritance if the sub-object is a major object (yellow folder in the MEGA explorer)
  - o **Deep**: the inheritance is applied if the sub-object is a major object (yellow folder)
- ***Major to Minor Behavior***

- **Abort**: there is no inheritance if the sub-object is a minor object (green folder)
- **Deep**: the inheritance is applied if the sub-object is a minor object (green folder)



The following figure shows this operator set for the *Composition* metaassociationtype. In that case, all associations linked to this type behaves the way defined for the operator else the operator is directly set in the association.

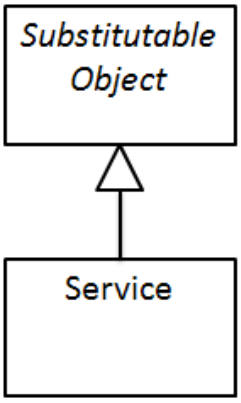
For this example, a component application is inherited but a aggregated application is not inherited.



3.2 Substitution

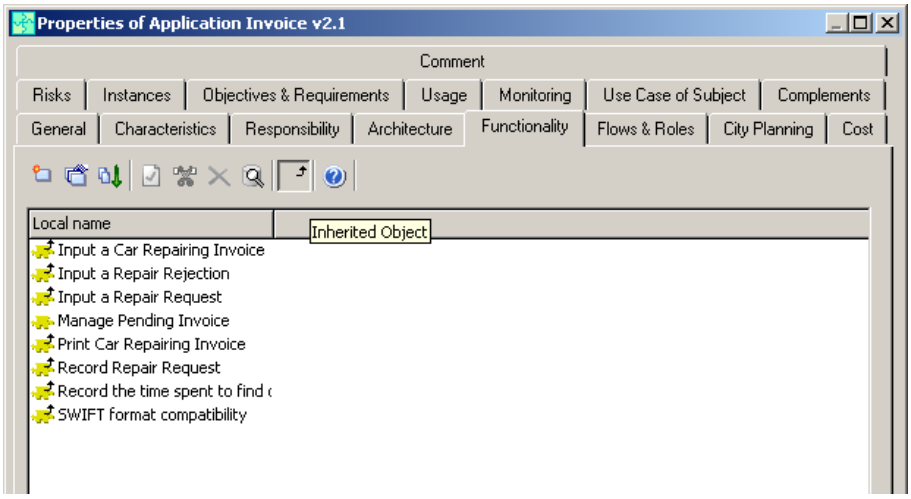
To be removed or replaced a sub-object must be inheritable but also substitutable. The inheritance allows only to grab the sub-object of the varied instance. It does not allow to replace or remove. To do that, the sub-object type must be declared as substitutable. This is done inheriting from the *Substitutable Object* metaclass.

In the following example, the IT Service concept is declared as substitutable. Then, all services inherited in an application can be removed or replaced.

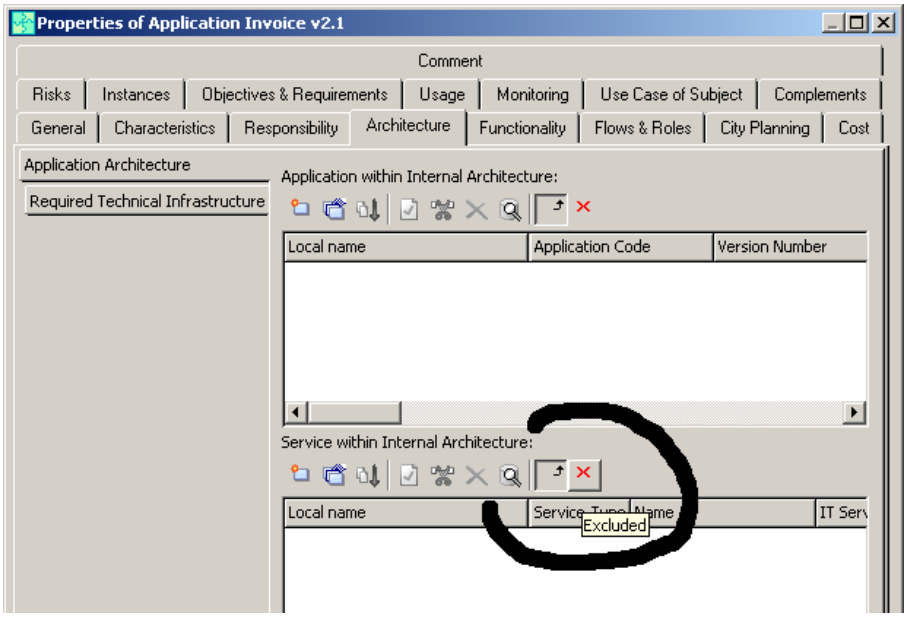


When the replacement and removal is available, all listview of the property pages showing inherited elements allows to perform the removal and the replacement.

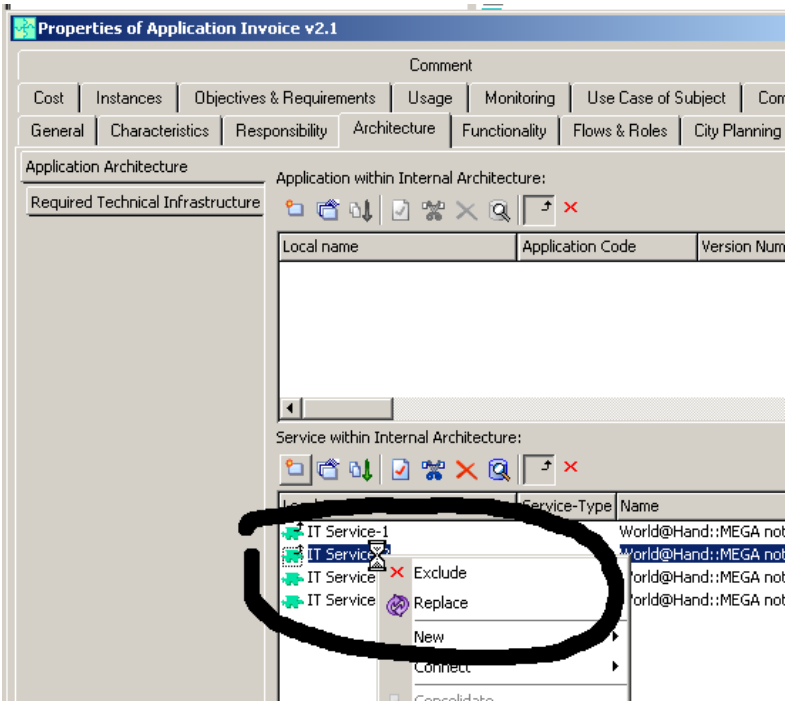
For example, the *Functionality* concept is not substitutable. The property page of an application shows only the inheritance (small black arrow).



The Service concept is substitutable, there is an additional button in the listview of the application property page. The red cross allows to show/hide excluded services.



To remove/replace, the popup menu shows two dedicated commands. The Replace command is available only if there is some objects to be replaced AND some other to do the replacement. The last one must be defined in the variant object and not be inherited.



Perimeters

This document explains how the "perimeter" and "MetaTool" concepts function in MEGA. A "perimeter" enables building a set of objects and links from a root object. A "Metatool" uses a perimeter in the context of a precise functionality. The "MEGA Supervisor" module is required to configure a perimeter.

Principle

In certain processing, you need to have a set of objects and links around the object that interests you, and this object will be referred to as the "root object". Depending on the context, description of this set can vary by taking into account certain objects or not, certain link types or not.

To solve this problem, the perimeter object has been installed. At the technical level, this is an occurrence of the "Operator" MetaClass, of which the "_OpNature" MetaAttribute is positioned on "Compound".

Configuration

A perimeter groups all the configuration necessary for building the set, and it is important not to confuse the perimeter and the set. Configuration is a behavior specified on the links. Possible behaviors are the following:

- **Deep:** The link and the object are added to the set, then processing continues from this object by propagation.
- **Standard**: The link and the object are added to the set, processing of this branch terminates, and propagation stops here.
- **Link:**The link is added to the set, but not the object. Processing stops at this level.
- **Abort:** Neither link nor object is added, and propagation stops.
- **Computed:**The nature of the link is not sufficient to define behavior on this particular link, and behavior is computed according to context by a macro (for implementation see chapter "Configuring perimeter links").

Defining a MetaTool

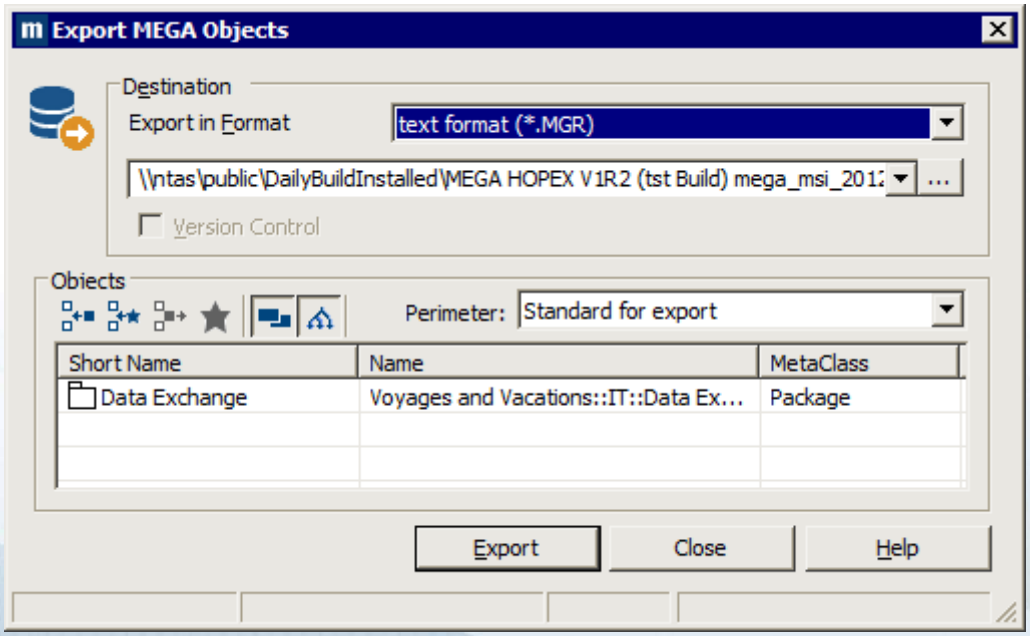
A MetaTool is a functionality which at processing needs to create a set from a root object. In practice, it is not a root object but a set of root objects, but the principle remains the same.
To build this set, the MetaTool applies a perimeter to the root object. It is possible to assign a default perimeter to each MetaTool. The aim of this perimeter is to provide a standard behavior if no customized perimeter exists in this context.

"Export" MetaTool

The "Export" MetaTool enables generation of an extraction file which represents part of the repository. A perimeter is required to build the set of objects and links to be extracted from one or several root objects.

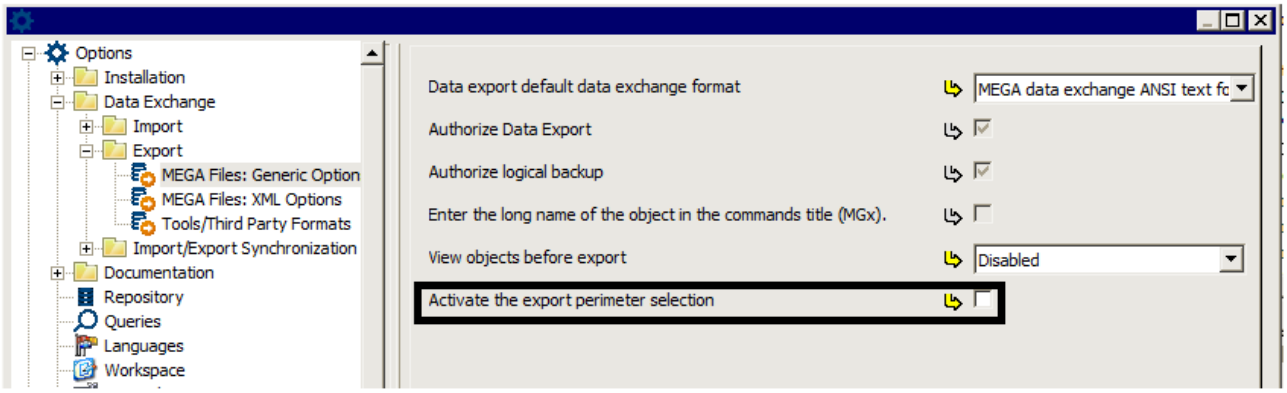
This functionality is available from the object menu by selecting command "Manage\Export", or from the desktop menu by selecting "File\Export\MEGA Objects".

The Export MetaTool dialog box looks like this:



By default, a perimeter is already present: "Standard for export", which is the MEGA standard perimeter of the export MetaTool.

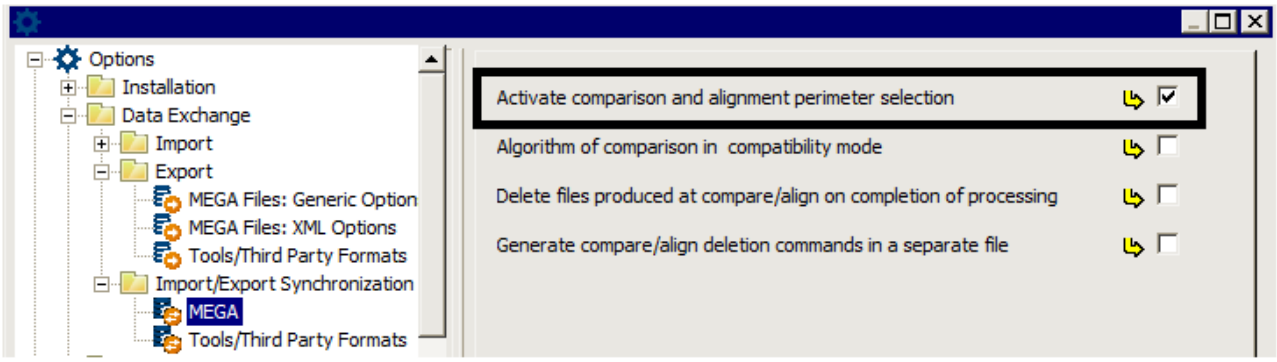
The perimeter is not visible as standard in the dialog box; it is activated by selecting the "Activate export perimeter selection" option as below:



"Compare and align" MetaTool

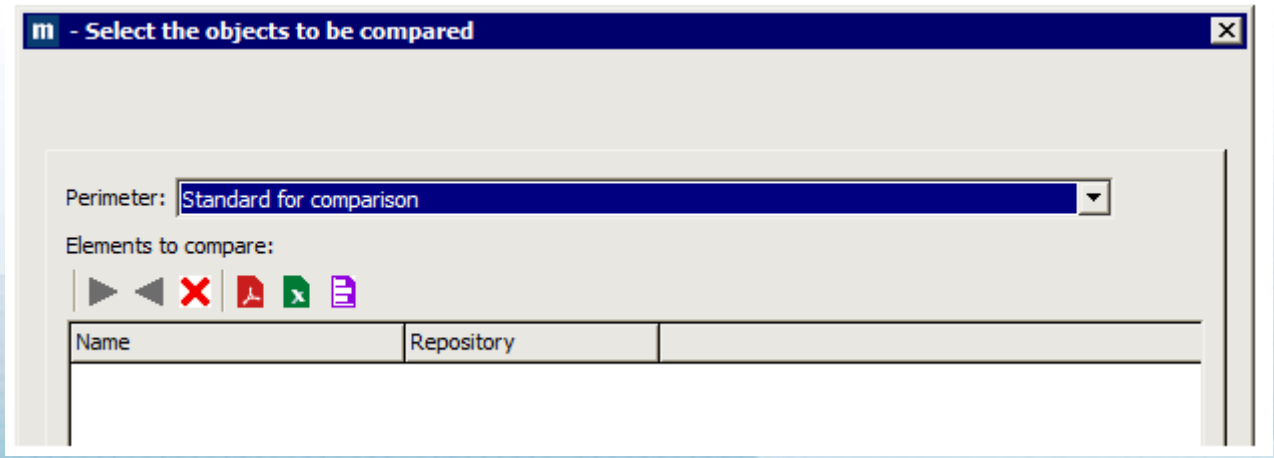
The "Compare and Align" MetaTool enables comparison of objects and links in two MEGA repositories. Processing of compare handles either all objects and links in the repository, or the set of objects and links built from a perimeter and from one or several root objects.

This functionality is available from the menu of an object by selecting "Manage/Compare and Align", from the desktop menu by selecting "Tools/Manage/Compare and Align", or from the menu of an environment by selecting "Compare and Align"



Perimeter visibility in this MetaTool is managed by the "Activate compare and align perimeter selection" available in options:

The "Standard for comparison" perimeter is the MEGA standard perimeter for use of the "Compare and align" MetaTool.

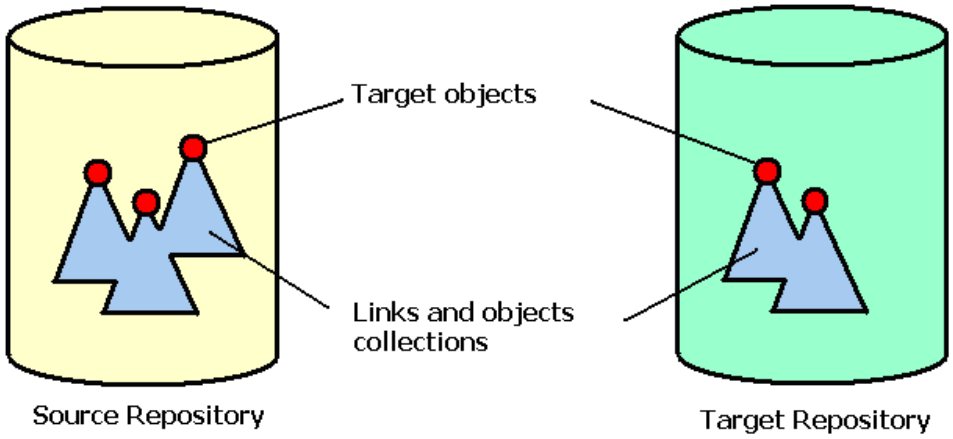


Building a set by propagation

When we install a perimeter configuration, we need to know how it will be interpreted by the propagation processing which builds the set of objects and links.

Root objects list

The user inputs a set of "Root" objects of the source repository, and a method of creating the set by propagation: the perimeter.



Root objects are not necessarily in the two repositories to be compared. The result will differ depending on whether processing is applied on a set of root objects, or if processing is applied to each root object.

Set of objects

The set of objects is built from root objects by propagation. A perimeter groups behavior of all MetaAssociations, and enables addition or not of an object to the set of objects opposite the MetaAssociation.

The MetaAssociations are directional. Each MetaAssociation has a "Major" MetaClass and a "Minor" MetaClass. Behavior is specified for each perimeter on "MajorToMinor" and "MinorToMajor" MetaAttributes of a MetaAssociation. Behavior can also be specified on the MetaAssociationType connected to the MetaAssociation.

Taking MetaClassA as the root object, we browse the MetaAssociation in direction Major to Minor. The behavior taken into account will be that specified on the MajorToMinor MetaAttribute. Depending on the value of this behavior, the opposite object (ie. MetaClassB) will be added or not to the set with this behavior. If the opposite object is accessible via another MetaAssociation or set of MetaAssociations with different behavior, behavior of this object in the set of objects is updated with the least restrictive behavior (orders being "DEEP", "STANDARD" and "LINK"). Only objects of « DEEP » or « STANDARD » behavior are used in the « Comparison » MetaTool. Objects with "LINK" behavior are used in link processing only.

Root object: by default, this object is added to the set with "DEEP" behavior.

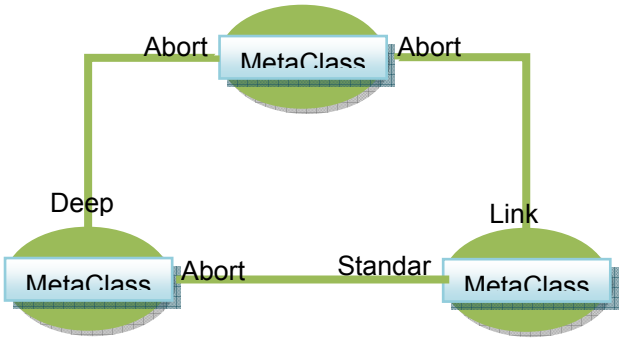
"DEEP" behavior: the opposite object is placed in the set with "DEEP" behavior and propagation continues on this object.

"STANDARD" or "LINK behavior: the opposite object is placed in the set with this behavior and propagation processing stops.

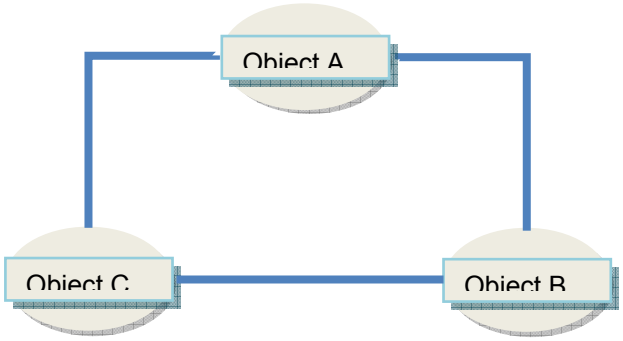
"ABORT" behavior: the opposite object is not placed in the set and propagation processing stops.

"COMPUTED" link: MetaAssociation behavior is not sufficient to define behavior on this particular link and behavior is computed by a macro which gives behavior related to the current object, which can be "DEEP", "STANDARD", "LINK" or "ABORT". The object is processed in the same way as other objects with the same behavior.

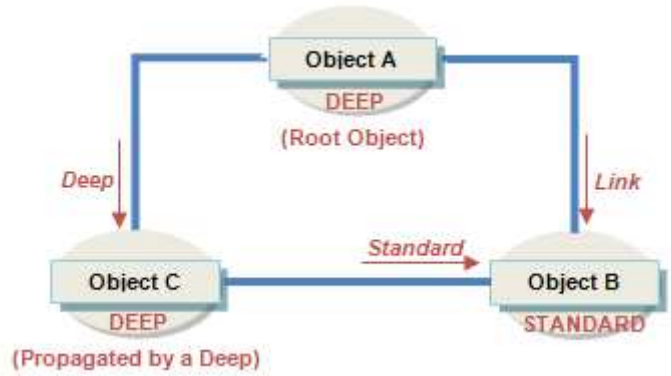
The next setting



with the next data



Gives us the following path with behavior of objects; we note that the "Object B" is browsed by "LINK" and by "STANDARD", finally taking "STANDARD" behavior.



Set of links

It is then important to see the repository as a chart rather than a tree, ie. that an object can be browsed by different links. The set of links is built from behavior of objects contained in the set of objects obtained in the previous paragraph.

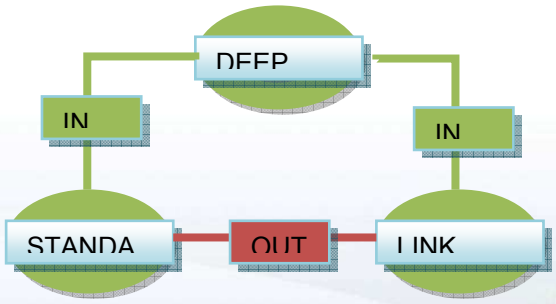
We list all links contained in the perimeter, ie. all occurrences of links between two objects belonging to the set of objects. Depending on the nature of these two objects ("DEEP", "STANDARD" or "LINK"), the link is added or not to the set according to the following table:

		Object		
		Deep	Standard	Link
Opposite object	Deep	IN	IN	IN
	Standard	IN	IN	OUT
	Link	IN	OUT	OUT

IN: the link is added to the set.

OUT: the link is not added to the set.

Example:

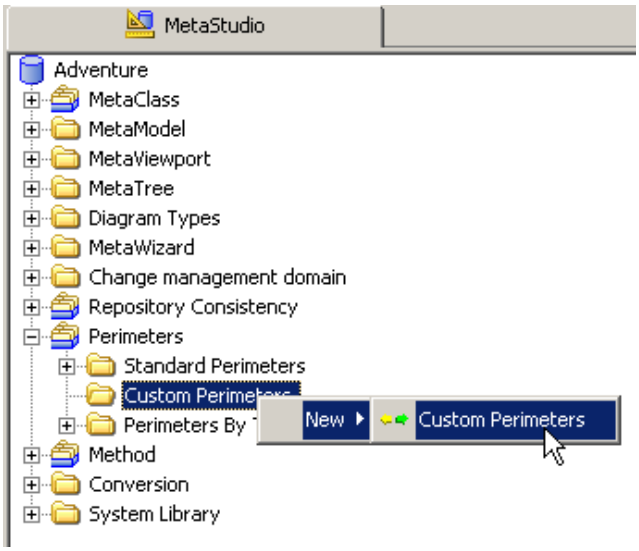


The (IN) link between the "DEEP" object and the "STANDARD" object is taken into account in building the set of links. The (IN) link between the "DEEP" object and the "LINK" object is taken into account in building the set of links. The (OUT) link between the "STANDARD" object and the "LINK" object is not taken into account in building the set of links.

In the "Compare" MetaTool, the "IN" links are taken into account, but not the "OUT" links.

Creating a perimeter

To create a perimeter, navigation of the "Meta Studio" tab in the desktop must be activated. This tab is available with "Expert" access to the metamodel. We then right-click the "Perimeter" folder and select "New/Perimeter".

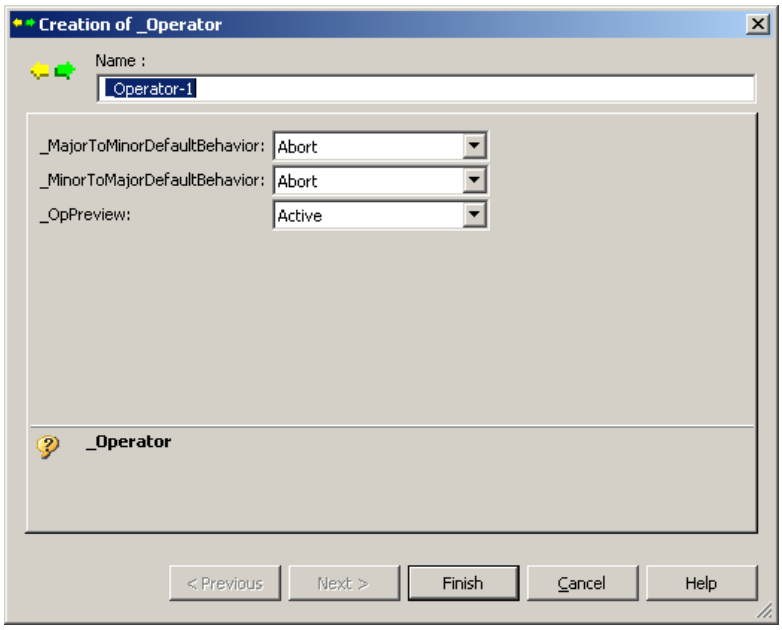


Certain properties are specified in the dialog box that appears:

- _MajorToMinorDefaultBehavior: This is the default behavior we want to give to links in direction Major Object to Minor Object. This property is positioned on "Abort". The strategy is to not propagate links so as to avoid a high-volume object set. We then specify behavior on the links we want to browse.

- _MinorToMajorDefaultBehavior: This is the default behavior we want to give to links in direction Minor Object to Major Object. This property is also positioned on "Abort".

- _OpPreview: (Active\Inactive) This property enables activation of perimeter display in the configuration MetaTool, which is covered in the next chapter.

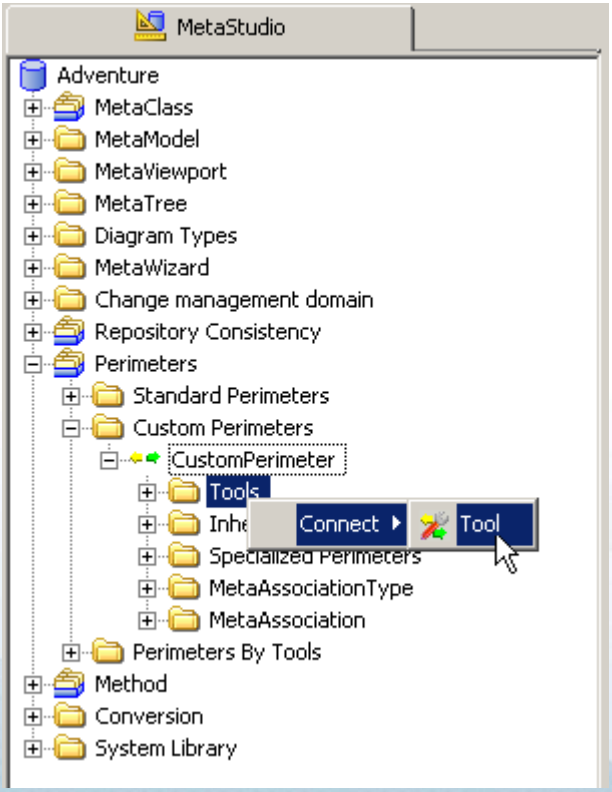

Creation of _Operator
Name :
Operator-1
_MajorToMinorDefaultBehavior:
Abort
_MinorToMajorDefaultBehavior:
Abort
_OpPreview:
Active
_Operator
< Previous
Next >
Finish
Cancel
Help

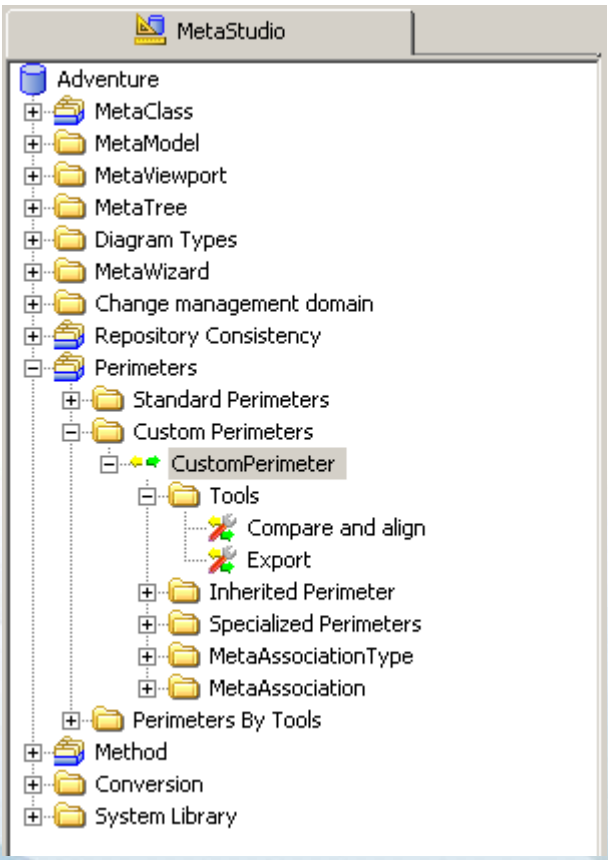
Configuring a perimeter

Using a perimeter in a MetaTool

To be able to use a perimeter in a MetaTool, we must connect the perimeter to the MetaTool concerned. A perimeter can have several MetaTools, enabling reuse of a perimeter in several different contexts.
Similarly, a MetaTool can have several perimeters, enabling us to build several different sets of objects and links from a root object.

Having created the new perimeter "CustomPerimeter" for example, we connect it to the "Export" and "Compare and align" MetaTools as here: (in this version, there are only these two MetaTools)

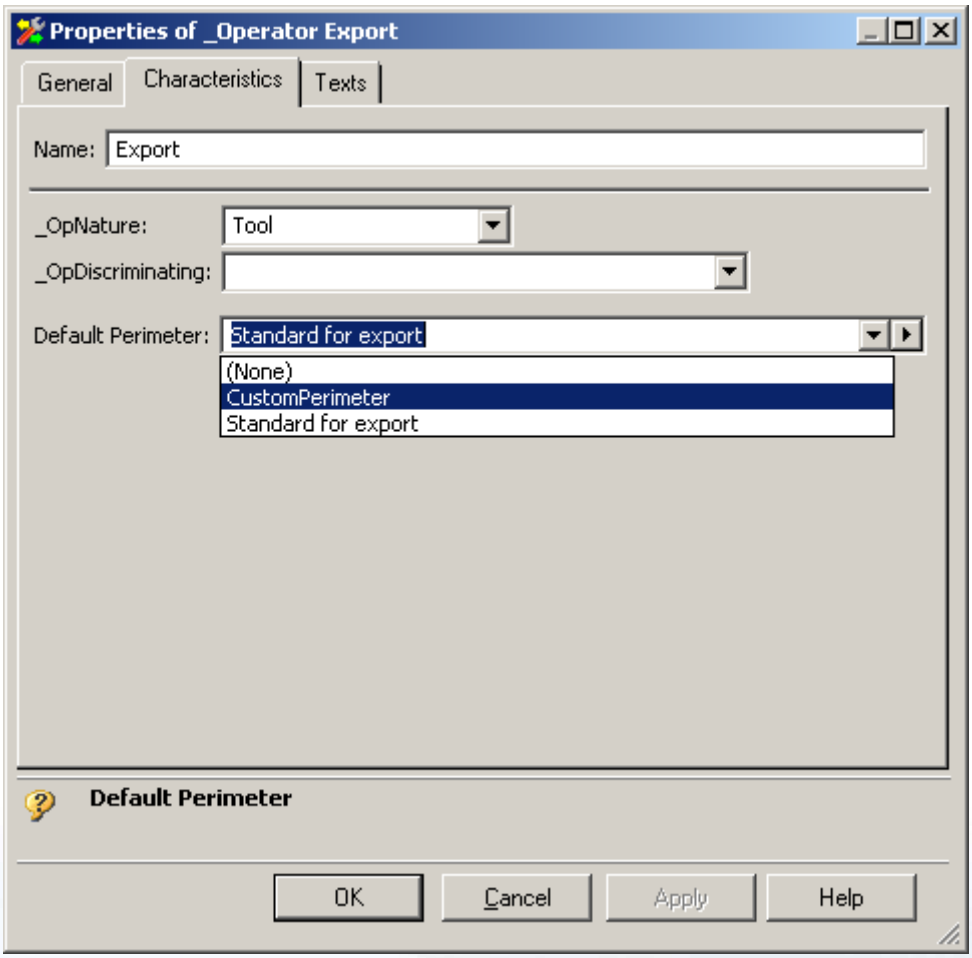




Default perimeter in a MetaTool

It is possible to change default perimeter for a MetaTool. To do this, the perimeter must be already connected the MetaTool and we modify the "Default perimeter" property.

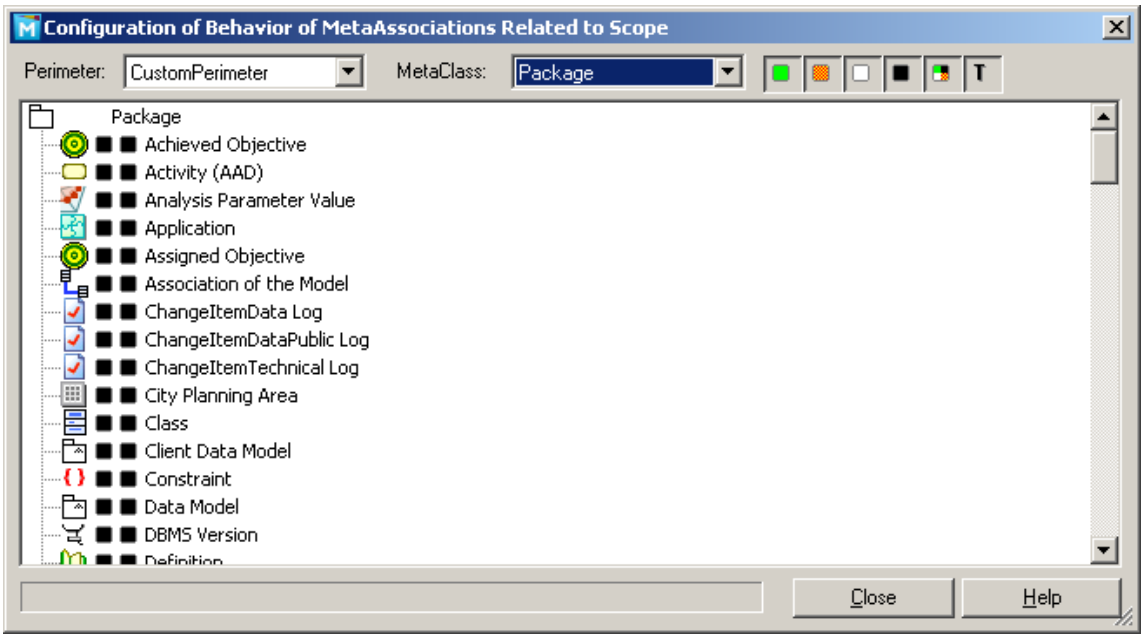
In the example below, we change default perimeter of the "Export" MetaTool to insert the perimeter we have created.



Configuring perimeter links

This chapter covers the configuration of links required for propagation of links and objects in building a set from a perimeter.

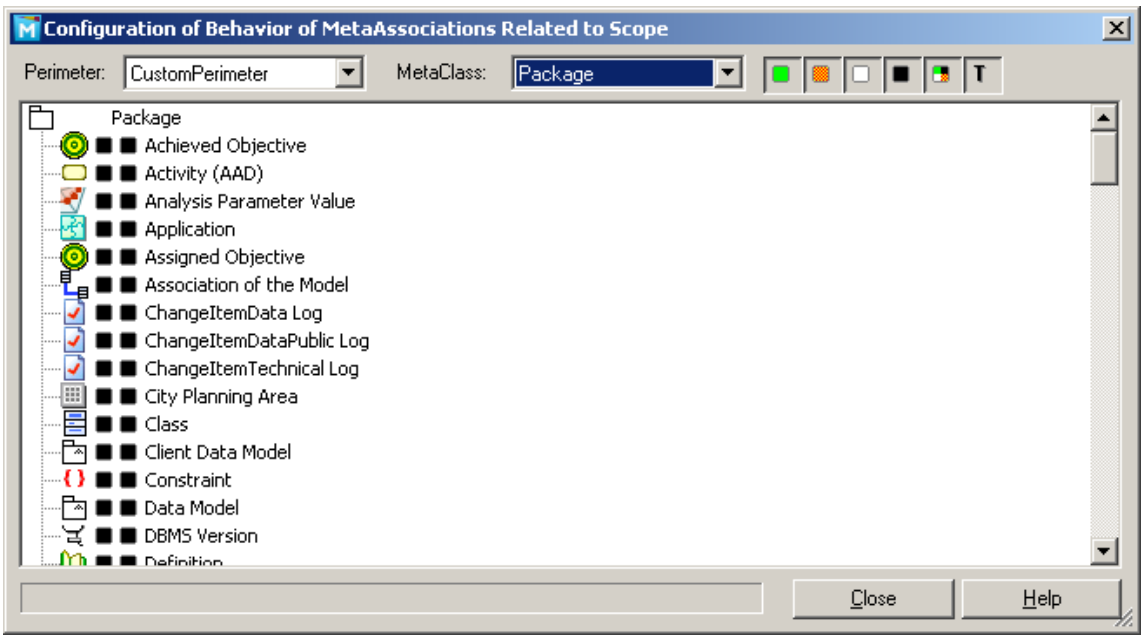
To open the configuration dialog box, select "Manage/Configure" in the perimeter menu:



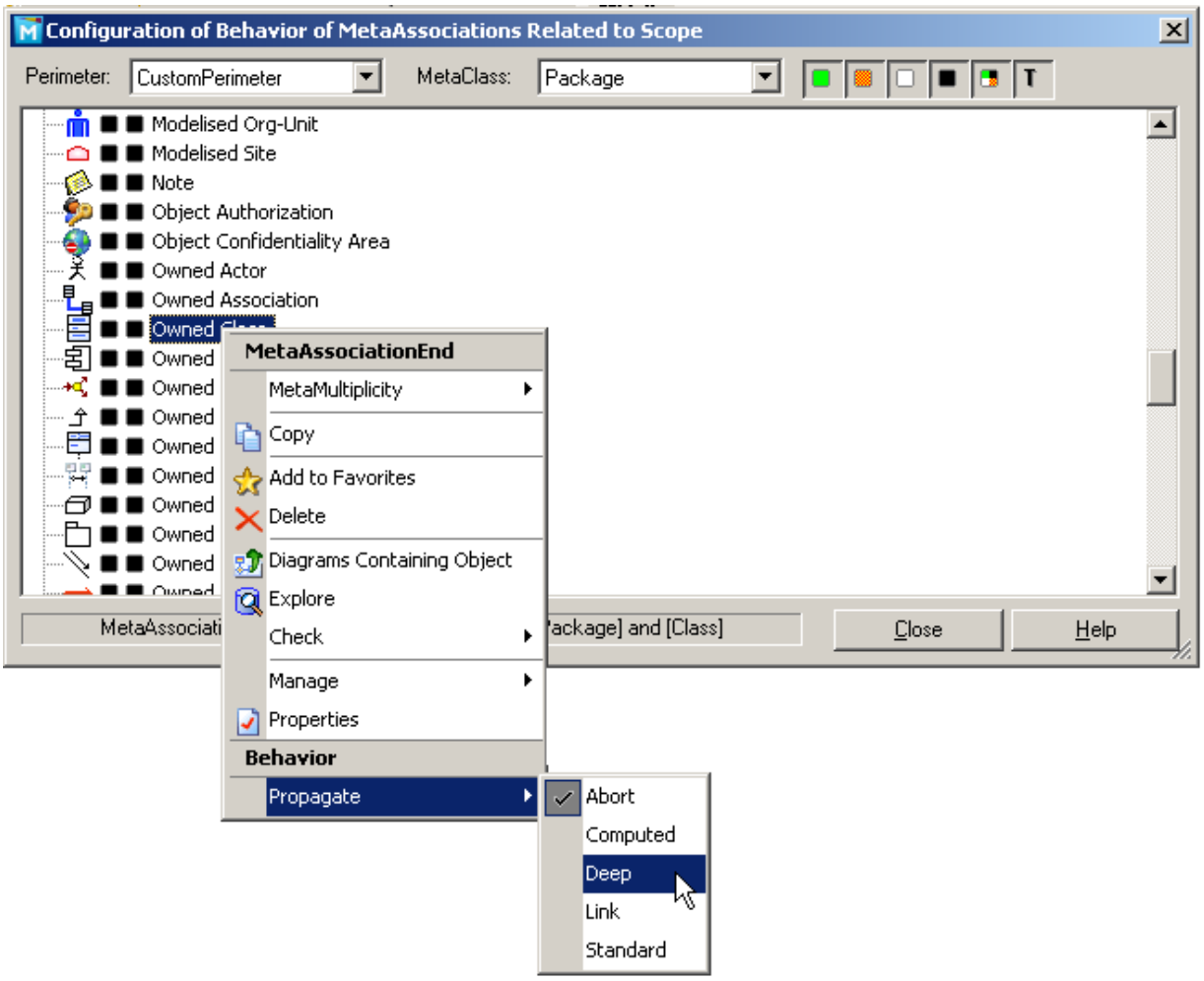
The "Perimeter" box enables selection of the perimeter we want to configure (only perimeters of which the"_OpPreview" MetaAttribute is "Active" are visible.
The "MetaClass" box enables selection of the MetaClass from which links will be browsed, and we begin by using the MetaClass of the root object, for example "Package".
The toolbar includes buttons to filter links (MetaAssociationEnd) according to their behavior:

- Filter the links that have "Deep" behavior.
- Filter the links that have "Standard" behavior.
- Filter the links that have "Link" behavior.
- Filter the links that have "Abort" behavior.
- Filter the links that have "Computed" behavior.

The "T" button enables display in the tree of default behavior of the link before customization, which is "Abort" in our example:

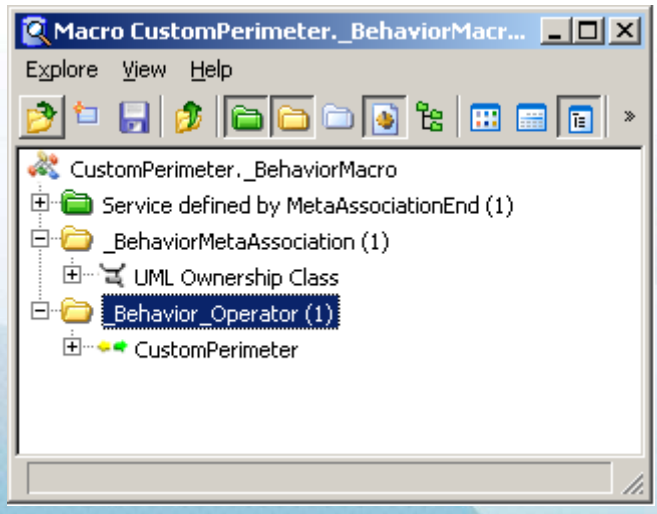


To modify propagation behavior of a link, select the new behavior in the "Propagation" menu. The current behavior is represented by a tick alongside the behavior. The standard menu of a link is also accessible.



The new behavior will be active for the next use, except for "Computed" behavior which requires additional configuration.

This behavior uses a macro which must be present, and for this we must create a macro and then connect it to the current link and perimeter as here:



You must then add VB code in the macro which uses the following method:

```
Function BehaviorCompute(oObject As MegaObject, oPerimeter As Variant) As String
```

Parameter:

- *oObject* is the object which is processed when building the set. It is obtained from propagation of another object: its parent; it is possible to recover this and the link used for propagation.
- *oPerimeter* is the perimeter object, enabling reuse of a macro for several perimeters if required.

Return:

- BehaviorCompute: This function returns the behavior of the link. It could have been computed in several ways, as shown in the example below.

"Computed" behavior example:

We position behavior of the link [Package/Owned Class] on "Computed" to restrict the list of classes owned by a package. We want to build a set of objects with owned classes that are not interfaces.

The following code is applied to the macro:

```
'MegaContext(Fields,Types)
Option Explicit

' ----------------------------------------------
' --  Variables
' ----------------------------------------------
Const cStereotypeInterface = "(NHm3AzqouC0"

' ----------------------------------------------
' --  BehaviorCompute
' ----------------------------------------------
Function BehaviorCompute(oClass As MegaObject, oPerimeter As Variant) As String
  ' Default Behavior : DEEP
  BehaviorCompute = "D"

  If (oClass.GetProp("~wzN3o0nDp840[Stereotype]") = cStereotypeInterface) Then
    ' Behavior : ABORT
    BehaviorCompute = "A"
  End If

End Function
```

"Compare and align" MetaTool filter

The "Standard for comparison" perimeter, which is the default perimeter of the "Compare and align" MetaTool, has a particularity specific to the "Compare" functionality. On this perimeter these is a specific configuration that enables:

- Management of the MetaAttributes used for comparison of objects and links.
- Exclusion of certain MetaClasses and MetaAssociations.

Technical installation

The configuration must be declared in the _Settings text of the perimeter used in the "Compare and align" MetaTool.
If we use the comparison functionality on the entire repository, the configuration used is that present on the default perimeter of the MetaTool. If this text is empty, configuration is taken on the _Settings text of this MetaTool.
If configuration is defined on a perimeter, that of the MetaTool will not be taken into account. It is advisable to filter Meta Attributes too technical :

[CmpFilteredMetaAttributes]
~510000000L00[CreationDate]=0
~710000000T00[LinkCreationDate]=0
~610000000P00[UpdateDate]=0
~810000000X00[LinkUpdateDate]=0
~(10000000v30[_Creator]=0
~720000000T40[_LinkCreator]=0
~b10000000L20[_Modifier]=0
~920000000b40[_LinkModifier]=0
~520000000L40[_CreateVersion]=0
~620000000P40[_UpdateVersion]=0
~f20000000b60[Update Log]=0
~a20000000H60[LanguageUpdateDate]=0
~b20000000L60[LinkLanguageUpdateDate]=0

Filtering MetaAttributes

To compare an object or link, all MetaAttributes are taken into account, which in certain contexts is pointless and produces too many differences.

Two general filter strategies are installed: either processing takes all MetaAttributes into account, or it takes none. To specify, position the CmpFilterMetaAttributes variable on 1 or on 0 in the "CmpFilterGeneral" section as here:

[CmpFilterGeneral]

CmpFilterMetaAttributes= {0|1}

If CmpFilterMetaAttributes= 1 (default value), all MetaAttributes of the object or link are used for comparison.

If CmpFilterMetaAttributes= 0, no MetaAttribute is taken.

When strategy has been defined, we add an additional filter, increasingly detailed to suit requirements. We can begin by specifying the MetaAttributes we want to always or never use for all objects and links at comparison. For this, we use the "CmpFilteredMetaAttributes" section, with the following syntax:

[CmpFilteredMetaAttributes]

<Field of MetaAttribute>= {0|1}

Ex: ~510000000L00[Date de creation]=0

If <Field of MetaAttribute>=0, the MetaAttribute is not taken into account for all MetaClasses and MetaAssociations.

If <Field of MetaAttribute>=1, the MetaAttribute is taken into account for all MetaClasses and MetaAssociations.

It is then possible to similarly configure for the MetaAttributes we want to manage or not according to the MetaClasses in the "CmpFilteredMetaAttributesByMetaClasses" section, with the following syntax:

[CmpFilteredMetaAttributesByMetaClasses]

<Field of MetaClass>, <Field of MetaAttribute>= {0|1}

Ex: ~d20000000U20[_Collection],~510000000L00[Creation date]=0

If <Field of MetaClass>, <Field of MetaAttribute>=0, the MetaAttribute is not taken into account for this MetaClass.

If <Field of MetaClass>, <Field of MetaAttribute>=1, the MetaAttribute is taken into account for this MetaClass.

We can similarly filter MetaAttributes on the MetaAssociations in the "CmpFilteredMetaAttributesByMetaAssociations" section, with the following syntax:

[CmpFilteredMetaAttributesByMetaAssociations]

<Field of MetaAssociation>, <Field of MetaAttribute>= {0|1}

Ex: ~d20000000U20[_Collection],~510000000L00[Creation date]=0

If <Field of MetaAssociation>, <Field of MetaAttribute>=0, the MetaAttribute is not taken into account for this MetaAssociation.

If <Field of MetaAssociation>, <Field of MetaAttribute>=1, the MetaAttribute is taken into account for this MetaAssociation.

Filtering MetaClasses and MetaAssociations

It is possible to filter the MetaClasses we do not want to take into account at comparison. In the "CmpExcludedMetaClasses" section, we list the MetaClasses we want to exclude, with the following syntax:
[CmpExcludedMetaClasses]
<Number>=<Field of MetaClass>
1=~o20000000A30[_Dispatch]
The <Field of MetaClass> MetaClass is excluded from comparison.

As for MetaClasses, we can filter MetaAssociations we want to exclude by listing these in the "CmpExcludedMetaAssociations" section as below.
[CmpExcludedMetaAssociations]
<Number>=<Field of MetaAssociation>
1=~oXRVCA8Dp8i1[Owned class]
The <Field of MetaAssociation> MetaAssociation is excluded from comparison.

Standard configuration

A configuration is already present on the "Compare and align" MetaTool, excluding comparison of technical MetaClasses such as "_Dispatch" for example, and not take into account technical MetaAttributes such as"Logfile" for example.
The following is complete standard configuration:
[CmpFilterGeneral]
CmpFilterMetaAttributes=1

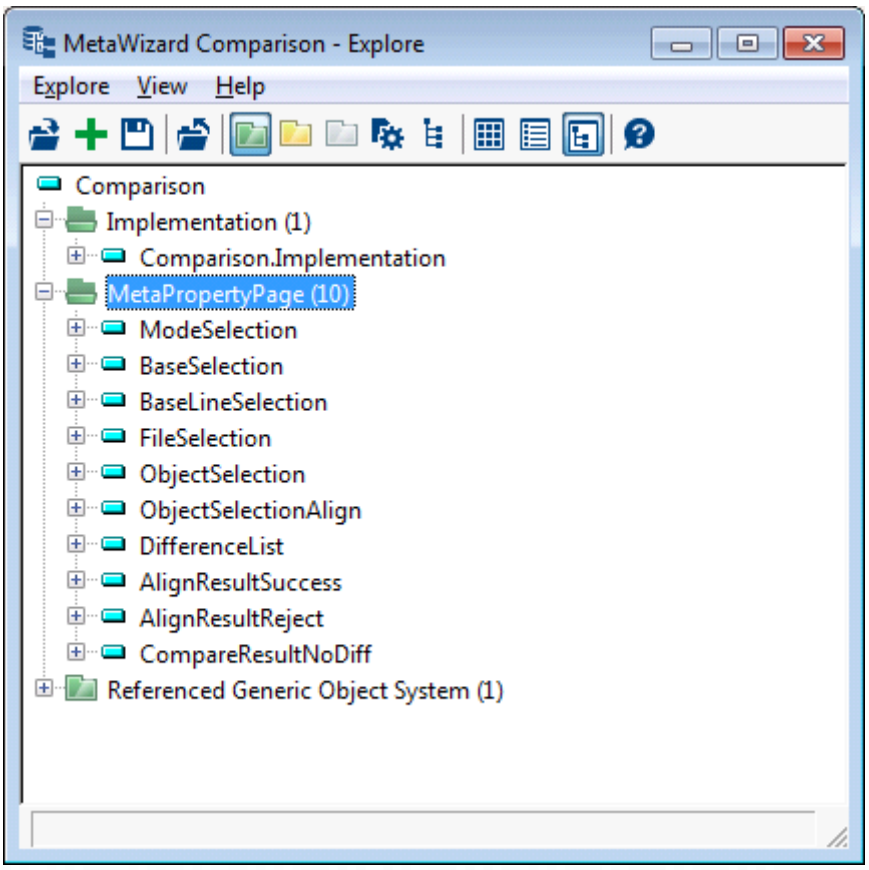
[CmpFilteredMetaAttributes]
~510000000L00[Creation date]=0
~610000000P00[Modification date]=0
~(10000000v30[Creator]=0
~b10000000L20[Modifier]=0
~520000000L40[Creation version]=0
~620000000P40[Modification version]=0
~f20000000b60[Logfile]=0

[CmpExcludedMetaClasses]
1=~o20000000A30[_Dispatch]
2=~n20000000630[_DispatchData]
3=~nFhC9FZc0P30[Recent query]
4=~d20000000U20[_Collection]
5=~e20000000Y20[_CollectionItem]
6=~f20000000c20[_CollectionItemLink]
7=~h20000000k20[ChangeItemData]
8=~l20000000(20[ChangeItemDataPublic]
9=~r20000000M30[ChangeItemDataTechnical]
10=~i20000000o20[ChangeItemSystem]
11=~m20000000230[ChangeItemSystemPublic]
12=~t20000000U30[ChangeItemSystemTechnical]

Compare Tool API (to compare and align)

CompareTool is a MEGA object enabling management of object comparison between the current private workspace and a target repository in VBScript. Comparison enables creation of an update file designed to align the target repository. This functionality is available with the "MEGA Supervisor" module.

NOTE: The comparison wizard provided as standard is a comprehensive and well maintained example of API use.



Implementation to VbScript

MegaObject can be installed using the following methods:

- Function OpenCompareToolEx(ByVal oParameterSource As Object, ByVal oParameterTarget As Object, ByVal strOption As String) As Object

This method enables instancing of the CompareTool object. Comparison is between a source and a target repository. You can either open the repository you wish to compare or provides the required parameter to let the compare tool opening the repository for you.

The two first parameter can be either a MEGARoot or a String. In case of a MEGARoot it'll be used by the CompareTool. If it is a String, it should describe the repository to be used:

strParameter
This is a string option of the form:
<optionName1>=<optionValue1>,<optionName2>=<optionValue2>,…
DbTarget = it can be either a base name, a base id or "TRANSACTION" or "FILE". If you select "TRANSACTION" the current private workspace will be used. If you select "FILE", a temporary repository will be created and your data file imported into it.
FileTarget = When specifying "FILE" as DbTarget, here is the path of the data file to import.
DbSource = Like DbTarget but for the source
FileSource = Like FileTarget but for the source
BaselineSource = When selecting a base name or a base id as DbSource you can open the repository according to an archived state.
BaselineTarget = Like BaselineSource for the target. Of course when a repository is opened according to an archived state, it is read only. You will not be able to align comparison result.

The last parameter is for option. You can leave it empty or use the previous syntax: <optionName>=<optionValue>
Compatibility = 0 or 1 lets you select the old comparison algorithm for compatibility purpose.

- **Sub Close()**
  Frees the target repository private workspace and frees the MegaObject variables.

Attributes

- **ContinueOnConfidential:**
  Enables specification of whether comparison processing should be aborted if a confidential object is present in the collection of objects to be compared (in source or target repository).
  TRUE: comparison processing is not aborted.
  FALSE: processing is aborted; this is the default value.

- **Perimeter:**
  Enables modification of the perimeter of extraction of sets of objects and links to be compared in repositories. This attribute is not used when the "CompareAll" method is used.

- **FoundConfidential:**
  This attribute is specified by comparison processing to indicate presence of a confidential object in one of the two object collections to be compared.

Methods

Comparison between complete repository or object sets is necessary to enable generation of an alignment file.

- **Sub CompareAll()**
  Compares all objects and links of the source repository related to the target repository.

- **Sub CompareObjects(mgobjcolFromSource as MegaCollection, mgobjcolFromTarget as MegaCollection)**
  Compares the a set of objects and links from repository source and/or target.
  Parameter:
  mgobjcolFromSource*: Object set of the source repository enabling, with assistance of the perimeter, constitution of a set of objects and links for the two repositories to be compared (source and target).*

mgobjcolFromTarget: Object set of the target repository enabling, with assistance of the perimeter, constitution of a set of objects and links for the two repositories to be compared (source and target).

- **Sub SetGenerateOutputOption(strOptionName As String, varOptionValue As Variant)**
  Enables specification of variables used at alignment file generation.
  Parameter:
  strOptionName*: Name of variable identifying the Option. The following three options are available: "UpdateExtract", "DeleteFile", "IgnoreAttributes" and "TransferredObjects".*
  varOptionValue: *Option value. The value depends on Option:*
  ** "DisconnectExtract": This option enables specification of whether the alignment file generates "disconnect" commands rather than "delete" commands; default value of this option is FALSE.*
  ** "DeleteFile: A second file name can be configured using this option in order to manage "delete" commands. The alignment file takes into account all actions, and if a file name is defined for this option, all "delete" commands are sent to this file.*
  ** "IgnoreAttributes": Processing compares all object MetaAttributes, but with this option it is possible to ignore certain MetaAttribute types. Each element is separated by " ; ". MetaAttribute types are as follows: "Date" ("Date" MetaAttributes are not taken into account at generation of alignment files, for example: Modification Date),"Creator" (MetaAttributes linked to object creation are ignored), "Authorization" (MetaAttributes linked to authorization are not managed), "Comment" (Comment MetaAttributes are ignored), "Order" (Order MetaAttribute is not taken into account). A MetaAttribute can be specifically ignored by adding it to the list. This MetaAttribute will be ignored for all objects and links,*
  for example: oCompareTool.SetGenerateOutputOption "IgnoreAttributes", "Date*; ~610000000P00[Date de modification];~b10000000L20[Modificateur];Order"*

** " TransferredObjects": This option lets the user choose to include or not objects of merging (_TransferedObject). By default this option is true. This value is a Boolean.*

- **Sub ResetGenerateOutputOptions()**
  This method enables reinitialization of options that were specified using the previous method: "SetGenerateOutputOption".

- **Sub GenerateOutput(strOutputFileName As String, strOutputType As String)**
  This method generates the alignment file named "strOutPutFileName". If the "DeleteFile" option was specified a file with "delete" commands is created.
  Parameters:
  strOutputFileName*: Alignment file name.*
  strOutputType*: enables modification of alignment file format. Default format is "MGR".*

- GetRootSource

Returns the MEGA Root used for comparison as Source Repository. This is convenient especially if you opened is through the call to OpenCompareTool

- GetRootTarget

Just like GetRootSource, returns the MEGA Root of the target repository.

- Align

This function will align the differences with the target repository. If an error occurs it returns a path (string) to the rejected commande file. If there are no rejected command an empty string is returned. You can only align if the target repository is writable. As an instance, you cannot use align if the target repository is an archived state based on a dispatch.

- Commit

This method must be called after the align function and before closing the target repository. Commit will save every update on the target repository after the alignment.

- Rollback

This method can be called instead of “Commit” to cancel every update processed in the target repository with the “align” function.

- SetMatchingObjectMacro(strMacroId)

You can call this method specifying a macro Id.

When comparing two repositories, the standard behavior to find the matching object in the other base is to look for an object with the same idabs.

The macro must implement this function:

Function GetMatchingObject(oContext, oReference)

oReference is the MEGAObject that is searched in the other repository

oContext is an object with the following available methods:

- GetDirection: returns a String with two possible values:
  - "ST": oReference is from the Source repository and the macro should look for the matching object in the Target repository.
  - "TS": oReference is from the Target repository and the macro should look for the matching object in the Source repository.
- GetMatchingObjectRoot: returns the MEGARoot of the repository where the matching object is searched.

This macro must always return a MEGAObject (even if no matching object have been found).

Here is a sample function with quite the same behavior as the standar.

```
Function GetMatchingObject(oContext, oReference)
    Dim oMatchingObjectRoot
    set oMatchingObjectRoot = oContext.GetMatchingObjectRoot()
    Dim szDirection
    szDirection = oContext.GetDirection()
    Dim oMatchingObject
    Set oMatchingObject =
oMatchingObjectRoot.GetCollection(oReference.GetClassId).Item(oReference.GetId())
    Set GetMatchingObject = oMatchingObject
End Function
```

Here, ".item" on a MEGACollection always returns a mega object even if the item is not found.

- GetDiffResultCollection

After a CompareAll or CompareObjects this methods returns a MEGACollection containing all differences.

This MEGA Collection contains object with attributes described with the following Informal Query: "~CcRkodhCELgI[Compare.DiffResult.Collection]"

- “~410000000H00[Order]” Number: This is the order of the current difference.
- “~310000000D00[Absolute Identifier]” Idabs: This is the identifier of the current difference.
- “~snk(al)kEboV[MetaPicture]” Idabs: this is the picture’s idabs of the current difference (according to its type: create, delete, link, …).
- “~K3Rv(lEmEvlI[DiffTypeMetaPicture]” MegaIdentifier: this metapicture depends if it is a update, a connect, a delete…
- “~3SqlLqFEEPoF[Kind]” Enumeration: this is the kind of the difference, a link or an object. Many attributes are available only link or object kind.
- ”~)TqlXLHEEn3G[SourceAvailable]” Boolean: true if the element is available in the source repository.
- ”~PUqliLHEEj4G[TargetAvailable]” Boolean: true if the element is availbale in the target repository.
- ”~ZApdg(vjEX67[Difference]” Enumeration: It is the difference type: Connect, create, delete,…

0: Aucune différence

1: Créé

2: Modifié

3: Supprimé

4: Relié

5 : Délié

- ”~CVJie9xjEzg5[Target]” String : The metaclass name of the object or the metaassociation name of the link.
- ”~ASJi1AxjEHj5[Object 1]” String: the object name if the difference kind is an object. The object next to the link if the difference is a link.
- ”~BTJiGAxjELm5[Object 2]” String: if the difference is a link the other object name next to the link.
- ”~ITqlPgcDEb8F[ObjectSourceIdabs]” Mega Idabs: if the difference is about an object, this property is the idabs of the object in the source database when it exists.
- “~jSqlClcDEzIF[ObjectSourceMetaclassIdabs]” Mega Idabs: if the difference is about an object, this property is the idabs of the metaclass of the object in the source database when it exists.
- “~sTqlmgcDEP9F[ObjectTargetIdabs]” Mega Idabs: if the difference is about an object, this property is the idabs of the object in the target database when it exists.

- “~ETqlBtcDEzKF[ObjectTargetMetaclassIdabs]]” Mega Idabs: if the difference is about an object, this property is the idabs of the metaclass of the object in the target database when it exists.
- “~LSqlvKfDE1OF[LinkSourceIdabs]” Mega Idabs: if the link exists in the source database, this property is the metaassociation idabs.
- “~9TqlULfDEzOF[LinkSourceMajorIdabs]” Mega Idabs: If the link exists in the source database, this property is the major metaassociationend idabs.
- “~5Vql)qGEErwF[LinkSourceMajorMetaclassIdabs]” Mega Idabs: if the link exists in the source database, this property is the metacalss idabs on the major side of the link.
- “~jTqlpLfDEvPF[LinkSourceMinorIdabs]” Mega Idabs: If the link exists in the source database, this property is the minor metaassociationend idabs.
- “~FSqlhrGEEjyF[LinkSourceMinorMetaclassIdabs]” Mega Idabs: if the link exists in the source database, this property is the metacalss idabs on the minor side of the link.
- “~OVqliuKEE59G[LinkTargetIdabs]” Mega Idabs: if the link exists in the target database, this property is the metaassociation idabs.
- “~LUqlCMfDErQF[LinkTargetMajorIdabs]” Mega Idabs: If the link exists in the target database, this property is the major metaassociationend idabs.
- “~uSql5sGEEfzF[LinkTargetMajorMetaclassIdabs]” Mega Idabs: if the link exists in the target database, this property is the metacalss idabs on the major side of the link.
- “~1UqlwOfDEfUF[LinkTargetMinorIdabs]” Mega Idabs: If the link exists in the target database, this property is the minor metaassociationend idabs.
- “~OTqlMsGEEb(F[LinkTargetMinorMetaclassIdabs]” Mega Idabs: if the link exists in the target database, this property is the metacalss idabs on the minor side of the link.

Sample (VBScript)

```
Option Explicit

Const cszDbSourceName = "Dev"
Const cszDbTargetName = "Prod"

Function DifferenceTypeName(strDiffValue)
  DifferenceTypeName = ""
  dim mgDiffTypeCurrent
  For Each mgDiffTypeCurrent In
GetObjectFromId("~ZApdg(vjEX67[Difference]").GetCOllection("~uGZC89p5ua00[_ParameterValue]")
    If strDiffValue  = mgDiffTypeCurrent.GetProp("~L20000000L50[Valeur interne]") Then
```

```
    DifferenceTypeName = mgDiffTypeCurrent.Name
    Exit For
   End If
  Next
End Function

Dim mgCmpTool
Set mgCmpTool = GetRoot.OpenCompareToolEx("DbSource=" & cszDbSourceName, "DbTarget=" &
cszDbTargetName, “”)
If Not mgCmpTool Is Nothing Then
  print "Repository " & mgCmpTool.GetRootSource.Name & " opened as source"
  print "Repository " & mgCmpTool.GetRootTarget.Name & " opened as target"
  mgCmpTool.CompareAll
  Dim mgcolDiff
  Set mgcolDiff = mgCmpTool.GetDiffResultCollection
  Dim mgDiff
  For Each mgDiff In mgcolDiff
    Dim strDiff
    strDiff = mgDiff.GetProp("~ZApdg(vjEX67[Difference]")
    Dim strtarget
    strTarget = mgDiff.GetProp("~CVJie9xjEzg5[Cible]")
    Dim strObject1
    strObject1 = mgDiff.GetProp("~ASJi1AxjEHj5[Objet 1]")
    Dim strObject2
    strObject2 = mgDiff.GetProp("~BTJiGAxjELm5[Objet 2]")
    print DifferenceTypeName(strDiff) & ": [" & strtarget & "] " & strObject1 & "/" & strObject2
  Next
  mgCmpTool.Close
End If
```

Sample (Java)

```
final MegaRoot mgRoot = mgobjSource.getRoot();
final MegaCurrentEnvironment mgEnvironment = mgRoot.currentEnvironment();
final MegaToolkit mgToolkit = mgEnvironment.toolkit();
final MegaCollection mgcolOrgProcess = mgRoot.getCollection("~gsUiU9B5iiR0[Organizational Process]");
final MegaObject mgNewOrgProcess = mgcolOrgProcess.create();
final MegaObject mgobjPerimeter = mgRoot.getObjectFromID("~cj6s5ij3q400[Standard for comparison]");
if ((null != mgobjPerimeter) && mgobjPerimeter.exists()) {
  final MegaCollection mgcolObjectToCompare = mgRoot.getSelection("");
  mgcolObjectToCompare.add(mgNewOrgProcess);
  final MegaCompareTool oCompareTool = new MegaCompareTool(mgRoot, "DbSource=TRANSACTION",
"DbTarget=" + mgRoot.getProp("~Z20000000D60[Short Name]"), "");
  oCompareTool.perimeter(mgToolkit.getString64FromID(mgobjPerimeter.getID()));
  oCompareTool.compareObjects(mgcolObjectToCompare);
  final String strRejects = oCompareTool.align();
  if (0 == strRejects.length()) {
    oCompareTool.commit();
```

```
  } else {
    oCompareTool.rollback();
  }
  oCompareTool.close();
  mgcolObjectToCompare.release();
  mgobjPerimeter.release();
}
mgNewOrgProcess.release();
mgToolkit.release();
mgEnvironment.release();
mgcolOrgProcess.release();
mgRoot.release();
```

Export Tool API (to Export)

ExportTool is a MEGA object enabling management of standard export of an object or collection of objects in VBScript or JAVA. This functionality is available with the "MEGA Supervisor" module.

Implementation to VbScript

MegaObject can be installed using the following methods:

Syntax:

Function OpenExportTool () As MeagObject

This method returns an instance of the "ExportTool" MegaObject.

Attributes

- **Propagate:** Allows you to specify if object roots must be propagated.
  - TRUE: To propagate; this is the default value.
  - FALSE: To take object roots only.

- **TransferedObject:** Takes account of transfered objects.
  - TRUE: The transfered objects are part of object roots; this is the default value.
  - FALSE: No transfered objects taken.

- **Format:** Choice of format of an exchange of data.
  - MGR: Ansi text data exchange; this is the default value**.**
  - XML: XML data exchange.

- **Perimeter:** Perimeter is used in Export Processing to extract sets of objects and links to generate the export file. The default perimeter is defined on MetaTool “Standard of Export”.

- **ContinueOnConfidential:** Enables specification of whether export processing should be aborted if a confidential object is present in the collection of objects to be exported.
  - TRUE: Export processing is not aborted; this is the default value.
  - FALSE: Export processing is aborted.

- **FoundConfidential:** This attribute is specified by Export processing to indicate presence of a confidential object in object collection. You cannot modify this attribute.

Methods

Export Function

Export an object or a collection of object in data exchange file.

Syntax:

```
Sub Export (
   oObject As MegaObject,
   szFilename As String
)
```

Parameters:

oObject [In]

MegaObject

This object is root of Export processing.

szFileName [In]

String

This is the name of the result file. It is the full name: local or network, directories, short name of the file and Extension.

Syntax:

```
Sub Export (
   oObjectColl As MegaCollection,
   szFilename As String
)
```

Parameters:

oObjectColl [In]

MegaCollection

This collection of objects is root of Export processing.

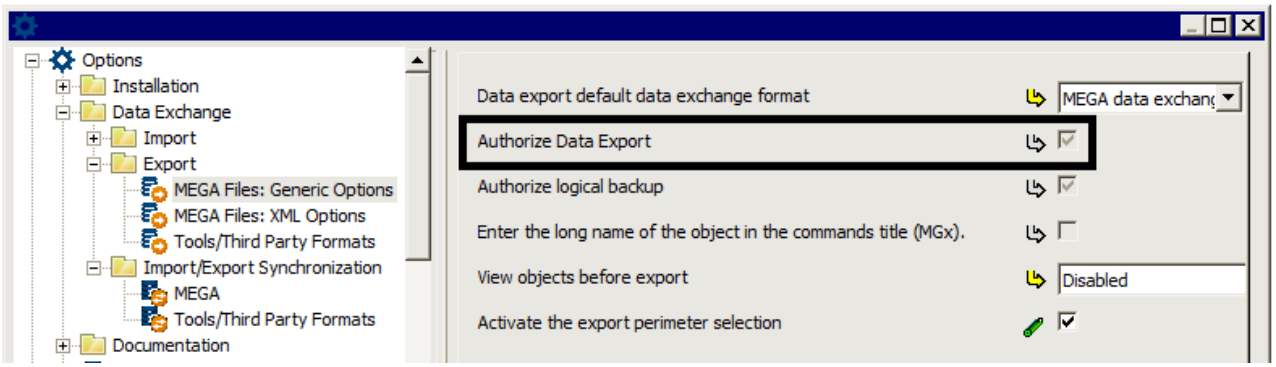
szFileName [In]

String

This is the name of the result file. It is the full name: local or network, directories, short name of the file and Extension.

Remarks

The option “Authorize Data Export’ must be activated to use the ExportTool component.



Sample

```
Option Explicit

' ----------------------------------
' -- ExportTool                V1.0
' ----------------------------------
' Samples of ExportTool object installation

' Creation of 'Component ExportTool' :
Dim oExportTool
Set oExportTool = OpenExportTool

' Display parameters :
Print "Default Parameters :"
Print "- Perimeter             = " & oExportTool.perimeter
Print "- Format                = " & oExportTool.Format
Print "- Propagate             = " & oExportTool.Propagate
Print "- TransferedObject      = " & oExportTool.TransferedObject
Print "- ContinueOnConfidential = " & oExportTool.ContinueOnConfidential
```

```
' ----------------------------------
' -- Sample 1 :
' ----------------------------------
' Export a package with no transfered object
' By Default : Propagation is Active and Format is 'MGR'
' -- Constants :
Dim strObjectFileName
strObjectFileName = "C:\TEMP\ObjectExport.MGR"
oExportTool.TransferedObject = FALSE
' --
Dim oPackage
Set oPackage = GetRoot.GetCollection("~h8rEkjZmo400[Paquetage]").Item(1)
If (oPackage.GetID <> 0) Then
  oExportTool.Export oPackage, strObjectFileName
  ' Dispaly result file in NotePad :
  Dim oObjWShell
  Set oObjWShell = CreateObject("WScript.Shell")
  oObjWShell.Exec """C:\WINDOWS\system32\notepad.exe"" " & strObjectFileName
End If
Set oPackage = Nothing
oExportTool.TransferedObject = TRUE


' ----------------------------------
' -- Sample 2 :
' ----------------------------------
' Export all packages with No Propagation
' and the 'XML' Format
' -- Constants :
Dim strCollectionFileName
strCollectionFileName = "C:\TEMP\CollectionExport.XML"
oExportTool.Propagate = FALSE
oExportTool.Format    = "XML"
' --
Dim oPackageColl
Set oPackageColl = GetRoot.GetCollection("~h8rEkjZmo400[Paquetage]")
oExportTool.Export oPackageColl, strCollectionFileName
' Dispaly result file in Internet Explorer :
```

```
Dim oCollWShell
Set oCollWShell = CreateObject("WScript.Shell")
oCollWShell.Exec """c:\program files\internet explorer\iexplore.exe"" " & strCollectionFileName

Set oPackageColl = Nothing

' Display "FoundConfidential" is need :
If (oExportTool.FoundConfidential) Then
  Print "--> Confidential object is found."
End If


Set oExportTool = Nothing
```

The following text shows the output from the preceding code example:

```
Default Parameters :
- Perimeter            = ~Bav0cNnAjyQR[Standard for export]
- Format               = MGR
- Propagate            = True
- TransferedObject     = True
- ContinueOnConfidential = False
```

Implementation to JAVA

Constructor Detail

MegaExportTool

```
public MegaExportTool(MegaRoot mgRoot)
```

This method enables instancing of the ExportTool object.

Method Detail

propagate

`public boolean propagate()`
Allows you to specify if object roots must be propagated.

Returns:

- TRUE: To propagate; this is the default value.
- FALSE: To take object roots only.

---

propagate

`public void propagate(boolean bNewPropagate)`
Allows you to specify if object roots must be propagated.

Parameters:

`bNewPropagate` -
- TRUE: To propagate ; this is the default value.
- FALSE: To take object roots only.

---

transferedObject

`public boolean transferedObject()`
TransferedObject takes account of transfered objects.

Returns:

- TRUE: The transfered objects are part of object roots; this is the default value.
- FALSE: No transfered objects taken.

---

transferedObject

`public void transferedObject(boolean bNewTransferedObject)`
TransferedObject takes account of transfered objects.

Parameters:

`bNewTransferedObject` -
- TRUE: The transfered objects are part of object roots; this is the default value.
- FALSE: No transfered objects taken.

---

format

`public java.lang.String format()`
Format Choice of the format of an exchange of data.

Returns:

- MGR: ANSI text data exchange; this is the default value.
- XML: XML data exchange.

---

format

`public void format(java.lang.String NewFormat)`
Format Choice of the format of an exchange of data.

Parameters:

`NewFormat` -

- MGR: ANSI text data exchange; this is the default value.
- XML: XML data exchange.

---

perimeter

`public void` **`perimeter`**`(java.lang.Object oPerimeterID)`

Perimeter is used in Export Processing to extract sets of objects and links to generate the export file. The default perimeter is defined on MetaTool "Standard of Export".

Parameters:

`oPerimeterID` -

New perimeter is used in Export processing.

---

perimeter

`public java.lang.Object` **`perimeter`**`()`

Perimeter is used in Export Processing to extract sets of objects and links to generate the export file. The default perimeter is defined on MetaTool "Standard of Export".

Returns:

Perimeter is used in Export processing.

---

continueOnConfidential

`public boolean continueOnConfidential()`

Enables specification of whether export processing should be aborted if a confidential object is present in the collection of objects to be exported.

Returns:

- TRUE: Export processing is not aborted; this is the default value.
- FALSE: Export processing is aborted.

---

continueOnConfidential

`public void` **`continueOnConfidential`**`(boolean bContinueOnConfidential)`

Enables specification of whether export processing should be aborted if a confidential object is present in the collection of objects to be exported.

Parameters:

`bContinueOnConfidential` -

- TRUE: Export processing is not aborted; this is the default value.
- FALSE: Export processing is aborted.

---

foundConfidential

`public boolean` **`foundConfidential`**`()`

This attribute is specified by Export processing to indicate presence of a confidential object in object collection. You cannot modify this attribute.

Returns:

- TRUE: A confidential object is detected.

- FALSE: No confidential object is found.

Export

```
public void Export(MegaObject moObject,
                   java.lang.String strFileName)
```

Export an object in data exchange file.

Parameters:

`moObject` -

This object is root of Export processing.

`strFileName` -

This string is the name of the result file. It is the full name: local or network, directories, short name of the file and Extension.

Export

```
public void Export(MegaCollection mcObjects,
                   java.lang.String strFileName)
    final String strFileName = "C:\\TEMP\\MegaExport.Log";
```

Export a collection of objects in data exchange file.

Parameters:

`mcObjects` -

This collection of objects is root of Export processing.

`strFileName` -

This string is the name of the result file. It is the full name: local or network, directories, short name of the file and Extension.

Sample

```
// ----------------------------------
// -- ExportTool              V1.0
// ----------------------------------
// Samples of ExportTool object installation

// Creation of 'Component ExportTool' :

MegaExportTool oExportTool;
```

```
oExportTool = new MegaExportTool(mgRoot);

// Display parameters :
String szMes;
szMes = "Default Parameters : \r\n";
szMes = szMes + "Propagate : " + oExportTool.propagate() + "\r\n";
szMes = szMes + "Format : " + oExportTool.format() + "\r\n";
szMes = szMes + "Propagate : " + oExportTool.transferedObject() + "\r\n";
szMes = szMes + "continueOnConfidential : " + oExportTool.continueOnConfidential() + "\r\n";
Object oPerimeterID = oExportTool.perimeter();
szMes = szMes + "Perimeter : " +
mgRoot.getObjectFromID(oPerimeterID).getProp("~210000000900[Nom]") + "\r\n";

// ----------------------------------
// -- Sample 1 :
// ----------------------------------
// Export a package with no transfered object
// By Default : Propagation is Active and Format is 'MGR'
// -- Constants :
String strObjectFileName = "C:\\TEMP\\ObjectExport.MGR";
oExportTool.transferedObject(false);
// ----
MegaObject oPackage = mgRoot.getCollection("~h8rEkjZmo400[Paquetage]").item(1);

if ((oPackage != null) && (oPackage.exists())) {
  szMes = szMes + "Sample1 : Package(" + oPackage.getProp("~210000000900[Nom]") + ")\r\n";
  oExportTool.Export(oPackage, strObjectFileName);
  szMes = szMes + "--> Found Confidential Object : " + oExportTool.foundConfidential() + "\r\n";
}

// ----------------------------------
// -- Sample 2 :
// ----------------------------------
// Export all packages with no propagate
// By Default : Propagation is Active and Format is 'XML'
// -- Constants :
String strCollectionFileName = "C:\\TEMP\\CollectionExport.XML";
oExportTool.transferedObject(true);
oExportTool.propagate(false);
```

```
oExportTool.format("XML");
oExportTool.continueOnConfidential(true);
oExportTool.perimeter("~ICRBhWpi6DF0[Standard for export MEGA product (internal)]");

// ----
MegaCollection oPackageColl;
oPackageColl = mgRoot.getCollection("~h8rEkjZmo400[Paquetage]");

szMes = szMes + "Sample2 : All Packages\r\n";
szMes = szMes + "Default Parameters :\r\n";
szMes = szMes + "Propagate : " + oExportTool.propagate() + "\r\n";
szMes = szMes + "Format : " + oExportTool.format() + "\r\n";
szMes = szMes + "Propagate : " + oExportTool.transferedObject() + "\r\n";
szMes = szMes + "continueOnConfidential : " + oExportTool.continueOnConfidential() + "\r\n";
oPerimeterID = oExportTool.perimeter();
szMes = szMes + "Perimeter : " + mgRoot.getObjectFromID(oPerimeterID).getProp("~210000000900[Nom]") + "\r\n";

oExportTool.Export(oPackageColl, strCollectionFileName);

this.writeInFile(strFileName, szMes);
```

Miscellaneous

Sommaire

STEERING CALENDAR

A **Steering calendar** enables to define and launch reminders regarding actions to be performed at predefined due dates.

The following points are covered here:

INTRODUCTION

A **Steering calendar** includes steering dates, which enable to perform recurrent actions at predefined due date (absolute or relative).

☛ *See "Scheduler configuration", page 28*

A **Steering date** defines at which (relative or absolute) date the action is to be performed.

☛ *See "Steering date", page 29.*

An **Element with steering calendar** is an element associated with a steering calendar.

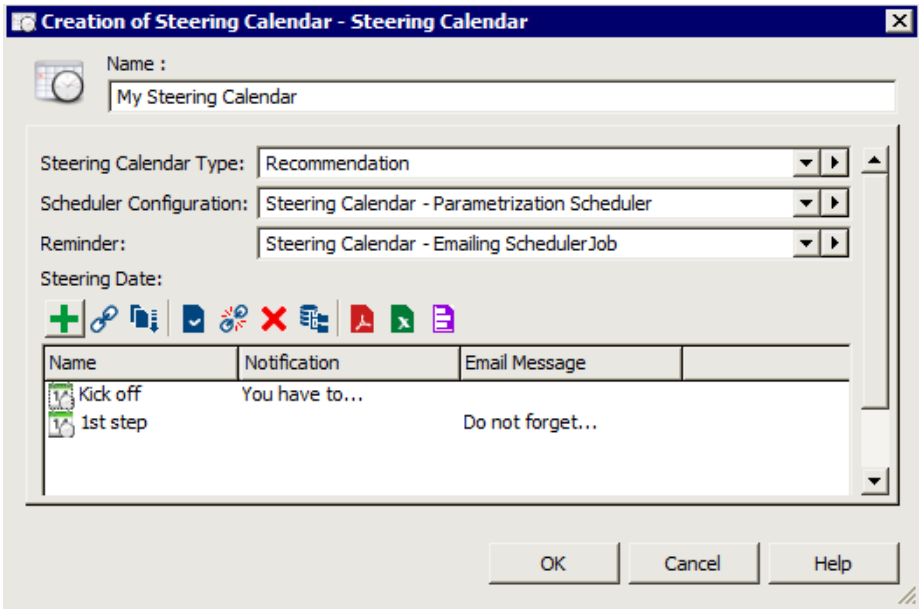
☛ *See "Steering calendar properties", page 27.*

The steering calendar is defined by the following **macros**:

- a parameterization macro, which defines the Scheduler parameterization. It includes the context: current object and steering date.
- a job macro, which defines the actions to be performed.

STEERING CALENDAR PROPERTIES

You cannot use steering calendars with GBMS repositories.



A steering calender includes the following parameters:

- "Steering calendar type", page 27
- "Scheduler configuration", page 28
- "Reminder", page 28
- "Steering date", page 29
- "Scheduling", page 30

Steering calendar type

The **Steering Calendar Type** determines the type of steering calendar. MEGA provide you with the following steering calendar types:

- action plan:
- control
- recommendation

You can create new steering calendar types.

See "Customizing a steering calendar", page 31.

Scheduler configuration

The **Scheduler Configuration** is described by a macro. This macro enables to retrieve:

- the current object
- the steering dates
- the scheduling
- the recording of triggers

MEGA delivers the following macros:

- "Steering Calendar - Parameterization Scheduler" (default value)
  Macro to be used for action plans or a recommendations.
- "Internal control - Execution Campaign Scheduler Parameterization"
  Macro to be used for controls.

You can create your own macro to customize your scheduler.

☛ *See "Customization", page 31.*

Reminder

The **Reminder** is described by a macro. This macro details what to do when the job is triggered

MEGA delivers the following macros:

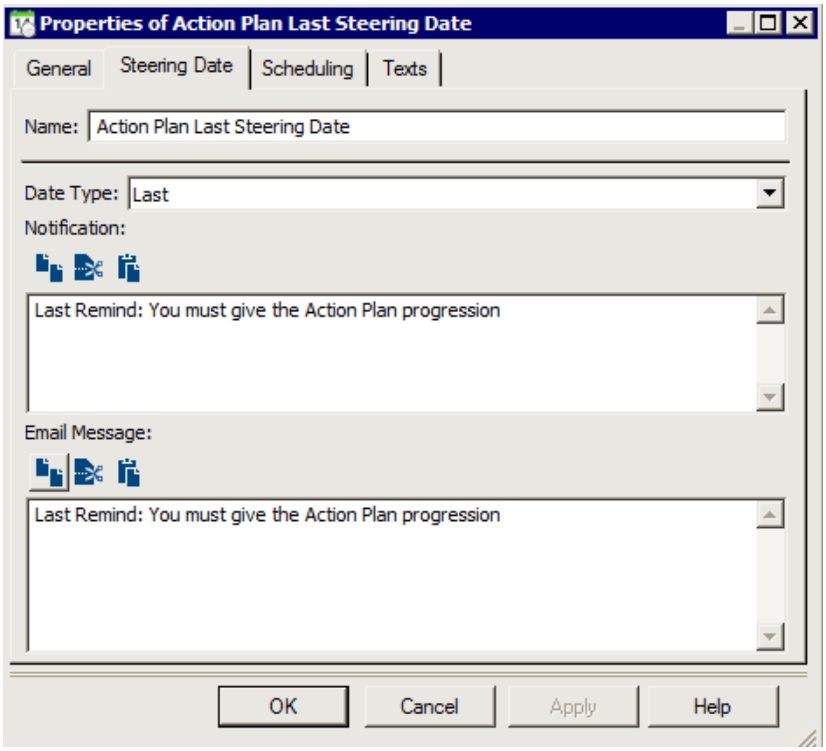
- "Steering Calendar - Emailing Scheduler Job" (default value)
  An email is sent to the person.

  ```
  Use case examples: action plans or recommendations.
  ```
- "Internal control - Execution Campaign Scheduler Job"

  ```
  Use case example: controls.
  ```

Steering date

A steering date enables the scheduler parameterization: specification of date repeat, start and end date. It also enables specification of the message and/or the notification to be sent (initial, remind or last date).



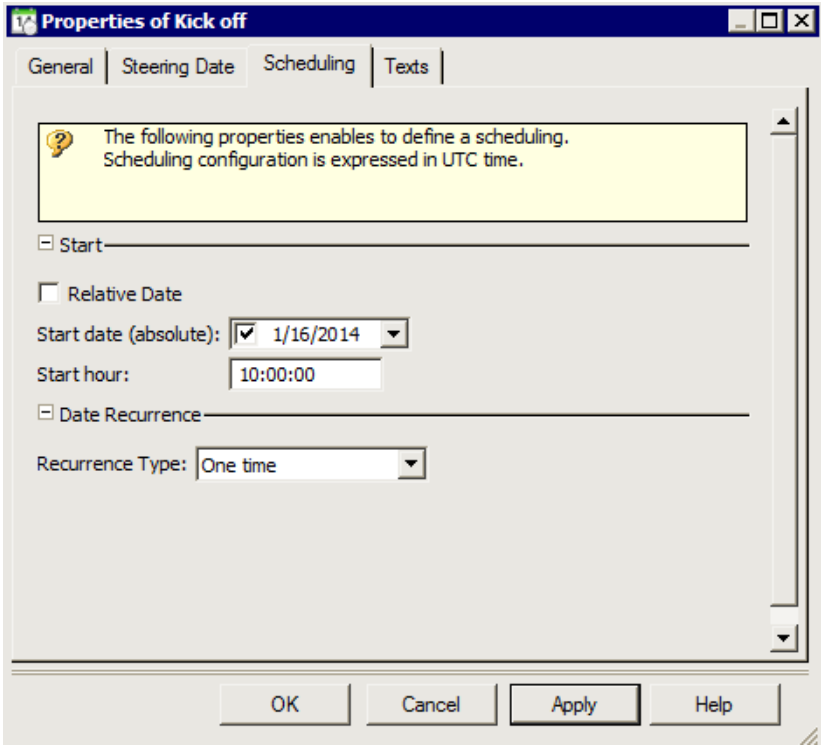
A steering date is defined by:

- its **Name**
- its type (**Date Type**):
  - "Initial": defines the start of the job (i.e.: action plan, control, etc.)
  - "Remind": enables to remind the person responsible for a job to complete the job or to report on the job status.
  - "Last": defines the end of the job (i.e.: action plan, control, etc.)
- a **Notification** (optional)
- an **Email Message** (optional)

Scheduling

For detailed information regarding the date configuration (e.g.: reference date, absolute date, date recurrence) see **HOPEX Studio - 2 - HOPEX Scheduler** guide.

☛ *Note that time is defined in UTC format, which means that daylight saving time is not considered.*



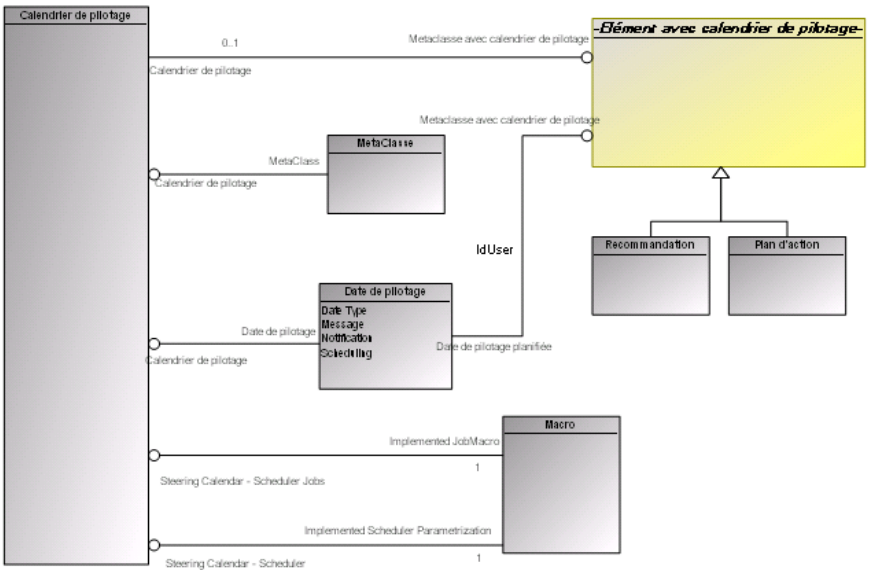
CUSTOMIZATION

Steering Calendar Metamodel

MEGA provides fully customized steering calendars for:

- action plans
- controls
- recommendations

The steering calendar Metamodel is as follows:



To use the steering calendar on other objects, you need to create a new steering calendar type, see "Customizing a steering calendar", page 31.

Customizing a steering calendar

To create a steering calendar type:

1. Link the object related MetaClass to the "Element With Steering Calendar" MetaClass, which creates a new steering calendar type.
2. Create the Scheduler Configuration macro.
   This macro includes the scheduler parameterization.

   The macro while executed retrieves the current object and the steering dates.

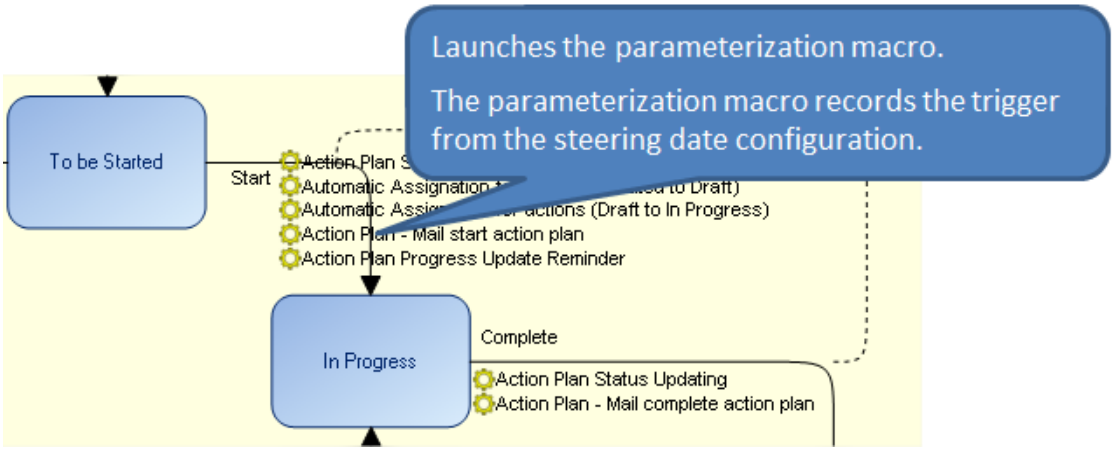
3. Create the job macro.
   This macro includes the job to be executed and the job trigger definition
4. Create the steering date(s).

☛ *See "Steering date", page 29.*

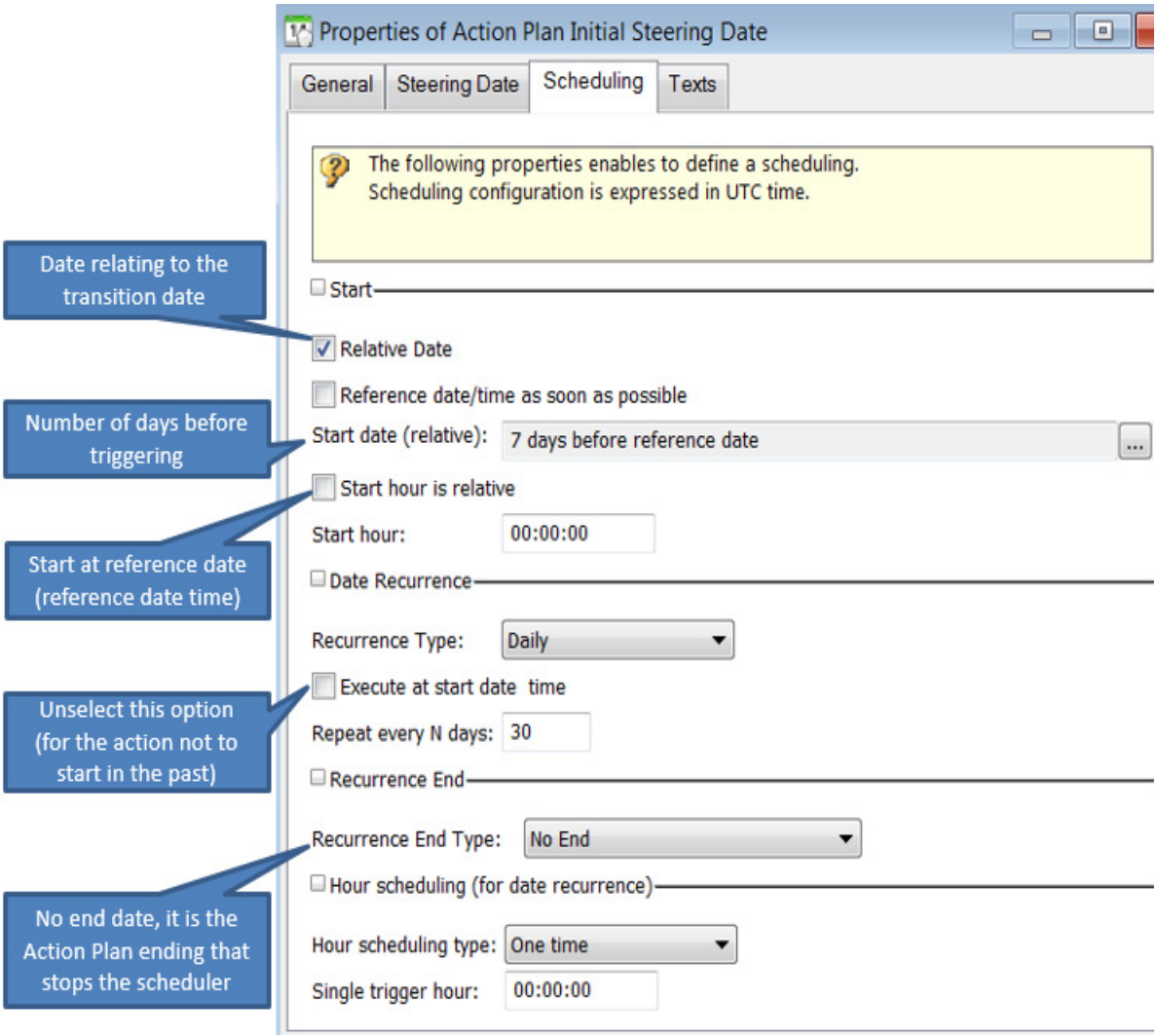
USE CASE: ACTION PLAN WORKFLOW

The following is an example of steering calendar use case with an Action Plan workflow.

1. The Workflow transition (i.e.: "To be be started" workflow status - "In progress" workflow status) launches the parameterization macro.



2. The parameterization macro records the trigger from the Steering date configuration (**Scheduling** tab).



3. The parameterization macro retrieves the current object and the steering dates.
   Parameterization macro: Steering Calendar - Parameterization Scheduler
   Signature: Sub SubscribeSchedulerJobs(currentObject As MegaObject)

```
parametrize a scheduler
 Dim job,date
   Set job = steering.getCollection("Implemented JobMacro").item(1)
   'get the job
   If job.Exists() Then
     ' get the steering dates
     For Each date In steering.getCollection("Steering Date")
       If (date.GetCollection("Scheduling").count>0) Then
         Set mgobjScheduling = date.GetCollection("Scheduling").Item(1)
         If ( (mgobjScheduling.getProp("Scheduling.IsRelative")="R") ) Then
           '---------- fix the begin date to now
           mgobjScheduling.getProp("Scheduling.Reference.DateTime")= DateAdd("s",2,now)
           ' SCheduler
           Set mgobjScheduledTrigger = objSchedulerClient.NewTrigger()
           mgobjScheduledTrigger.GetCollection("Scheduling").Add mgobjScheduling
           Set mgobjScheduledJob = mgobjScheduledTrigger.GetCollection("System Job").Item(1)
           mgobjScheduledJob.GetProp("Name") = "NotificationAndMessageSender"
           mgobjScheduledJob.GetProp("Implementation Macro") = job.megaField()
           ' fix the begin date
```

Reference date at workflow transition on the Action Plan

Job macro

4. The trigger launches the job macro.
   Job macro: Steering Calendar - Emailing SchedulerJob
   Signature: RunScheduledJob(mgobjJob As MegaObject, objJobResult As Object)

```
Function RunScheduledJob(mgobjJob As MegaObject, objJobResult As Object)
Dim currentObject                                   'action plan
Dim root As MegaRoot                                'get root
Dim context,schedulerContext                        'context callback of the scheduler
Dim assignment, personAssigned, personsystem        'objects to find an email user
Dim steeringDate                                    'object to get the text message and notification

Set root = mgobjJob.getRoot()
'get the context of the steering job
schedulerContext = mgobjJob.CallFunction("GetContextString")

Set context = ExtractContext(schedulerContext, root)
Set currentObject = context.Item(1)
Set steeringDate = context.Item(2)
If schedulerContext <> "" Then
  If DoJob(currentObject) Then
    set assignment = currentObject.getCollection("Person Assignment").item(1)
    If (assignment.Exists()) Then
      set personAssigned = assignment.getCollection("Assigned Person").item(1
      If (personAssigned.Exists()) Then
        set personsystem = personAssigned.getType("Person <System>")
        If (personsystem.Exists()) then
          If (steeringDate.GetProp("Email Message")<>"") Then
          SendEmail steeringDate, personsystem, root
          End If
```

Retrieving the context

Finding the Action Plan assigned person

Sending email function

Scheduler

1 INTRODUCTION

1.1 Aim of this document

The aim of this document is to describe the Scheduler component embedded into HOPEX platform:

- what it is used for
- how to configure it
- how to add scheduled jobs

1.2 Presentation

The Scheduler component enables to execute a macro at a given date and time.

Example: the following features rely on the Scheduler possibilities:

- Steering Calendars
- Alignment: Automatic transfers
- Scheduled Actions in Workflows
- Reminders
- Scheduled transitions
- Assessment Campaign

Macro execution scheduling combinations are the following:

- Trigger at a date and time
- Recurrent triggering (daily, weekly, monthly)
- Trigger relative to another reference date

1.3 Requirements

The Scheduler component meets the following requirements:

- Repository requirement: RDBMS repository
- SOA deployment (see **Error! Reference source not found.** section)
  - Scheduler is a web service hosted into the MOS, it requires the MOS to be deployed and configured
  - MOS can be deployed either with Windows front end or HOPEX Web front end, so that Scheduler can be used in a full Windows front end environment, in a HOPEX Web front end environment, or in a mixed environment
- Availability: The scheduler is available with TeamWork product and other products containing features relying on the Scheduler like MEGA Alignment and MEGA Assessment. For an exhaustive list of products in which the Scheduler is available, see each specific product description documentation.

1.4 Limitation

Due to date/time internal storage in HOPEX application, the Scheduler does not currently manage date/time after year 2038.

This limitation will be overridden in future versions.

1.5 Architecture

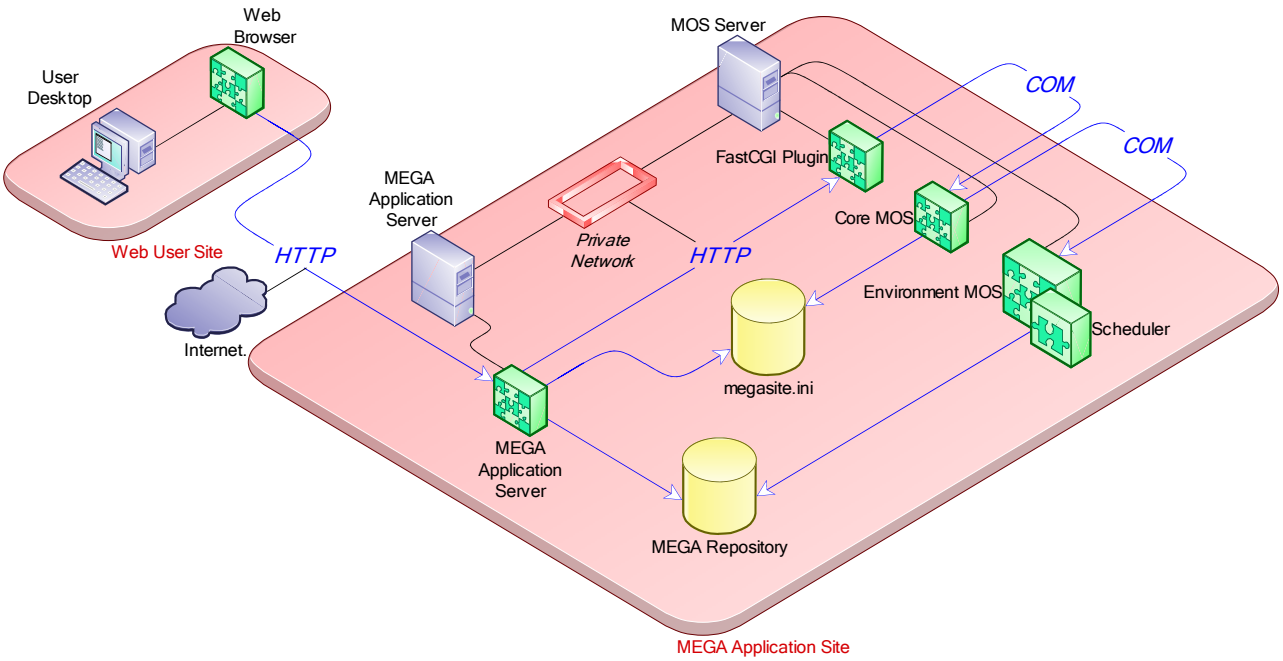
The Scheduler component is hosted into the MOS.

There is one Scheduler instance per environment.

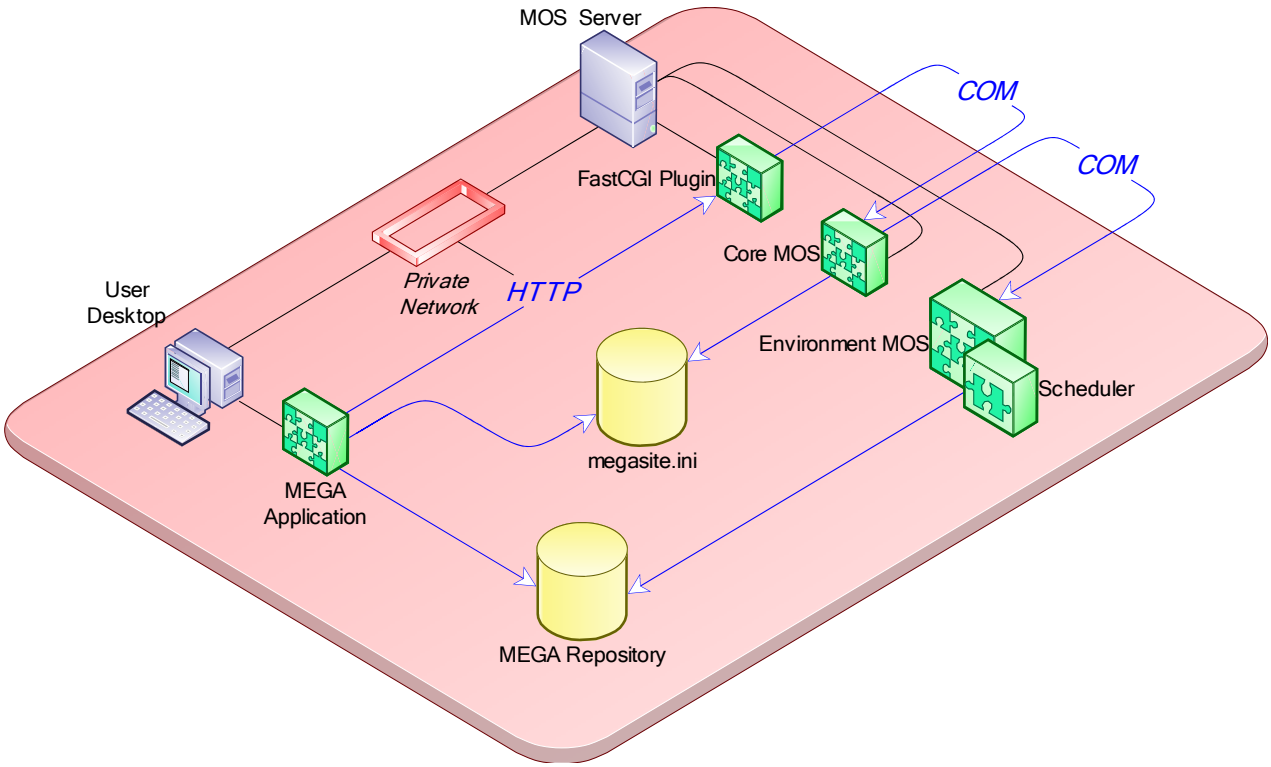
The MEGA Application client requires an HTTP connection to the Core MOS. The Core MOS application delegates the Scheduler solicitations to the matching Environment MOS.

Scheduler clients have access to the Scheduler using a specific API (see Scheduler Client API section p. 11).

1.5.1 Web Front-End architecture

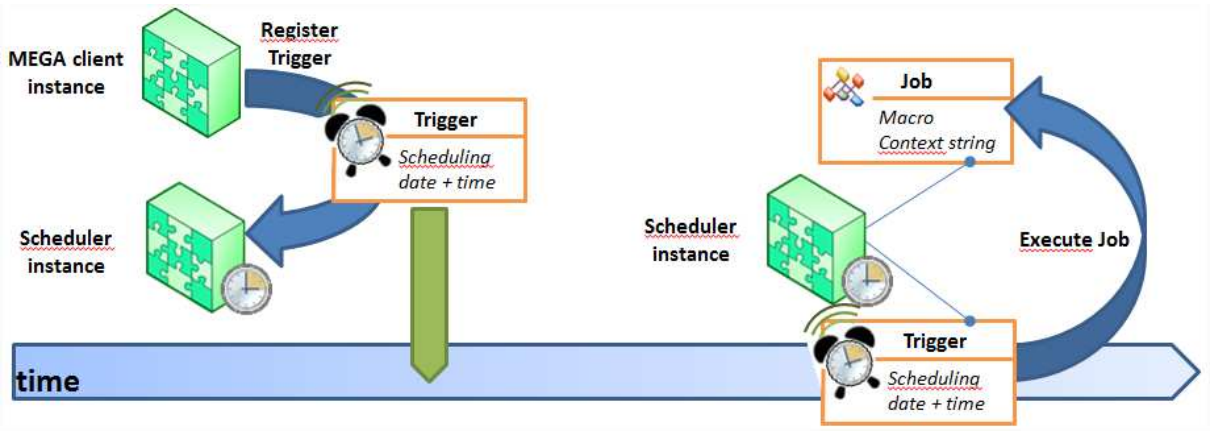


1.5.1 Windows Front-End architecture



2 SCHEDULER PRINCIPLES

2.1 Concepts



2.1.1 Job

The Job is the item identifying the Macro to be executed. It includes a "Context" string used to pass needed information at Macro execution.

2.1.2 Trigger

A Trigger is associated to the Job to define when to execute this Job.

The Scheduling enables to define when (date & time) to execute the Job and which frequency to apply if any.

2.1.3 Scheduler

The Scheduler component provides the following services:

- Add a Trigger (same as register a Trigger)
- Delete a Trigger
- Find existing Triggers
- Modify a Trigger
- Execute Jobs associated to Triggers at the date and time specified in the Scheduling

2.2 Scheduler execution details

2.2.1 Persistance

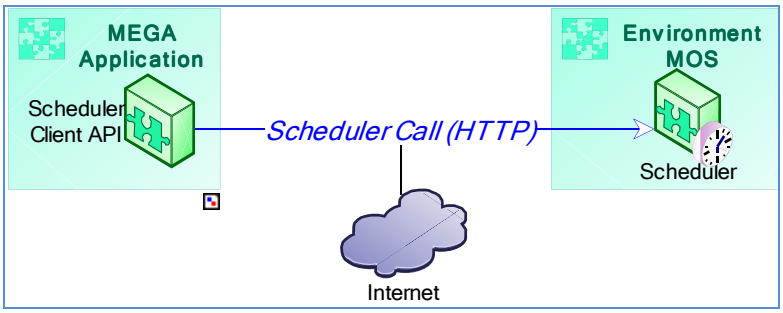
The Scheduler component instantiated into the Environment process loads and manage the Triggers into its memory.

Triggers and Jobs are replicated into the Repository. In case the process hosting the Scheduler dies, all Triggers and Jobs are recovered.

Consistency of the Triggers and Jobs are managed by the Scheduler component and Scheduler Client API. It is forbidden to handle Triggers and Jobs repository objects.

2.2.2 Scheduler API

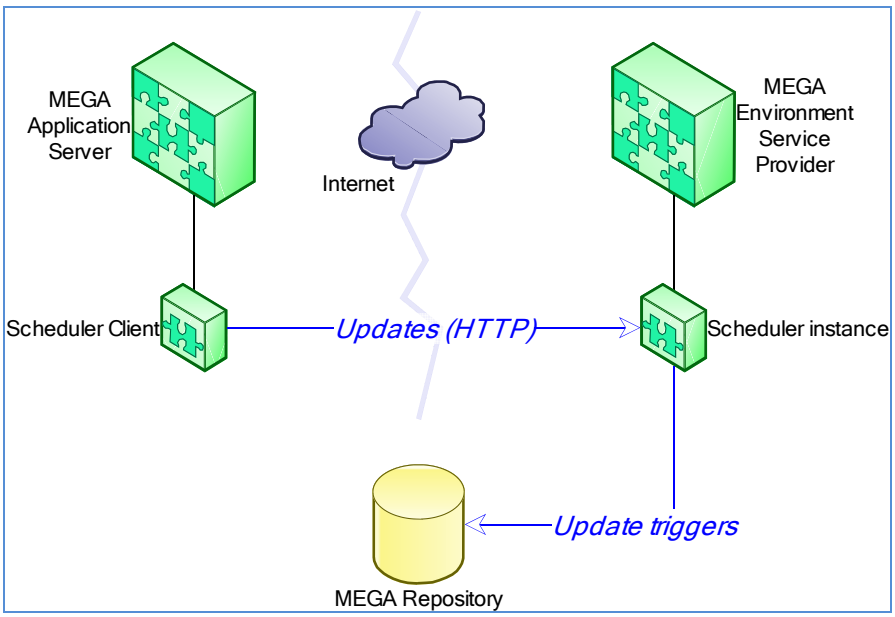
A dedicated API is provided to access Scheduler services. This API is called the Scheduler Client API (see Scheduler Client API section p.11).



You must use the Scheduler Client API for any operation (Add, Remove, Update, Find) performed with the Scheduler. **This API guaranties Triggers and Jobs consistency.**

2.2.3 Scheduler and Workspaces

All operations on the Scheduler are done by default via an HTTP call so that they are executed as soon as the HTTP call is received and processed. The operation is executed without any dependency with the current opened Workspace in the MEGA Application process.



1 WebService mode

Several Scheduler operations update Triggers of are registered into a Scheduler instance:

- Add a Trigger
- Delete a Trigger
- Modify a Trigger

Update mode

If the Scheduler operation depends on changes made into the current Workspace and this operation is executed outside the current Workspace, this could lead to inconsistencies.

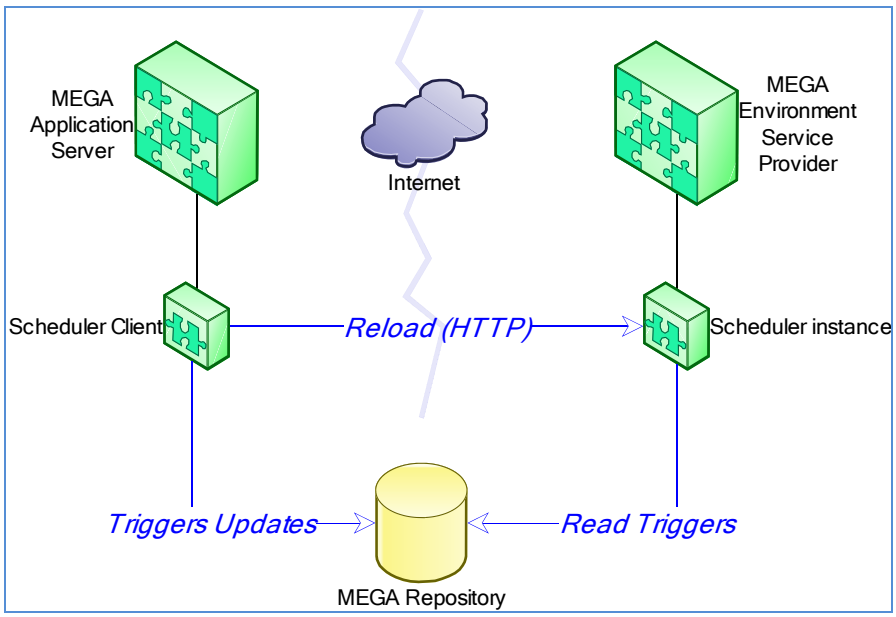
Example:

In a Workspace, an object and a Scheduled Job to process something later on that object are created. If the Workspace is discarded and the Trigger is registered, when the matching Job is executed, it can generate errors, or worse, apply inconsistent updates.

Specify the "*update modes*" for the Scheduler update operations:

- The "Web Service" mode
- The Transactional mode

A Scheduler operation executed in Transactional mode is effective into the Scheduler only once the work in the Workspace has been dispatched. If the work is discarded, the Scheduler operation is also discarded.



2 Tansactional mode

2.2.4 Job execution: user/profile/repository

All operations into a MEGA process require a connection (Session & private Workspace):

- A connection is necessary prior to Job execution.
- A Scheduler instance is hosted into an Environment MOS, meaning it is global to an Environment.
- The Job connection implies to know which user connects, to which repository, with which profile.

For security reasons, the user used to execute a Job is always the one who added (registered) the matching Trigger from the MEGA Application Client into the Scheduler instance. Profile and Repository can be specified as parameters of the AddTrigger operation. By default Profile and Repository are the ones specified at the Trigger registering.

3 SCHEDULER CONFIGURATION

Scheduler requires a properly configured MOS. One Scheduler instance is automatically instantiated into each Environment MOS.

3.1 MEGA installations without Scheduler

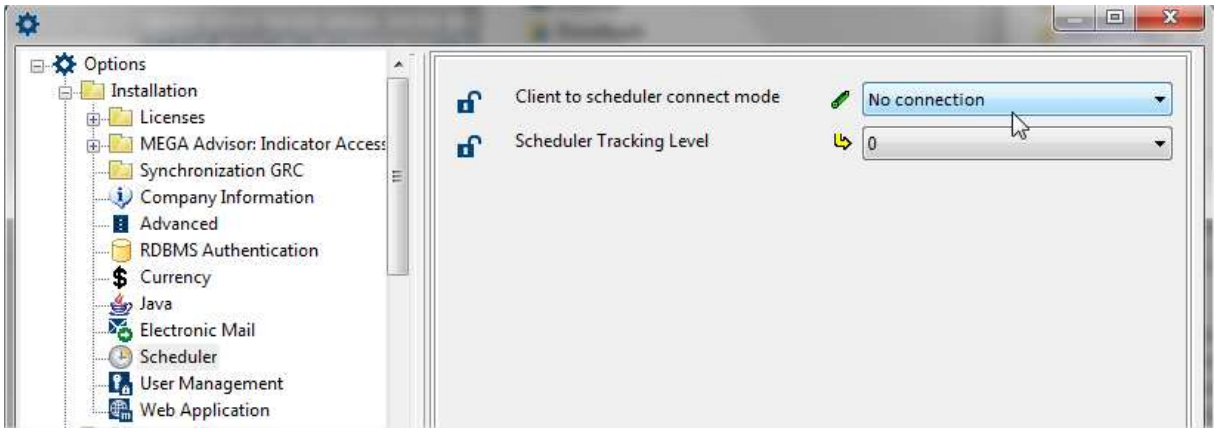
Some standard features (as provided standard Workflow Definitions) require a Scheduler instance. But Scheduler needs MOS and RDBMS repository.

For features relying on Scheduler to be executed, an option enables to deactivate the Scheduler so that all Scheduler operations calls are ignored.

To deactivate dependencies with Scheduler:

1. Go to MEGA Options: **Installation > Scheduler.**

2. In the right pane, set the **Client to scheduler connect mode** option value to "No connection".

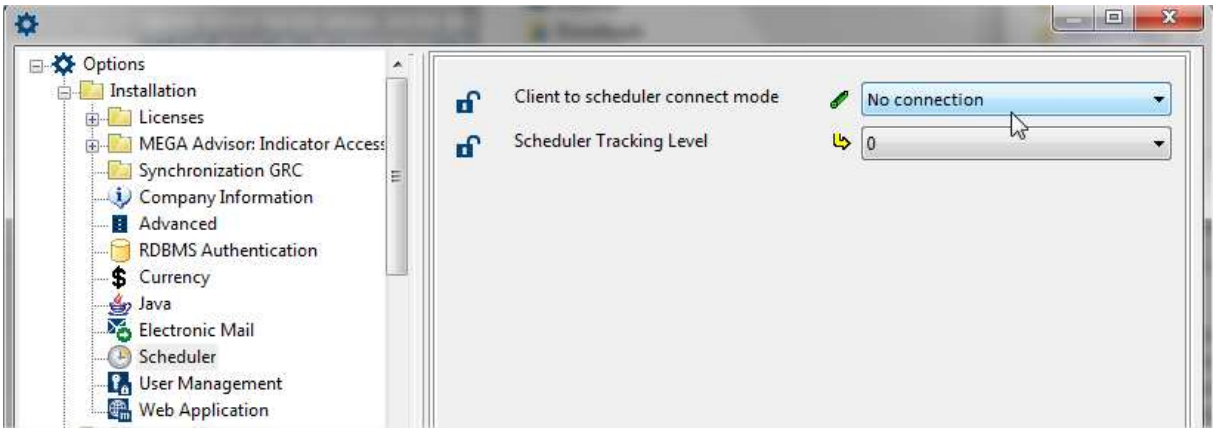


3.2 Traceability

The **Scheduler Tracking Level** option enables to log the Scheduler actions into MEGA error log.

To follow up Scheduler activity:

1. Go to MEGA Options: **Installation > Scheduler**.
2. In the right pane, select the **Scheduler Tracking Level** option value.



The tracking levels are:

- 0: no information logged
- 1: global scheduler actions
  - Scheduler start
  - Scheduler stop
  - Scheduler Triggers reload from repository
- 2: updates and all client side calls
  - Add Trigger
  - Remove Trigger
  - Update Trigger
  - Get Trigger
  - All client side calls
- 3 : job executions
  - Job triggering
  - Job execution

4 SCHEDULER CLIENT API

All features relying on the Scheduler **must** use the **SchedulerClient** API to add, remove, update or get Triggers to or from the Scheduler instance.

Direct accesses into the repository to System Triggers, System Jobs are forbidden.

4.1 Detailed description of the SchedulerClient API

The **SchedulerClient** API is detailed into the JavaDoc provided with MEGA installation.

See "*MegaSchedulerClient*" into the "***mj_api.doc.zip***" documentation package stored in the "***java\doc***" folder of MEGA installation site.

4.2 Java

4.2.1 Documentation

See "*MegaSchedulerClient*" into "***mj_api.doc.zip***" documentation package stored in the "***java\doc***" folder of MEGA installation site.

4.3 VB Script

4.3.1 Documentation

To get access to the **SchedulerClient** API, use the ***SchedulerClient*** Method available on MegaRoot.

Example:

```
Set objSchedulerClient = GetRoot().InvokeFunction("~nNmUTwNTF94K[SchedulerClient]")
```

Members available on the **SchedulerClient** object are defined below (for detailed description see JavaDoc documentation):

```
Class SchedulerClient
  Public Property Get State() As String
  Public Property Get ConnectionMode() As Integer
  Public Property Let ConnectionMode(Integer intConnectionMode)
  Public Property Get DefaultUpdateMode() As Integer
  Public Property Let DefaultUpdateMode(Integer intDefaultUpdateMode)
  Public Sub StartScheduler()
  Public Sub StopScheduler()
  Public Sub ReloadTriggers()
  Public Function NewTrigger() As MegaObject
  Public Sub AddTrigger(mgobjTrigger As MegaObject, strOptions As String)
  Public Sub RemoveTrigger(mgobjTriggerId As Object, strOptions As String)
  Public Sub UpdateTrigger(mgobjTrigger As MegaObject, strOptions As String)
```

```
  Public Function GetTrigger(mgidTriggerId As Object, strOptions As String) As MegaObject
  Public Function GetAllTriggers() As MegaCollection
End Class
```

4.3.2 Examples

Registering a Trigger

```
Dim mgRoot
Dim objSchedulerClient
Dim mgobjScheduling
Dim mgobjScheduledTrigger
Dim mgobjScheduledJob

Set mgRoot = GetRoot()

' Get the SchedulerClient API object
Set objSchedulerClient = mgRoot.SchedulerClient

' Allocate a "System Trigger" MegaObject to be configured
Set mgobjScheduledTrigger = objSchedulerClient.NewTrigger()
' Get the Scheduling and "System Job" MegaObject to be configured
Set mgobjScheduling =
mgobjScheduledTrigger.GetCollection("~e)PMRueCFbLR[Scheduling]").Item(1)
Set mgobjScheduledJob = mgobjScheduledTrigger.GetCollection("~pWuUJAATF5mD[System
Job]").Item(1)

' Configure the Scheduling
mgobjScheduling.SetProp "~azPMryeCFfRR[Scheduling.IsRelative]", "A"
mgobjScheduling.SetProp "~h)PMS5fCFTXR[Scheduling.RepeatKind]", "S"
' Set the start at Now + 30 seconds
mgobjScheduling.SetProp "~CaW0Vl0PG1pC[Scheduling.Start.AbsDateTime]",
DateAdd("s",30,Now)

' Configure the System Job (with dumy macro and contextstring, just for example)
mgobjScheduledJob.GetProp("~210000000900[Name]") = "My Job"
mgobjScheduledJob.GetProp("~MXuU7x9TFLlD[Implementation Macro]") = "~oHDN0jc0I1p4[My
Macro]"
mgobjScheduledJob.CallMethod "~zdj0fUG4GXRM[SetContextString]", "my informations"

' Register the System Trigger into the Scheduler instance
objSchedulerClient.AddTrigger mgobjScheduledTrigger, "UPDATEMODE=TRANSACTIONAL"
```

Different ways to setup the Scheduling

- Configure the Scheduling directly using the Scheduling MegaObject properties

```
' Get the Scheduling MegaObject to be configured
Set mgobjScheduling =
mgobjScheduledTrigger.GetCollection("~e)PMRueCFbLR[Scheduling]").Item(1)
' Set the Scheduling date as absolute
mgobjScheduling.SetProp "~azPMryeCFfRR[Scheduling.IsRelative]", "A"
' Set the Scheduling as single start
mgobjScheduling.SetProp "~h)PMS5fCFTXR[Scheduling.RepeatKind]", "S"
' Set the start at Now + 30 seconds
mgobjScheduling.SetProp "~CaW0Vl0PG1pC[Scheduling.Start.AbsDateTime]",
DateAdd("s",30,Now)
```

- Setup the Scheduling from a Scheduling definition stored in an XML string

```
' Get the Scheduling MegaObject to be configured
Set mgobjScheduling =
mgobjScheduledTrigger.GetCollection("~e)PMRueCFbLR[Scheduling]").Item(1)
' Setup the Scheduling from Xml string
mgobjScheduling.CallMethod "~ sJYouhVuFP7S[UpdateFromString]", strXmlSchedulingDefinition
```

- Recover the scheduling from a repository object on which it is defined

The repository object must have a "*~iUhj4likEzJQ[Scheduling]*" MetaAttribute. The Scheduling definition is stored into the standard "*~iUhj4likEzJQ[Scheduling]*" MetaAttribute.

In this example, the Scheduling definition is set as relative. Only the reference date and time have to be set.

```
' Get the repository object on which the Scheduling is defined
Set mgobjSteeringDate = mgobjSteeringCalendar.GetCollection("~qcuD3s3SFXL7[Steering
Date]")
' Get the Scheduling MegaObject to be configured
Set mgobjScheduling =
mgobjSteeringDate.GetCollection("~e)PMRueCFbLR[Scheduling]").Item(1)
' Set the reference date and time (VBScript Date type)
mgobjScheduling.getProp("~V(PMgweCF5OR[Scheduling.Reference.DateTime]")=
dtReferenceDateTime
```

5 SCHEDULING

The Scheduling must be defined when adding (registering) a Trigger into the Scheduler. Scheduling information must be passed as an XML string.

Abstract layers enable to:

- handle the Scheduling information as a MEGA object. In this form, the Scheduling information is the set of properties of this MEGA object and this enables easy update via MegaObject API.
- give access - for reading and updating - to the Scheduling information into reusable MetaPropertyPages.

5.1 Scheduling information: XML Scheduling format

Date and time are interpreted as UTC date and time.

```
<?xml version="1.0" encoding="utf-8"?>
<scheduling>
  <reference                          (defined for relative scheduling)
    date="yyyy/mm/dd"
    hour="hh:mm:ss"
    />
  <start
    date="yyyy/mm/dd"                 (defined for absolute scheduling)
    hour="(A/R)hh:mm:ss"              (A=> absolute time, R=> relative: adds to current
time)
    executeatstart="y/n"              (y: launches the trigger at defined start date and
time
                                       n: only launches at recurrent date and time)
    >
    <relativedate                     (defined for relative scheduling)
      daysfromreference="(-)1"        (cannot be used with "dayofweek" or "dayofmonth")
      dayofweek="Su"                  (cannot be used with "daysfromreference" or
"dayofmonth")
      dayofmonth="01"                 (cannot be used with "daysfromreference" or
"dayofweek")
      weeksfromreference="(-)1"       (in case "dayofweek")
      weekofmonth="1"                 (in case "dayofweek")
      monthsfromreference="(-)1"      (in case "dayofmonth" or "weekofmonth")
      month="Jan"                     (in case "dayofmonth" or "weekofmonth")
      />
  </start>
  <dailyrepeat                        (cannot be used with <weeklyrepeat> or
<monthlyrepeat>)
    daysperiod="1"
    >
    <endrepeat
      noend="y/n"                     (cannot be used with "repeatnumber" or "date" and
"time")
      repeatnumber="1"                (cannot be used with "noend" or "date" and "time")
      date="yyyy/mm/dd"               (cannot be used with "repeatnumber" or "noend")
      hour="(A/R)hh:mm:ss"
      >
      <relativedate                   (defined for relative scheduling)
        daysfromreference="(-)1"      (cannot be used with "dayofweek" or "dayofmonth")
        dayofweek="Su"                (cannot be used with "daysfromreference" or
"dayofmonth")
```

```
        dayofmonth="01"             (cannot be used with "daysfromreference" or
"dayofweek")
        weeksfromreference="(-)1"   (in case "dayofweek")
        weekofmonth="1"             (in case "dayofweek")
        monthsfromreference="(-)1"  (in case "dayofmonth" or "weekofmonth")
        month="Jan"                 (in case "dayofmonth" or "weekofmonth")
        />
    </endrepeat>
    <timescheduling
      singlestart="hh:mm:ss"        (cannot be used with "begin", "end" and "repeat")
      begin="hh:mm:ss"              (cannot be used with "singlestart")
      end="hh:mm:ss"                (cannot be used with "singlestart")
      repeat="hh:mm:ss"             (cannot be used with "singlestart")
      />
  </dailyrepeat>
  <weeklyrepeat                     (cannot be used with <dailyrepeat> or
<monthlyrepeat>)
    weeksperiod="1"
    daysofweek="Su,Mo,Tu,We,Th,Fr,Sa"
    >
    <endrepeat
      noend="y/n"                   (cannot be used with "repeatnumber" or "date" and
"time")
      repeatnumber="1"              (cannot be used with "noend" or "date" and "time")
      date="yyyy/mm/dd"             (cannot be used with "repeatnumber" or "noend")
      hour="(A/R)hh:mm:ss"
      >
      <relativedate                 (defined for relative scheduling)
        daysfromreference="(-)1"    (cannot be used with "dayofweek" or "dayofmonth")
        dayofweek="Su"              (cannot be used with "daysfromreference" or
"dayofmonth")
        dayofmonth="01"             (cannot be used with "daysfromreference" or
"dayofweek")
        weeksfromreference="(-)1"   (in case "dayofweek")
        weekofmonth="1"             (in case "dayofweek")
        monthsfromreference="(-)1"  (in case "dayofmonth" or "weekofmonth")
        month="Jan"                 (in case "dayofmonth" or "weekofmonth")
        />
    </endrepeat>
    <timescheduling
      singlestart="hh:mm:ss"        (cannot be used with "begin", "end" and "repeat")
      begin="hh:mm:ss"              (cannot be used with "singlestart")
      end="hh:mm:ss"                (cannot be used with "singlestart")
      repeat="hh:mm:ss"             (cannot be used with "singlestart")
      />
  </weeklyrepeat>
  <monthlyrepeat                    (cannot be used with <dailyrepeat> or
<weeklyrepeat>)
    monthsperiod="1"                (cannot be used with "months")
    months="Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec" (cannot be used with
"monthsperiod")
    daysofmonth="01,02,03,04,05,...,28,29,30,31,L"  (cannot be used with "weeksofmonth")
    weeksofmonth="1,2,3,4,L"                        (cannot be used with "daysofmonth")
    daysofweek="Su,Mo,Tu,We,Th,Fr,Sa"               (cannot be used with "daysofmonth")
    >
    <endrepeat
      noend="y/n"                   (cannot be used with "repeatnumber" or "date" and
"time")
      repeatnumber="1"              (cannot be used with "noend" or "date" and "time")
      date="yyyy/mm/dd"             (cannot be used with "repeatnumber" or "noend")
      hour="(A/R)hh:mm:ss"
      >
      <relativedate                 (defined for relative scheduling)
        daysfromreference="(-)1"    (cannot be used with "dayofweek" or "dayofmonth")
        dayofweek="Su"              (cannot be used with "daysfromreference" or
"dayofmonth")
```

```
        dayofmonth="01"             (cannot be used with "daysfromreference" or
"dayofweek")
        weeksfromreference="(-)1"   (in case "dayofweek")
        weekofmonth="1"             (in case "dayofweek")
        monthsfromreference="(-)1"  (in case "dayofmonth" or "weekofmonth")
        month="Jan"                 (in case "dayofmonth" or "weekofmonth")
        />
    </endrepeat>
    <timescheduling
      singlestart="hh:mm:ss"        (cannot be used with "begin", "end" and "repeat")
      begin="hh:mm:ss"              (cannot be used with "singlestart")
      end="hh:mm:ss"                (cannot be used with "singlestart")
      repeat="hh:mm:ss"             (cannot be used with "singlestart")
      />
  </monthlyrepeat>
</scheduling>
```

5.1.1 <reference>

Used in case the Scheduling is relative. Defines the date and time in UTC which is used as reference for relative start date and time and relative endrepeat date and time.

5.1.2 <start>

Defines the date and time at which the Scheduling starts.

If "executeatstart" is:

- "y", the Trigger launches the Job at this date and time.
- "n", the Trigger waits for the next recurrent date and time to launch the Job (see <dailyrepeat>, <weeklyrepeat>, <monthlyrepeat>).

In case the scheduling is:

- relative (see <reference>), the date is defined by the <relativedate> structure and the time "attribute" can be relative (example time="R00:02:00", means +2minutes).
- absolute, the date is defined by the "date" attribute.

5.1.3 <relativedate>

Defines a relative date. This structure is used into <start> and <endrepeat>.

The relative date refers to the <reference> date and time.

5.1.4 "daysfromreference"

Defines a number of days applied to the reference date, if:

- negative: number of days before
- positive: number of days after the reference date.

5.1.5 "dayofweek"

Defines a day of the week at which the Trigger launches the Job.

The value is one of the two characters items: Su=Sunday, Mo=Monday, Tu=Tuesday, We=Wenesday, Th=Thursday, Fr=Friday, Sa=Saturday.

"dayofweek" must be used with "weeksfromreference" or "weekofmonth".

5.1.6 " weeksfromreference "

Defines the number of weeks for the reference date, if:

- zero: the Trigger launches the Job the day of week defined by "dayofweek" in the next 7 days.
- positive: the Trigger launches the Job the day of week defined by "dayofweek" **after** the number of weeks defined
- negative: the Trigger launches the Job the day of week defined by "dayofweek" **before** the number of weeks defined

5.1.7 "weekofmonth"

Defines a week in the month at which the Trigger launches the Job.

The value is one of the following values 1=1st week of the month, 2=2nd week of the month, 3=3rd week of the month, 4=4th week of the month, L=last week of the month (4th or 5th).

"weekofmonth " must be used with "monthsfromreference" or "monthofyear".

5.1.8 "dayofmonth"

Defines a day of the month at which the Trigger launches the Job.

The value can be a number between 1 and 31 or L=last day of month.

"dayofmonth" must be used with "monthsfromreference" or "month".

5.1.9 "monthsfromreference"

Defines the number of months for the reference date, if:

- zero: the Trigger launches the Job the day defined by "dayofmonth" in the next 31 days or the day of the week defined by "dayofweek" and "weekofmonth".
- positive: the Trigger launches the Job the day defined by "dayofmonth" in the next 31 days or the day of the week defined by "dayofweek" and "weekofmonth" **after** the number of weeks defined.
- negative: the Trigger launches the Job the day defined by "dayofmonth" in the next 31 days or the day of the week defined by "dayofweek" and "weekofmonth" **before** the number of weeks defined.

5.1.10 "month"

Defines a month in the year at which the Trigger launches the Job.

The value is one of the three characters items: Jan=January, Feb=February, Mar=March, Apr=April, May=May, Jun=June, Jul=July, Aug=August, Sep=September, Oct=October, Nov=November, Dec=December.

"month" must be used with "dayofmonth" or "weekofmonth".

5.1.11 <endrepeat>

Defines the end of a recurrent scheduling defined by <dailyrepeat>, <weeklyrepeat> or <monthlyrepeat>.

One and only one of the following item can be defined into the <endrepeat> tag:

- "noend": the recurrence never stops
- "repeatnumber": number of time the recurrence occurs (the <start> date and time is not included into this number)
- "date" and "time": absolute date and time
- <relativedate> and "time":
  - date relative to <reference> date
  - time relative to <reference> time if "time" value starts with "(R)"
  - time absolute if "time" value starts with "(A)" or nothing

5.1.12 <timescheduling>

Can only be used into <dailyrepeat>, <weeklyrepeat>, or <monthlyrepeat>. Used to define the hours at which the trigger launches the Job in the recurrent days. Use "singlestart" or the set of attributes "begin", "end", and "repeat".

If used with:

- "singlestart", the Job is launched only once in the day.
- "begin", "end", and "repeat" the Job is launched every "repeat" time between "begin" and "end".

5.1.13 <dailyrepeat>

Used to define a day-based recurrence. The recurrence occurs every "daysperiod" days.

5.1.14 <weeklyrepeat>

Used to define a recurrence based on weeks.

The recurrence occurs every "weeksperiod" weeks on the days defined by "daysofweek". "daysofweek" is a coma separated list of values in the following two characters items : Su=Sunday, Mo=Monday, Tu=Tuesday, We=Wenesday, Th=Thursday, Fr=Friday, Sa=Saturday.

5.1.15 <monthlyrepeat>

Used to define a recurrence based on months. The recurrence occurs every "monthsperiod" months or on "months" of the year.

"months" is a coma separated list of values in the following three characters items : Jan=January, Feb=February, Mar=March, Apr=April, May=May, Jun=June, Jul=July, Aug=August, Sep=September, Oct=October, Nov=November, Dec=December.

The recurrence can occur days of the months using "daysofmonth" or days of week using "weeksofmonth" combined with "daysofweek".

"daysofmonth" is a coma separated list of values that can be a number between 1 and 31 or L=last day of month.

"weeksofmonth" is a coma separated list of values in the following values 1=1st week of the month, 2=2nd week of the month, 3=3rd week of the month, 4=4th week of the month, L=last week of the month (4th or 5th).

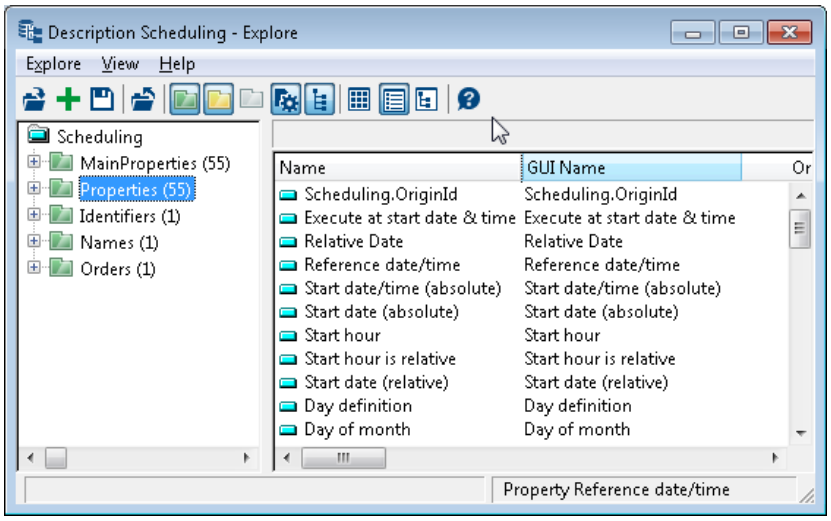
"daysofweek" is a coma separated list of values in the following two characters items : Su=Sunday, Mo=Monday, Tu=Tuesday, We=Wenesday, Th=Thursday, Fr=Friday, Sa=Saturday.

5.2 Scheduling MegaObject interface

The properties of the Scheduling MegaObject available via the Scheduler client API can be listed in the MEGA Explorer as following:

```
Dim mgRoot
Dim objSchedulerClient
Dim mgobjScheduledTrigger
Dim mgobjScheduling
Dim mgobjSchedulingType

Set mgRoot = GetRoot()

' Get the SchedulerClient API object
Set objSchedulerClient = mgRoot.SchedulerClient

' Allocate a "System Trigger" MegaObject
Set mgobjScheduledTrigger = objSchedulerClient.NewTrigger()

' Get the Scheduling MegaObject
Set mgobjScheduling =
mgobjScheduledTrigger.GetCollection("~e)PMRueCFbLR[Scheduling]").Item(1)

' Get the Scheduling ObjectType MegaObject
Set mgobjSchedulingType = mgobjScheduling.GetTypeObject()

' Explores the Scheduling ObjectType
mgobjSchedulingType.Explore
```

The result is an Explorer window as bellow:



All the properties match information elements detailed in "**Error! Reference source not found.**".

A description is available in the comment of each property.

5.3 Scheduling Property Page

5.3.1 Presentation of the Scheduling Property Page

Scheduling configuration on a MetaClass on which a scheduling has to be defined is stored into a text property of the MetaClass.

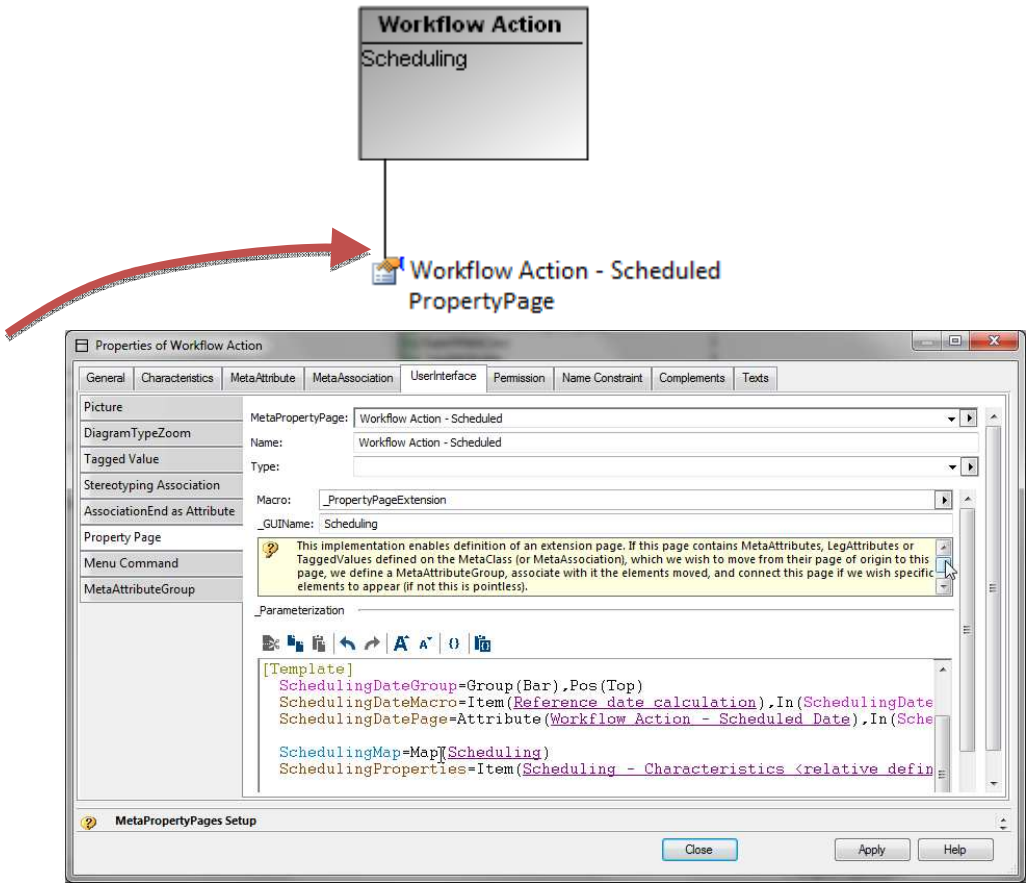
The scheduling property page is a graphical layer over the scheduling XML text format that enables easy edition/display of scheduling configurations.

This property page can be integrated in property pages of custom MetaClasses.

Example: scheduled Workflow Action

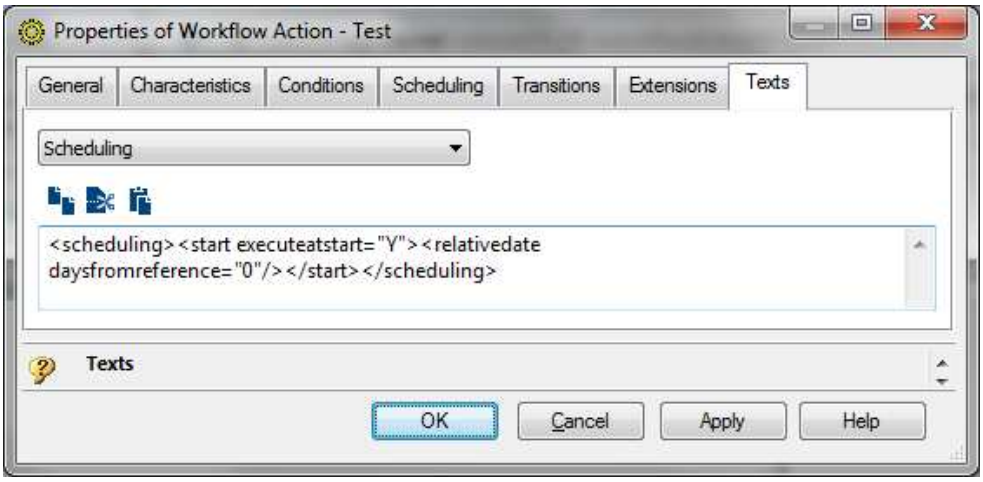
The MetaAttribute "Scheduling" is added on "Workflow Action" MetaClass. A MetaPropertyPage is added on "Workflow Action" MetaClass, this MetaPropertyPage includes a platform provided scheduling propertypage using this syntax:

```
SchedulingMap=Map(~e)PMRueCFbLR[Scheduling])
SchedulingProperties=Item(~bwjCkgpSFTAH[Scheduling - Characteristics <relative
definition>]),From(SchedulingMap),Control(SubPage),Param(Owner=off,Name=off)
```

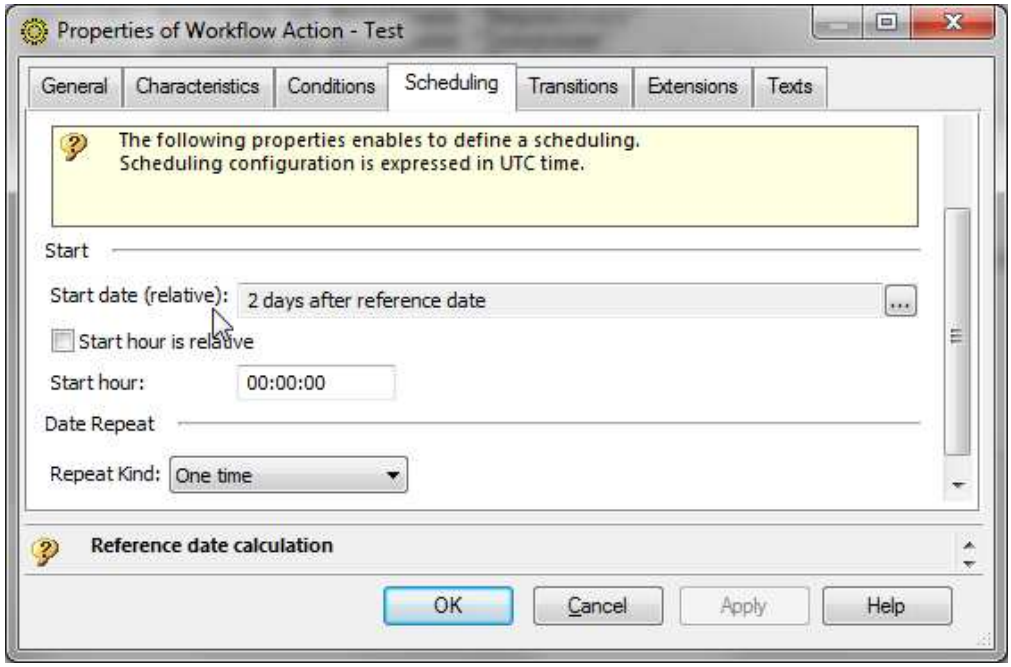


The result on "Workflow Action" objects is as follows:

- Physical scheduling XML text format value



- Scheduling graphical layer



5.3.2 Provided property pages

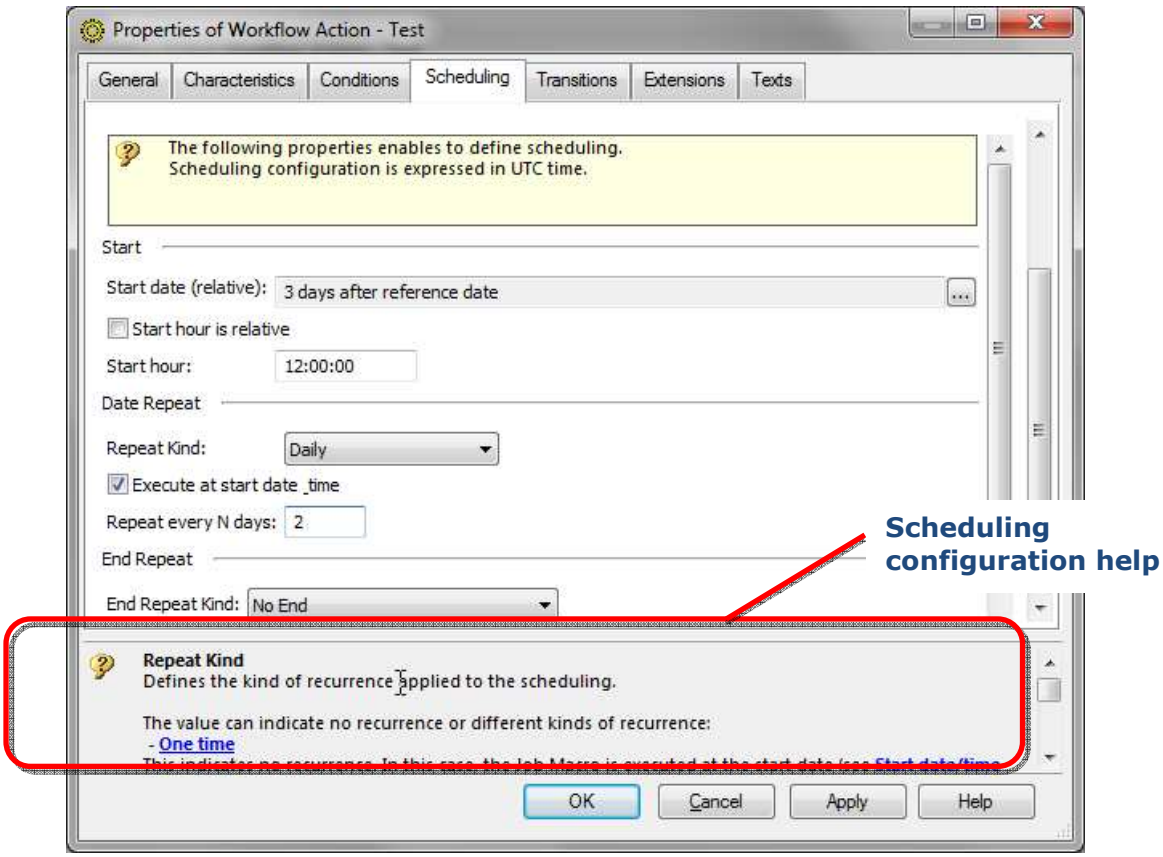
The following MetaPropertyPages can be included into MetaPropertypages of MetaClasses having the "~iUhj4likEzJQ[Scheduling]":

- *~I8QfkFtCF19M[Scheduling - Characteristics]*: this property page enables to configure all possible kind of scheduling
- *~bwjCkgpSFTAH[Scheduling - Characteristics (relative definition)]*: this property page enables to configure scheduling that must be relative
- *~fgkxoz(xH5zN[Scheduling - Characteristics (readonly)]*: this property page enables to display a scheduling configuration

5.3.3 Property page description help

All scheduling properties are commented. These properties are the ones that can be listed by exploring the scheduling MegaObject type as described in section Scheduling MegaObject interface p. 20.

Thus, help for all properties is available into the contextual help:



6 IMPLEMENTING A JOB

6.1 Implemented Function description

When a Trigger is launched the Job Macro is executed. A Job Macro must implement the following function:

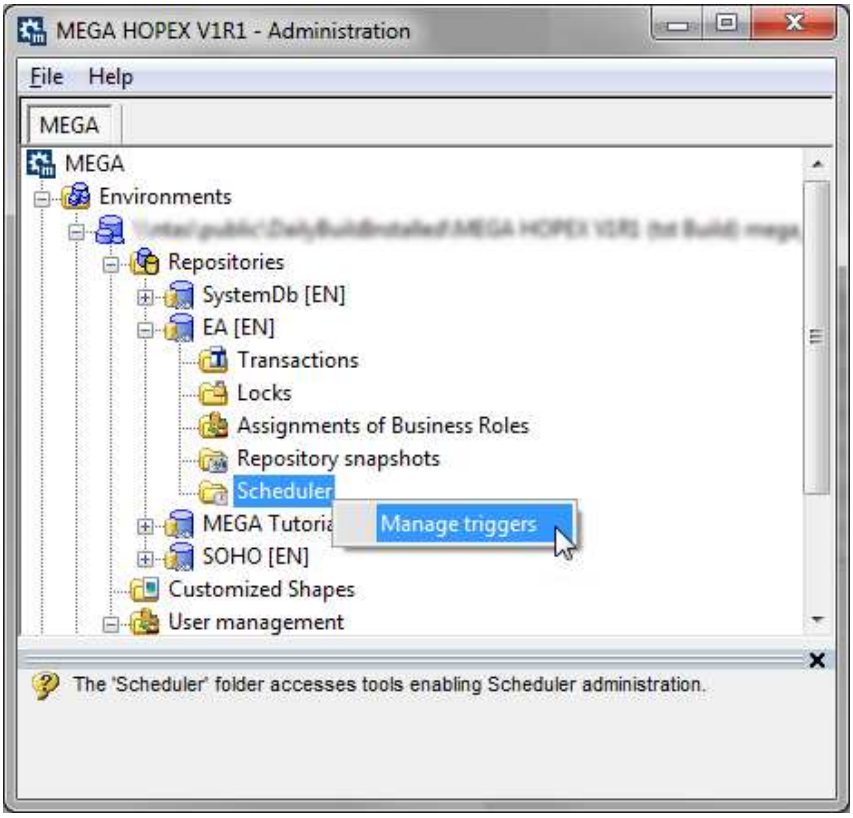
- Function:
  - RunScheduledJob
- Parameters:
  - mgobjJob As MegaObject
    - This parameter is the object describing the Job.
    - Use *GetTypeObject().Explore* on a Job MegaObject to explore the Job type.
    - In particular, use *GetContextString()* Method to recover the string set at Trigger registering (see AddTrigger).
  - objJobResult As Object
    - This parameter is the object enabling to return information needed by the Scheduler after the Job execution.
    - The objJobResult parameter type is composed of the following properties to be set into the Job implementation
      - DiagMessage As String: a diagnosis message used when the job succeed.
      - ErrorMessage As String: an error message in case of job failure (when the Job Function returns False to indicate the job failure).
      - RemoveTrigger As Boolean: in case of job success, set blnRemoveTrigger to True to remove the job trigger so that the job will not be executed anymore.

6.2 Job Function Template

```
'MegaContext(Fields,Types)
Option Explicit

' Function    : RunScheduledJob
' Description : Implement this function to specify what to do when a job is triggered.
' Parameter   :
'   - mgobjJob : object describing the job (this object is not a repository object)
'   - objJobResult : object enabling to return some informations
'       * DiagMessage As String : a diagnosis message used when the job succed
'       * ErrorMessage As String : an error message in case of job failure => return
False to indicate job failure
'       * RemoveTrigger As Boolean : in case of job success, set blnRemoveTrigger to
True in order to remove the job trigger so that the job will not be executed anymore
' Return      :
'   - Boolean : return True in case of job success or False in case of job failure, in
this case, set an error message into strErrorMessage

Function RunScheduledJob (mgobjJob As MegaObject, objJobResult As Object) As Boolean
  ' Initializing variables
  Dim mgRoot
  Set mgRoot = mgobjJob.getRoot()


  ' Write some code here ...


  ' Return True in case of job success
  ' Return False in case of job failure, in this case, set an error message into
strErrorMessage
  RunScheduledJob = True
End Function
```

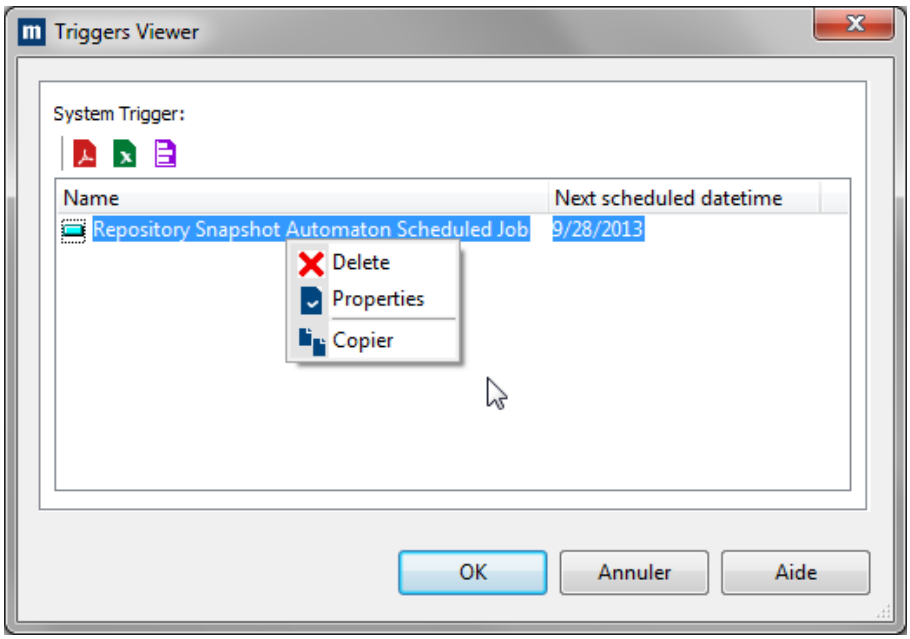
7 TRIGGERS ADMINISTRATION

Triggers can be managed from the **Administration** application:



The **Manage triggers** tool enables to:

- access to triggers properties:
  - Next execution dates
  - Scheduling configuration associated
- delete triggers


Triggers Viewer
System Trigger:
Name
Next scheduled datetime
Repository Snapshot Automaton Scheduled Job
9/28/2013
Delete
Properties
Copier
OK
Annuler
Aide

HOPEX APIs

Sommaire

All about starting with APIs

VB Script and Java APIs

1 INTRODUCTION

MEGA provides APIs enabling access to the repository and handling actions on the repository.

The aim of this document is to provide information required to start with VB Script and Java APIs in MEGA.

In MEGA, to execute code using APIs, always use the macro concept. Macros can be written in VB script or Java. Depending on the implementation type to be added, you must choose the suitable programming language as follows:

- VB Script for simple implementations (few algorithmic code) and when implementing code close to the MetaModel (Command, Property Pages, calculated attributes, etc.).

The advantage of using VB Script is that you can access and easily read your code.

- Java for lengthy processing (including more algorithmic code) and requiring higher performance such as for exports, synchronizations, etc.

The disadvantage of using Java is that the code is directly stored in a jar library and not directly readable.

This document details these two access modes to APIs (VB Script and Java) and provides a toolkit to use in your implementation.

It is important that you bear in mind the recommendations included in this document regarding performance and confidentiality:

- MEGA applications are Web applications, execution time must be optimized to avoid any potential timeout of the application.
- Clients are increasingly demanding regarding security.

Be careful during migration and maintenance, sources must remain available.

Be aware that:

- Advanced customizations may require additional development to work properly after migrating to a MEGA next major version.
- MEGA Technical Support does not provide assistance with developing, maintaining or upgrading MEGA advanced customizations.

For information on:

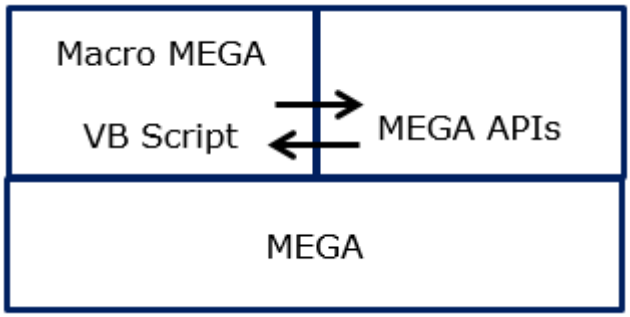
- VB Script functions see the ***API annex - API Reference guide*** Technical Article
- Java MEGA API documentation, see the ***JavaDoc*** documentation (html format) accessible from MEGA menu bar (**Help > APIs > JavaDoc**).

  The ***JavaDoc*** documentation is provided in the MEGA installation "java\doc" directory.

2 CREATE A VB SCRIPT COMPONENT

2.1 VB Script component architecture

MEGA APIs are natively COM and can be used by COM application. The VB Script language uses the Dispatch protocol to handle objects and access their methods. Therefore using VB Script language to access to MEGA objects is completely standard. The following diagram illustrates the way the VB Script component interacts with MEGA APIs:



2.2 Creating and editing a VB Script component

VB Script code can be written and used:

- in an external file
- in a MEGA macro
- directly in the MEGA script editor.

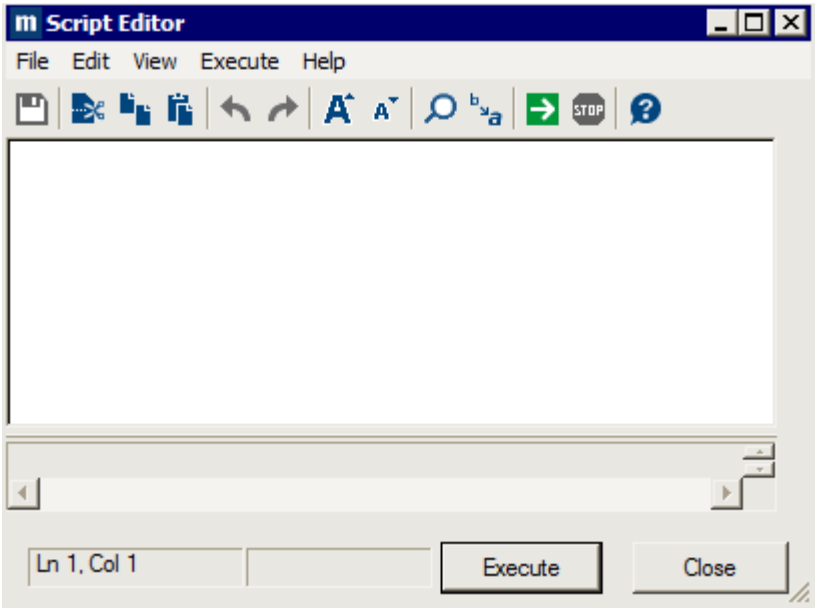
You can use the script editor to write VB scripts using MEGA APIs, which enables you to execute operations on MEGA repository content.

2.2.1 Creating a VB Script component

To create a VB Script component with the Script Editor:

1. From your workspace, in the **MEGA** menu bar, select **Tools > Script Editor** to launch the script editor.

   The **Script Editor** dialog box opens.



2. From **Script Editor** menu bar, select **File > Save As**.
3. Select either:
   - **a macro** to save the script as a macro in the repository,
   - **a file** to save the script as an external VB Script file.
4. Enter your VB Script code.

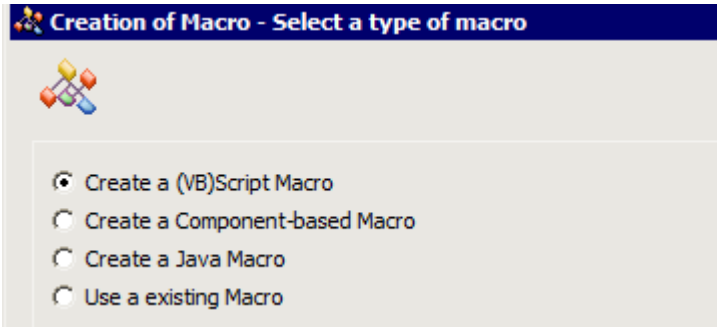
2.2.2 Creating a VB Script macro with the wizard

MEGA includes a wizard that enables preparation of macro parameters and code according to the use imagine by the user. Code is initialized according to the MEGA concept implemented.

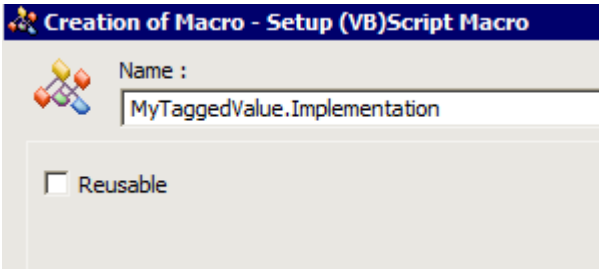
To use the macro creation wizard on an object of TaggedValue type:

1. From MEGA toolbar, click **Explore**  .
2. From **Explore** window, create a new object of the **TaggedValue** MetaClass.
3. Display **Empty Collections**.
4. Right-click **Implementation** folder and select **New**.

   The **Creation of Macro** wizard appears.

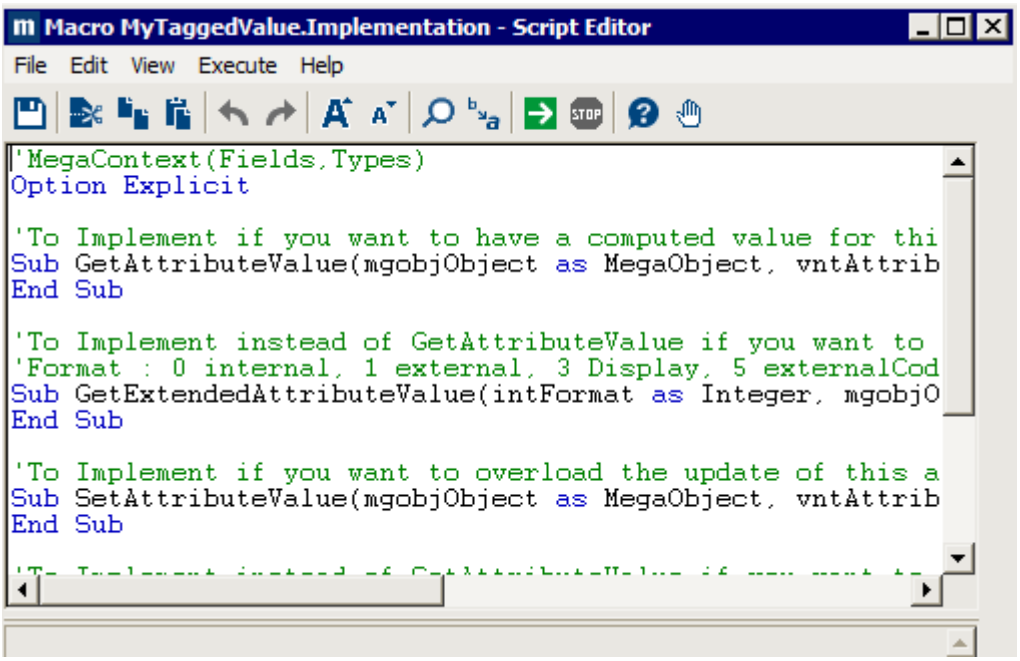


5. Select **Create a (VB)Script Macro**.
6. Click **Next**.

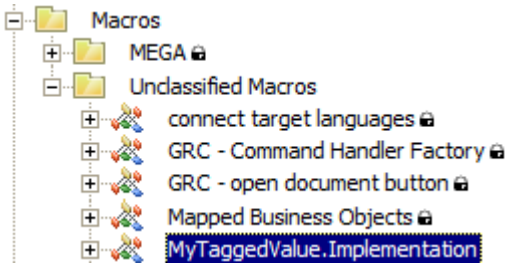


7. (Optional) Modify the **Name** of the macro.
8. (Optional) If the macro you have created can be reused for another TaggedValue, select the **Reusable**.
9. Click **Finish**.

   The macro (e.g.: "MyTaggedValue.Implementation) is created and its code already initialized.



   The macro (e.g.: "MyTaggedValue.Implementation) is stored in the **Macros** folder, **Unclassified Macros** sub-folder.



2.2.3 Editing a VB Script code

The **Script Editor** menu bar enables you access commands that ease your editing actions.

The **Script Editor** tool bar enables you to easily access these commands to simplify your editing work.

Alternatively, you can use the menu shortcuts.

To edit a VB Script file/macro with the Script Editor:

1. In the **Script Editor** menu bar, select **File > Open** and select the script type **a File** or **a Macro**.
2. Select the file/macro.

   The file/macro code is displayed in the **Script Editor**.

To execute the script:

➔ In the **Script Editor** toolbar, click **Execute**  .

Alternatively in the **Script Editor** menu bar, select **Execute > Execute**.

Global code execution starts.

To stop your script execution, in the **Script Editor** toolbar, click **Stop**  .

To save the script code:

➔ In the **Script Editor** toolbar, click **Save**  .

Alternatively in the **Script Editor** menu bar, select **File > Save As.**

To move parts of the code to another position:

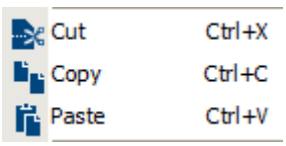
➔ In the **Script Editor** toolbar, select the code part and click either:

Cut  to cut the selected code

Copy  to copy the selected code

Paste  to paste the selected code

Alternatively in the **Script Editor** menu bar, select the **File** menu and then select the corresponding command or use the menu shortcuts:



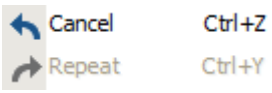
To cancel or repeat an action you have carried out:

➔ In the **Script Editor** toolbar, click:

Cancel  to cancel your last action

Repeat  to repeat you last action

Alternatively in the **Script Editor** menu bar, select the **Edit menu** and then select **File** menu and then select the corresponding command or use the menu shortcuts:



To enlarge or reduce font size:

➔ In the **Script Editor** toolbar, click:

Enlarge Font  to enlarge the font size

Reduce Font  to reduce the font size

Alternatively in the **Script Editor** menu bar, select the **View** menu and then select the corresponding command.

To query or replace part of your code:

➔ In the **Script Editor** toolbar, click:

Query  to find a character string in your code

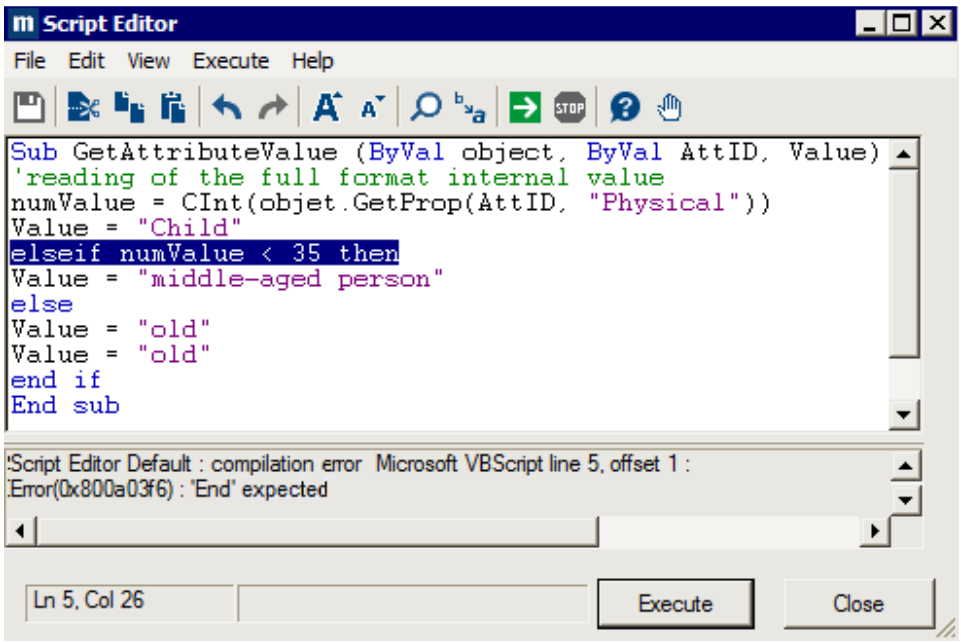
Replace  to find a character string in your code and replace it by another one

Alternatively, in the **Script Editor** menu bar, select **Edit > Query** or **Edit > Replace**.

To test your script:

➔ In the **Script Editor** toolbar, click **Execute**  to test your script.

If there is an error in your script, the line in error is highlighted and an error message is displayed in the bottom pane. This error message includes the line number in error and the error type.



To stop your script execution, in the **Script Editor** toolbar, click **Stop** .

Alternatively, in the **Script Editor** menu bar, select **Execute > Execute** or **Execute > Stop**.

To reach a specific point in your code:

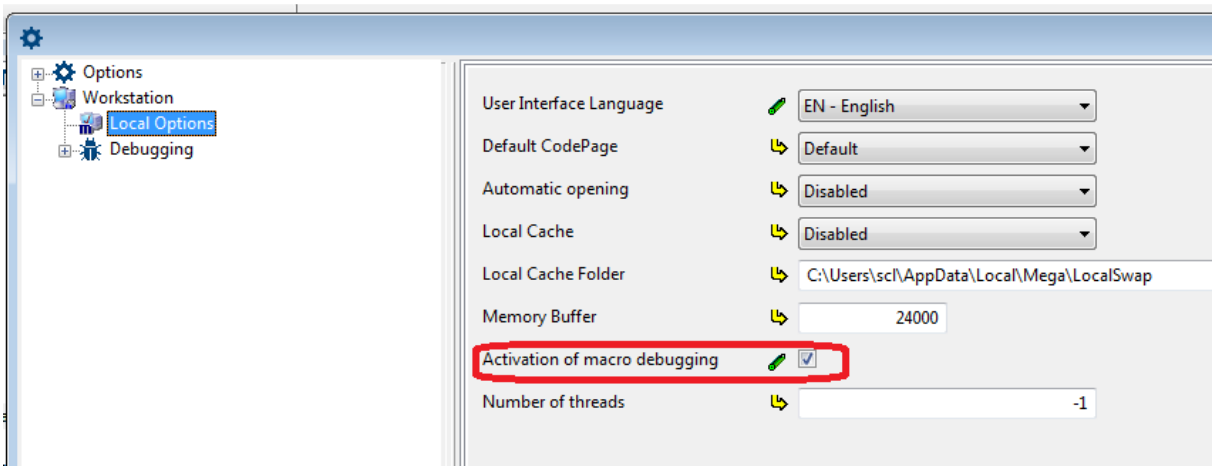
1. In the **Script Editor** menu bar Select **Edit > Go to Line**.
2. In the dialog box that appears, enter the line number you want to reach and click **OK**.

2.3 Debugging a VB Script component

2.3.1 Activating macro debugging

To activate macro debugging:

1. From the main MEGA menu bar, select **Tools > Options**.
2. In the left pane, expand **Workstation** and select **Local Options**.
3. In the right pane select **Activation of macro debugging**.
4. Exit MEGA for the activation to be taken into account.



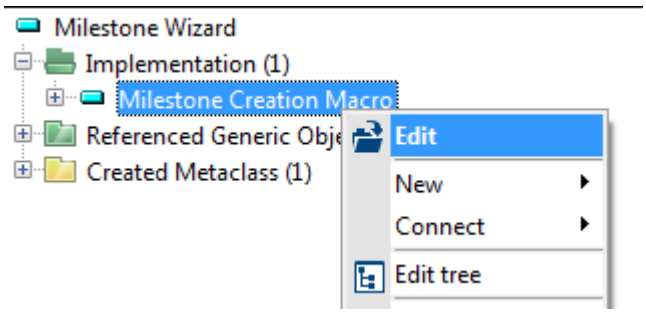
Upon checking this box, this option state is automatically written in the MEGAWKS.INI file:

```
[API Script]
ScriptDebug=1
```

2.3.2 Macro Debugging

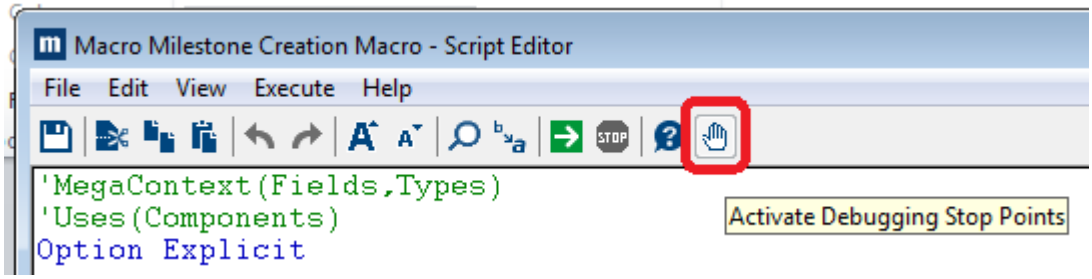
To debug a macro:

1. Right-click the macro to be debugged and select **Edit**.



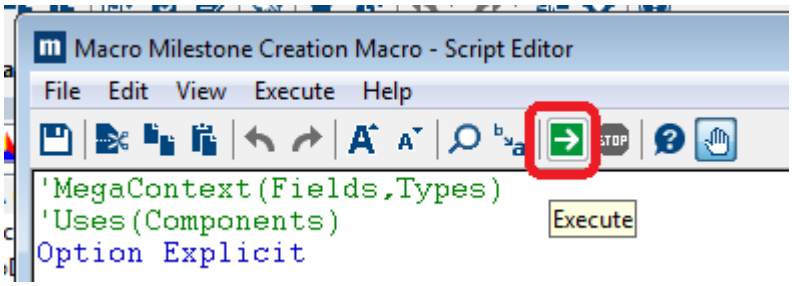
   The selected macro is edited in the **Macro - Script Editor**.

2. In the **Macro - Script Editor** tool bar, click **Activate Debugging Stop Points**  .



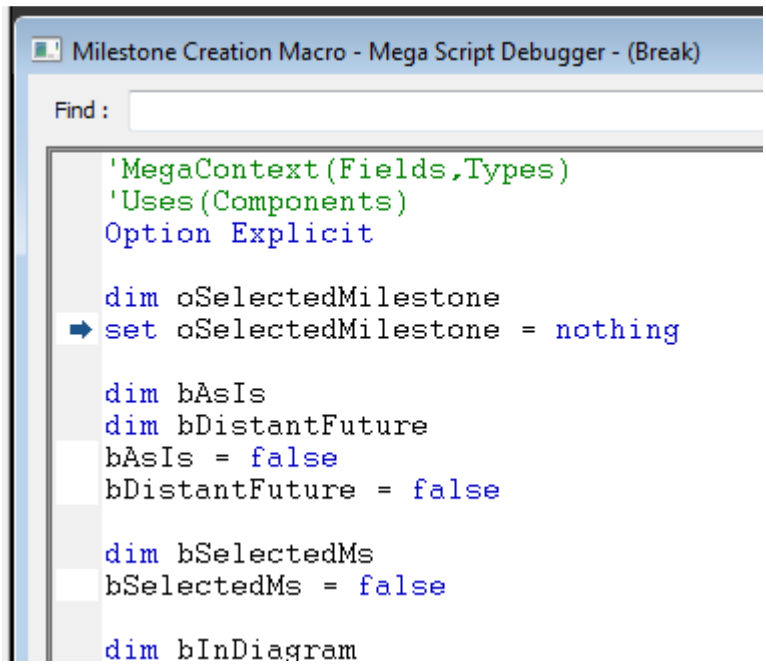
3. Either:

   - **execute** the macro: in the **Macro - Script Editor** tool bar, click **Execute**  , or



   - perform the sequence of events which calls the macro.

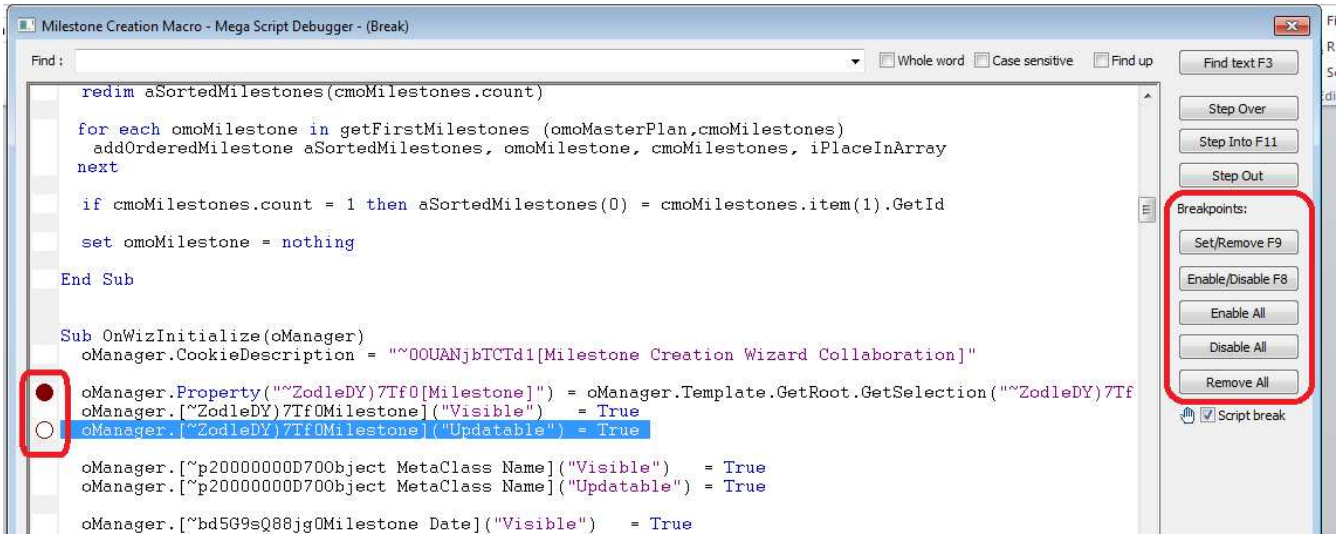
   In both cases, the **Mega Script Debugger** opens:



4. Add breakpoints:

   In the **Mega Script Debugger**, with the **Breakpoints** buttons you can:

   - create and delete (**Set/Remove**) a breakpoint, or
   - enable and disable (**Enable/Disable**) a breakpoint.



5. To create a breakpoint, put the cursor at the chosen code line and click **Set/Remove**: the breakpoint is created. It looks like a red sphere.

   You can:

   - disable a breakpoint: put the cursor at an already created breakpoint line and click **Enable/Disable**: the breakpoint is disabled.

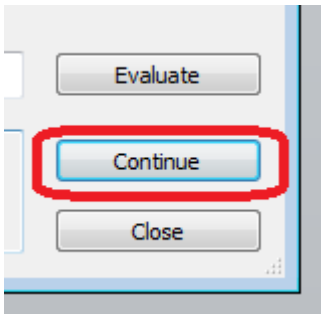
     It looks like a red circle.

   - remove or enable a breakpoint: put the cursor at the chosen breakpoint line and click the **Set/Remove** button.

Click **Enable All**, **Disable All**, or **Remove All** when you want to apply the same command to all macro breakpoints.

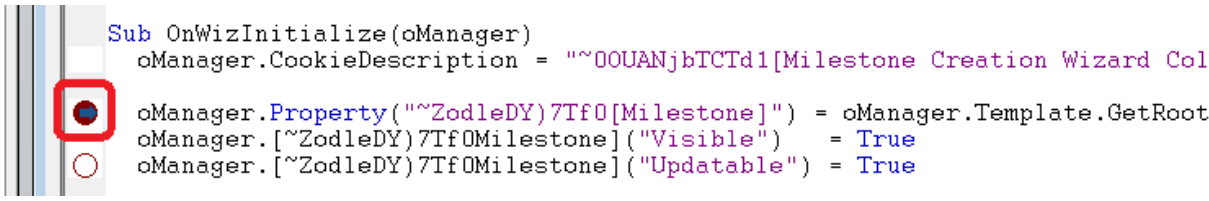
Note: Commands on breakpoints are active whenever the Mega Script Debugger is opened.

6. Click **Continue**.



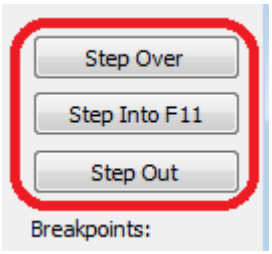
7. Do the sequence of events which allows executing the code where breakpoints are created and enabled.

   The macro Script Debugger opens again,



A blue arrow in the breakpoint red sphere shows at which breakpoint the debugger has stopped.

- The three buttons **Step Over**, **Step Into** and **Step Out** allow running the program step by step, entering or going out of procedures: Functions and Subroutines.

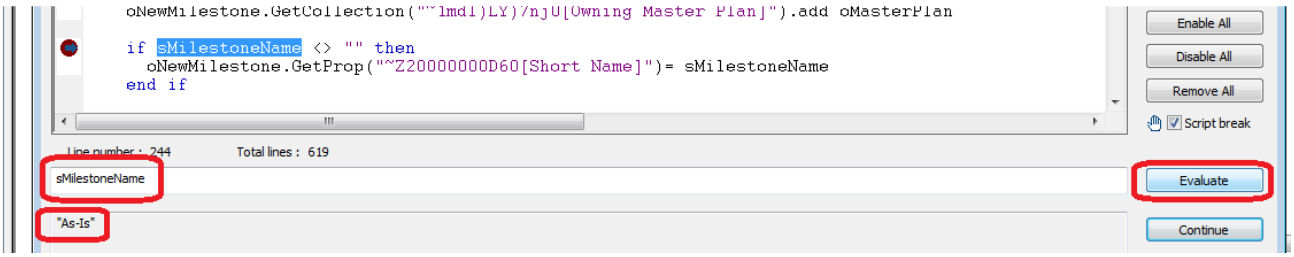


  A blue arrow always shows the current line.

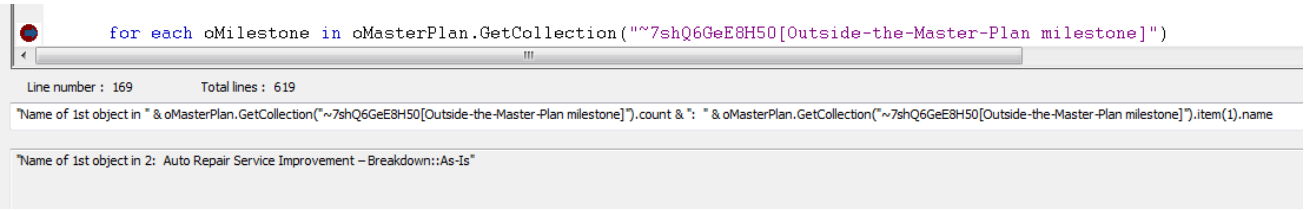
- You can examine the current state of the program and track the variable values.

  Write the variable name or expression to be evaluated at the bottom of the Mega Script Debugger window, and click **Evaluate**.

  The value is displayed under the expression to be evaluated.



- You can evaluate complex expressions using API:



The result of:

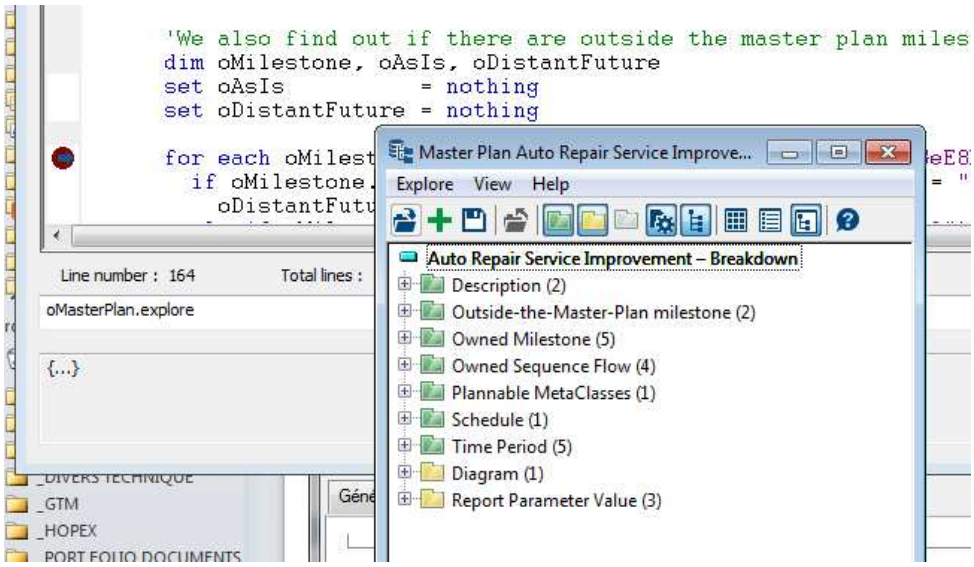
```
"Name of 1st object in " & oMasterPlan.GetCollection("~7shQ6GeE8H50[Outside-the-Master-Plan milestone]").count & ": " & oMasterPlan.GetCollection("~7shQ6GeE8H50[Outside-the-Master-Plan milestone]").item(1).name
```

Is:

```
"Name of 1st object in 2: Auto Repair Service Improvement – Breakdown::As-Is"
```

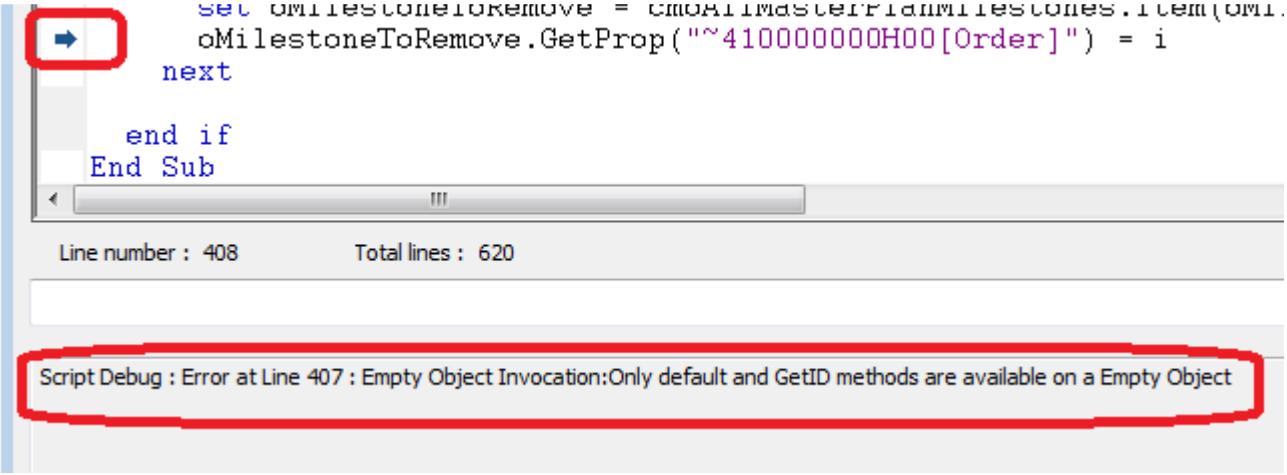
- Another example of API use: the method Explore can be called on a MegaObject, and the Explorer is opened in another window:

In the Evaluation field enter: `mgobjMegaObject.Explore`.



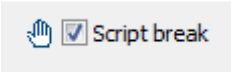
1. Click **Continue** to continue the program execution.

   When there is an error during the program execution, the Mega Script Debugger window opens, the program is stopped at the wrong line and an error message is displayed:



2. Click **Continue** to continue the program execution or click **Close** to stop it.

- To deactivate debugging for a macro when the Mega Script Debugger is opened, unselect **Script break**. It may be useful if the macro is called several times during a sequence of events and if the Debugger window opens every time it is called.



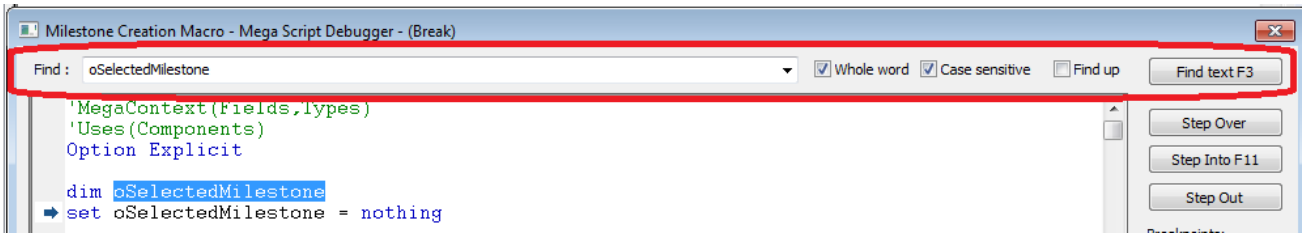
- If several macros have to be debugged at the same time, Debugging must be activated for each of them. The Mega Script Debugger window which corresponds to each of them is displayed when the corresponding macro code is executed to allow creating breakpoints.

2.3.3 Find Text command bar

At the top of the Mega Script Debugger window, a specific command bar allows searching for a string in the macro script.

To find a string in the macro script:

➔ In the **Find** field, enter the string and click **Find Text**.



3 CREATE A JAVA COMPONENT

This chapter details the creation of a MEGA plug-in in Java.

Availability: from HOPEX 1.0.

The Java APIs provided by MEGA enable access to the possibilities offered by the existing VB Script API.

The HOPEX 1.0 Java APIs allow for the creation of plug-ins written in Java. The component written in Java is deployed in the form of a Jar library referenced via a macro.

The following subjects are covered:

- Using Eclipse as the environment for developing a MEGA plug-in written in Java

  See Debugging a Java component called from MEGA p. 45.

- Creating a MEGA component implemented in Java

  See Debugging a Java component called from MEGA p. 45.

- Implementing the Java MEGA API using the example of creating a MetaCommand

  See Debugging a Java component called from MEGA p. 45.

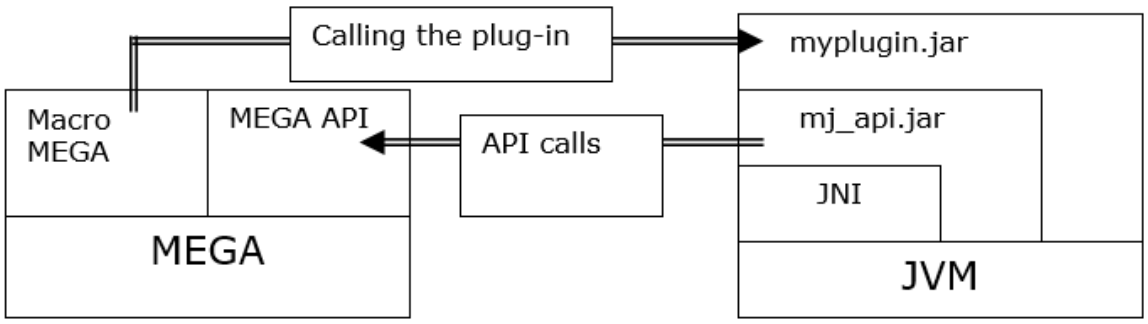
- Debugging a MEGA plug-in written in Java with Eclipse

  See Debugging a Java component called from MEGA p. 45.

3.1 MEGA and Java general considerations

3.1.1 Architecture of a MEGA plug-in written in Java

The following diagram illustrates the call from MEGA of a plug-in written in Java. The plug-in uses repository content via MEGA APIs.



Versions of the applications used in this chapter are as follows:

MEGA	HOPEX 1.0
JDK	1.7 Available at the following location: http://www.oracle.com/technetwork/java/javase/downloads/index.html
Eclipse	Juno Available at the following location: http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/juno/SR2/eclipse-java-juno-SR2-win32.zip

Calling the plug-in

A macro enables referencing of the plug-in in MEGA; It identifies the class to be instanced.

Depending on the macro use case, the instanced class must implement a specific interface. It is via this interface that the plug-in will be called. In HOPEX 1.0, the functionalities that can be implemented in VB script can also be implemented in Java.

Examples:

- Meta Wizards
- Methods
- Update Tools
- Commands
- etc.

Using the MEGA API in the plug-in

When called, the plug-in can in turn call the MEGA application via the Java MEGA API.

The Java MEGA API is a set of objects and interfaces enabling, via JNI, calling the existing VB script MEGA API (Dispatch, Automation…).

This set of objects and interfaces is available in the mj_api.jar library delivered with MEGA.

3.1.2 The JVM

In order to function, the Java MEGA API depends on a JVM (Java Virtual Machine).

This JVM is contained in the JRE (Java Runtime Environment) delivered with MEGA.

Java plug-ins and applications integrating with MEGA must be developed using a version of Java compatible with the JRE delivered with MEGA.

3.1.3 The JDK

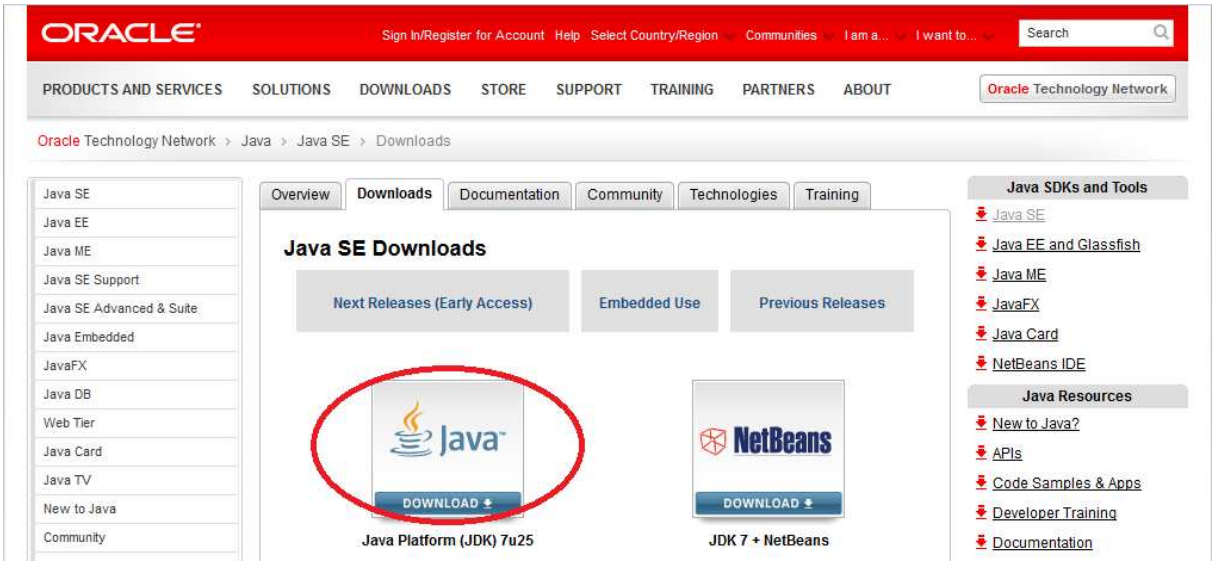
While the JRE enables only the execution of Java applications, the JDK also enables their creation: its contents include the Java compiler and all elements required for developing a Java application.

To develop a MEGA plug-in in Java, it is necessary to obtain and install the compatible JDK.

To download and install the JDK:

1. Go to the Oracle Web Site.

   To get the url path see table Architecture of a MEGA plug-in written in Java section p. 15.
2. Download the JDK.



3.2 Development environment (Eclipse)

Eclipse is the Java-oriented development environment preferred for developing Java components integrated in MEGA or interfacing with MEGA.

3.2.1 Installing Eclipse

Downloading

MEGA recommends using the Juno release of Eclipse.

➔ Download **Eclipse** from the url given table Architecture of a MEGA plug-in written in Java section p. 15.

You obtain a zip Archive file **eclipse-java-juno-SR2-win32**.

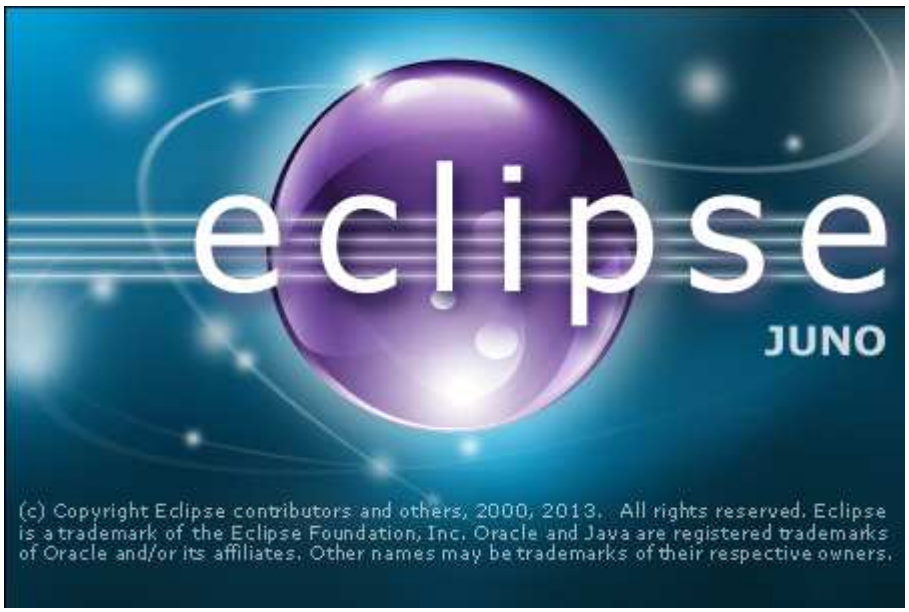
Installing Eclipse

To install Eclipse:

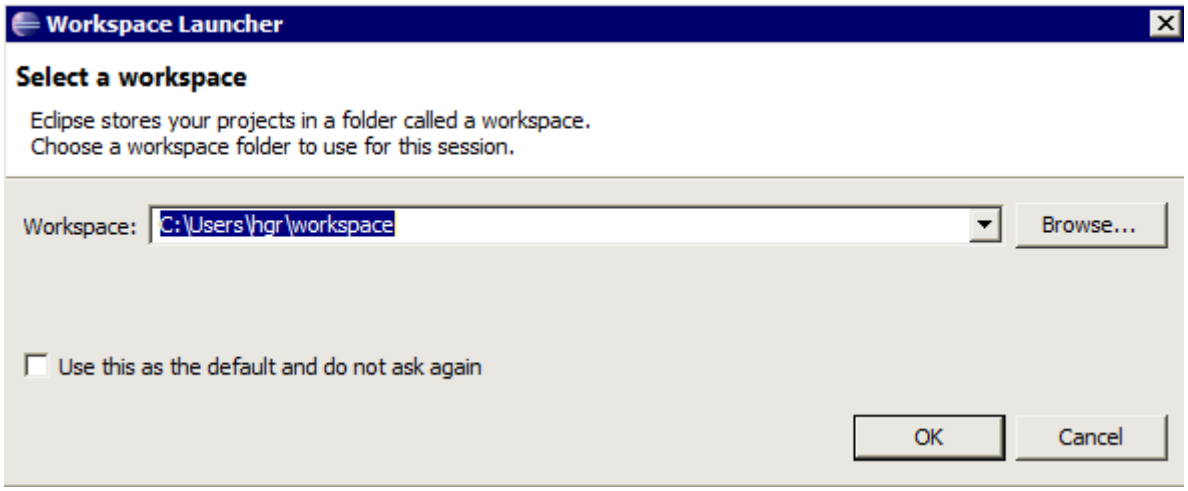
1. Unzip the zip Archive file at the hard drive root or in the **Program Files** folder.

   Note: Certain versions of Eclipse are delivered with a JRE, for others you must obtain it.

2. Launch **Eclipse**.



3. (optional) In the Workspace Launcher, modify your workspace folder.



4. Click **OK**.

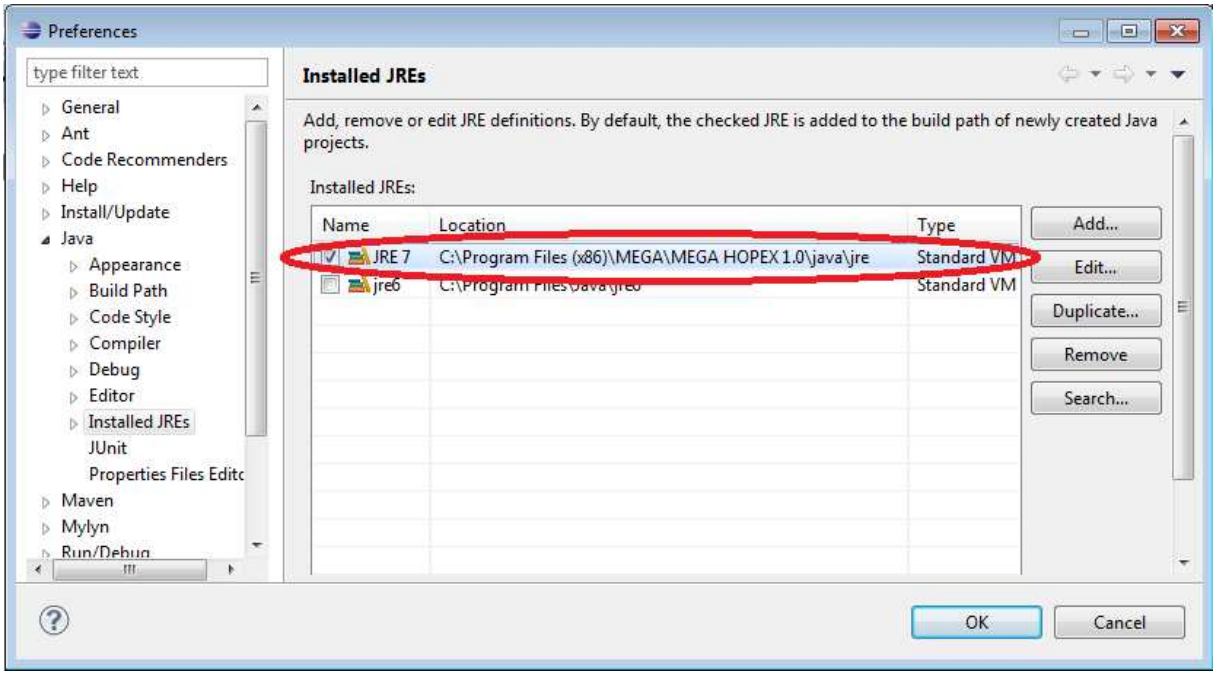
Selecting a JRE compatible with MEGA in Eclipse

Selecting the JRE to be used by default in Java projects avoids using an incorrect JRE in new Java projects.

Using a JRE version delivered with MEGA insures total compatibility.

To select in Eclipse a JRE compatible with MEGA:

1. From the Eclipse menu bar, select **Window** > **Preferences**.
2. From the **Preferences** dialog box, in the left pane, expand **Java** node and select **Installed JREs**.
3. In the **Installed JREs** right pane, add a reference to the MEGA compatible JRE.



3.2.2 General writing conventions

Compliance with a writing convention is highly recommended.

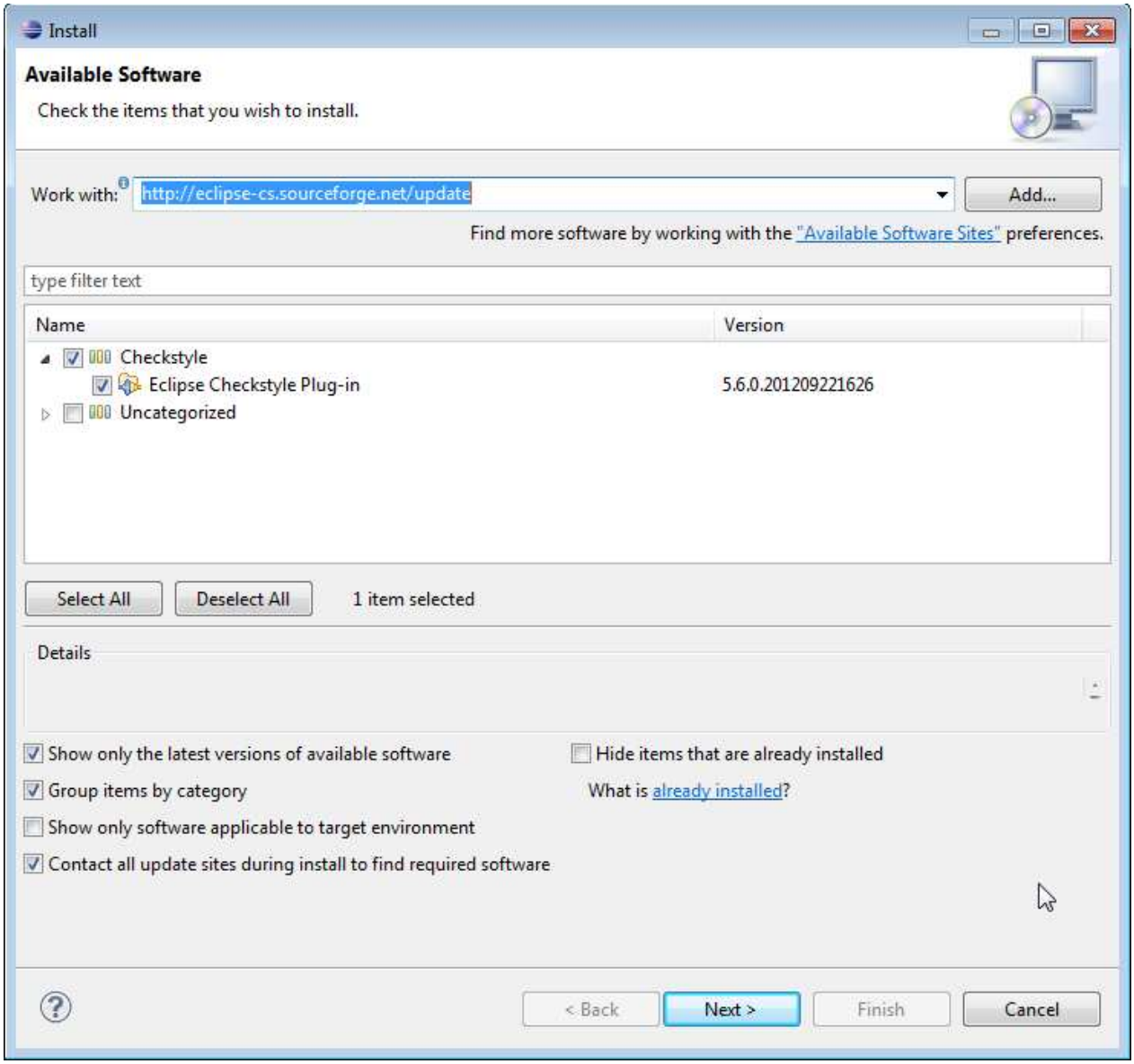
The Eclipse "CheckStyle" plug-in enables integration in Eclipse of automatic management of Java source style validation.

Installing the writing style validation plug-in

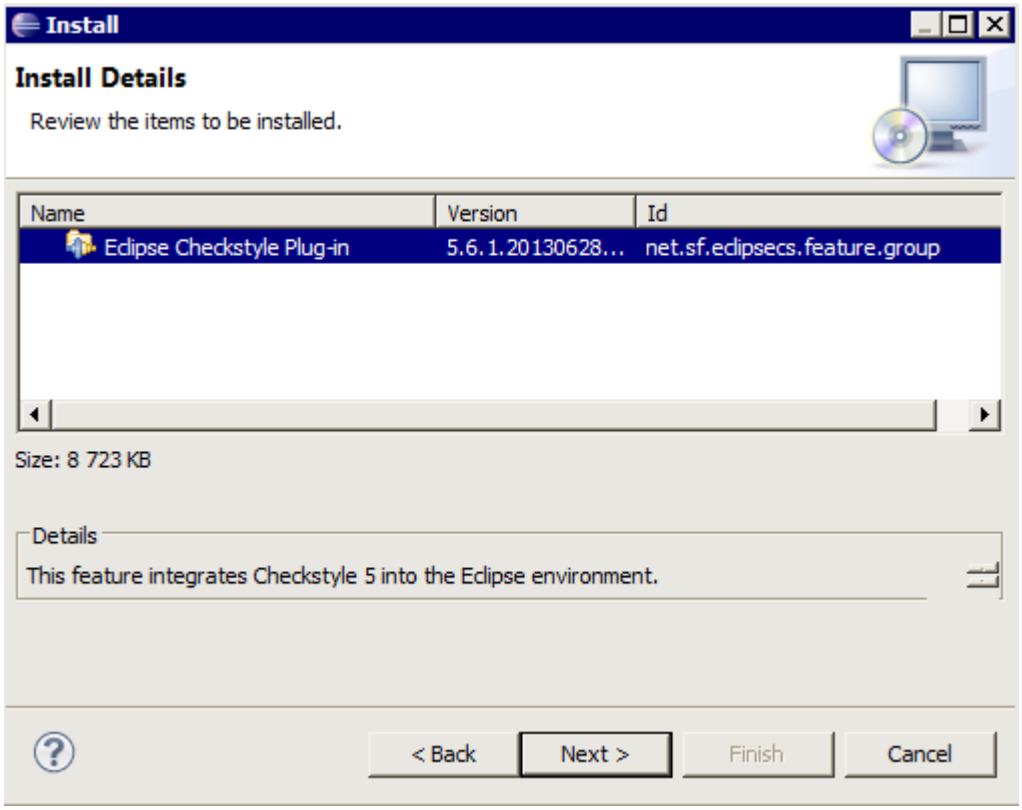
To install the writing style validation plug-in:

1. From Eclipse, in the menu bar, select **Help > Install New software**.

   The **Install** wizard appears.
2. In the **Work with** field, enter the site path: http://eclipse-cs.sourceforge.net/update.
3. Click **Add**.



4. In the pane, select **Eclipse CheckStyle Plug-in**.
5. Click **Next**.

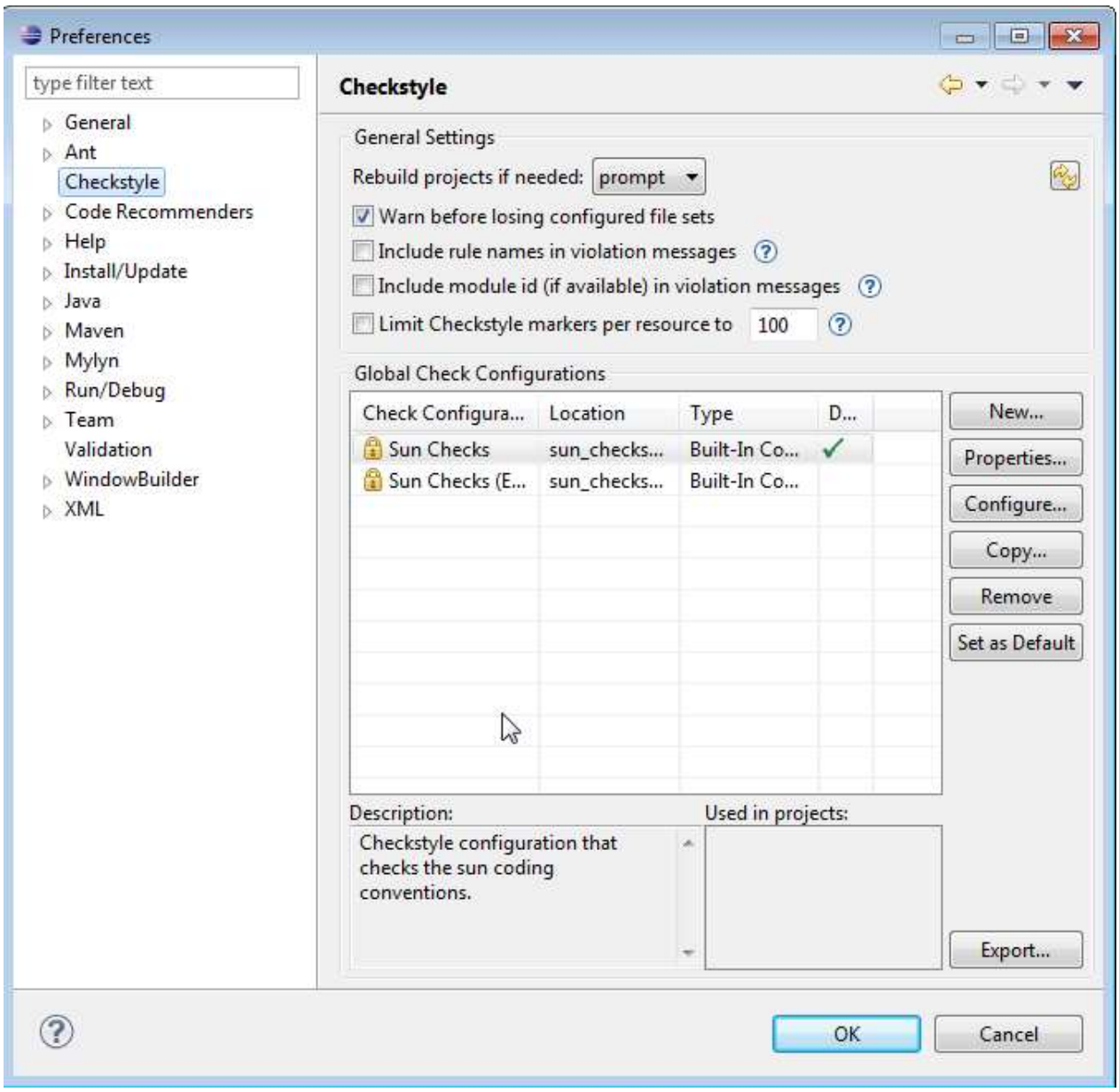


6. Click **Next**.
7. Accept the plug-in installation to execute the plug-in setup.

Configuring writing style check

To configure writing style check:

1. From Eclipse, in the menu bar, select **Window** > **Preferences**.
   The **Preferences** dialog box appears.



2. From the **Preferences** dialog box, in the left pane, select **Checkstyle**.
3. In the **Checkstyle** right pane, select an encoding standard. Standards supplied with the plug-in are:
   - Sun development conventions.
   - Sun conventions slightly modified, adapted to Eclipse default formatting.

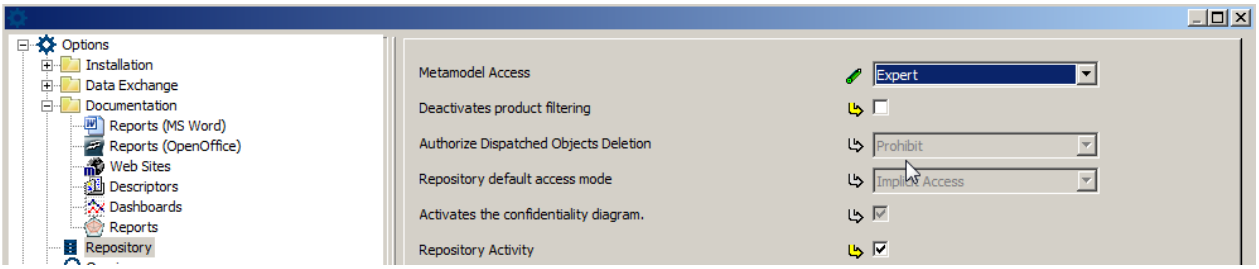
3.3 Calling a component written in Java from MEGA

This section illustrates a call on a Java component using an example of how to create a MEGA macro implemented in Java.

This macro enables writing of text in a file.

3.3.1 Creating a Java macro in MEGA

Prerequisite: To perform the following procedures, the user must have "Expert" access to the metamodel (In Options > Repository).



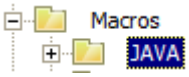
Configuring macros implemented in Java

All macros implemented in Java must reference the target class instanced and called via the macro:

1. From the macro properties dialog box, in the **Characteristics** tab, set the **SystemComponent** attribute value to "Dispatch".
2. Attribute a value of form "java:*Package*/*Class*" to the "_ObjectFactory" property (see example in the next chapter). *Package* is the name of the Java package in which the referenced class is defined. Period characters ('.') of the package name must be replaced by slash characters ('/'). The name of the class is separated from the name of the package by the slash character ('/').

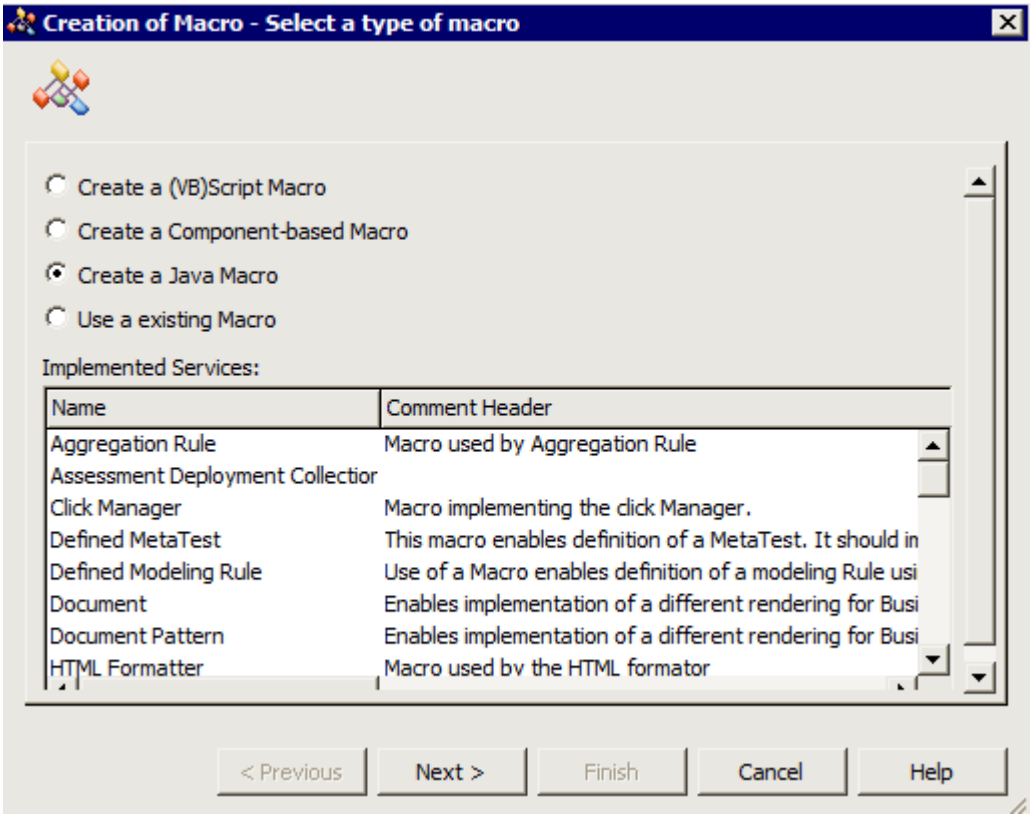
To create and configure a Java macro in MEGA:

1. From the main MEGA menu bar, select **View > Navigation Windows > Utilities**.
2. In the **Utilities** tab, to create a new macro folder, select **Macro > New > folder of Macros**, and name it, for example "JAVA".

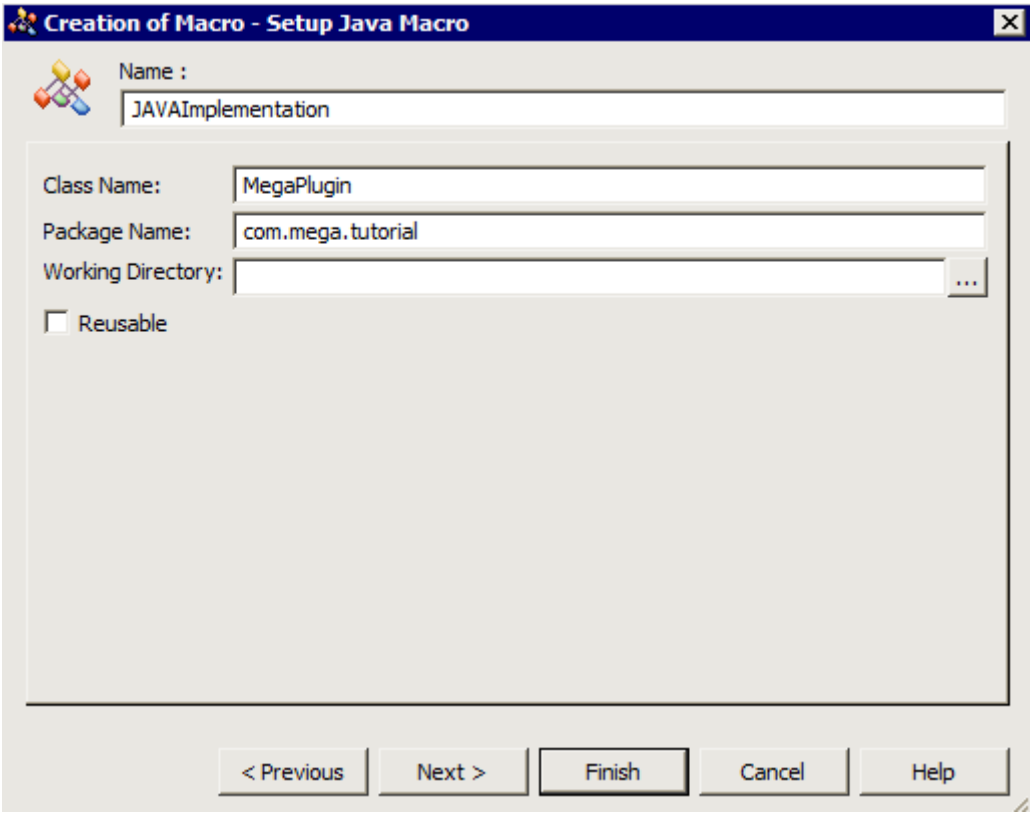


3. From your **JAVA** folder, create a new Java Macro.

The **Creation of Macro** wizard appears.

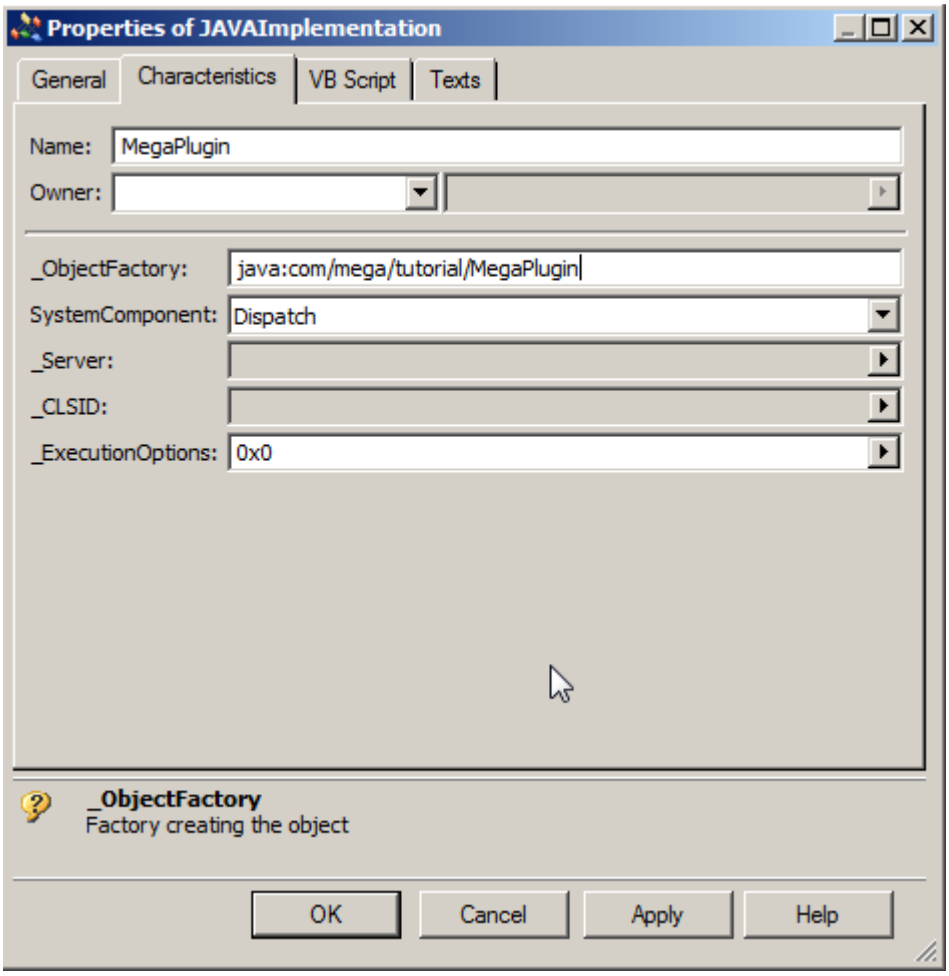


4. Click **Next**.
5. In the **Setup Java Macro**:
    a. In the **Class Name** field, enter for example “MegaPlugin”.
    b. In the **Package Name** field, enter for example “com.mega.tutorial”.



6. Click **Finish** to terminate the wizard.

7. Open the "JAVAImplementation" macro properties, and check that:
   - the **SystemComponent** attribute value is "Dispatch".
   - the **_ObjectFactory** attribute contains the package and class name with the syntax defined in the above section. The value here is defined as "java:com/mega/tutorial/MegaPlugin" (identifies the "MegaPlugin" class of the "com.mega.tutorial" package):

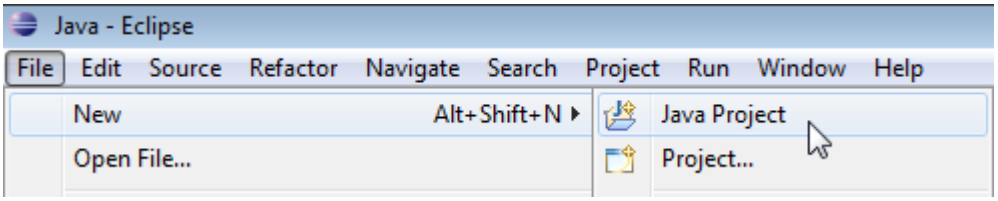


3.3.2 Creating an Eclipse project

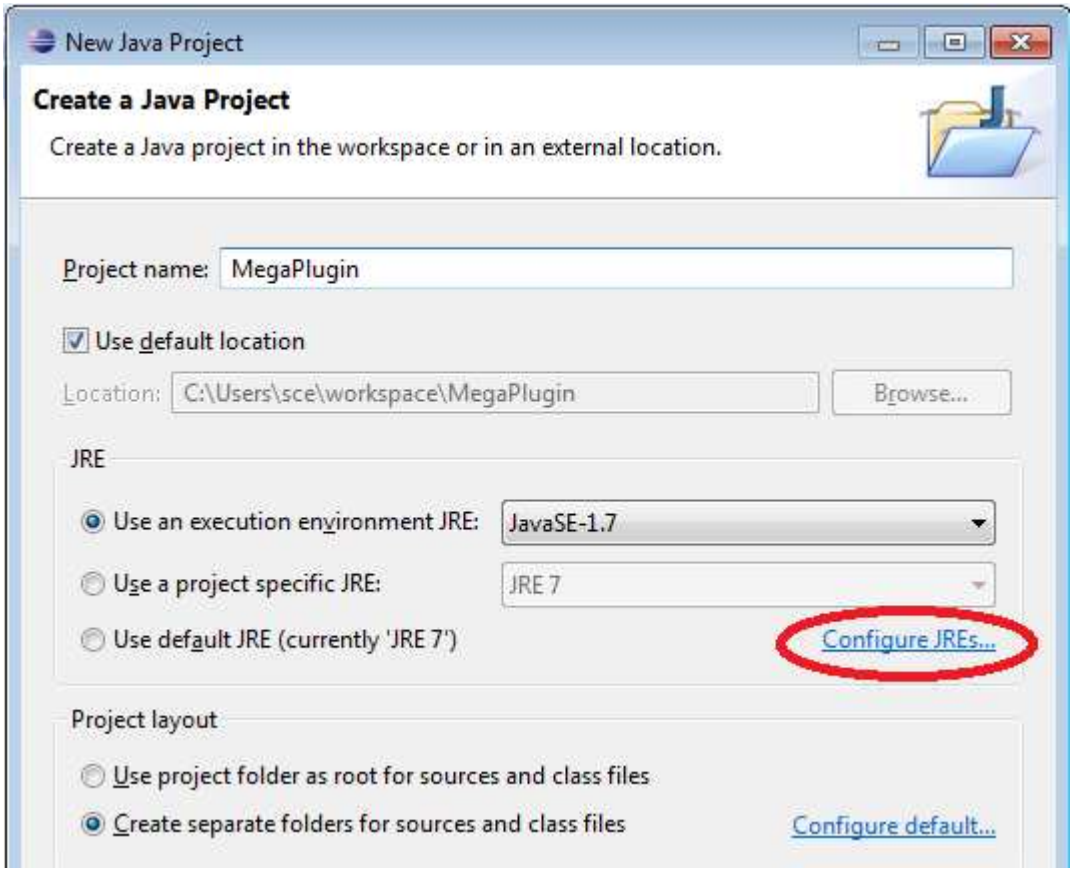
To create the Java component, the "Java" perspective should be used (in Eclipse online Help, see the information relating to use of perspectives).

To create a new Java project:

1. From the Eclipse menu bar, select **File > New > Java project**.



2. Enter the Java **Project name**.
3. Ensure that the JRE used is compatible with the MEGA version for which the Java component is created ("Configure JREs").

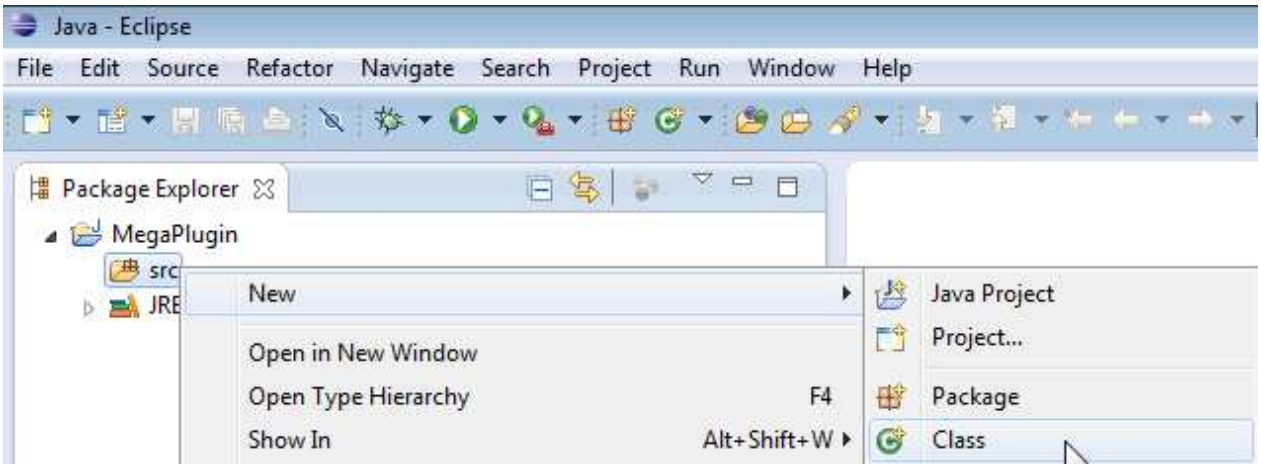


4. Click **Finish** to validate the Java project creation.

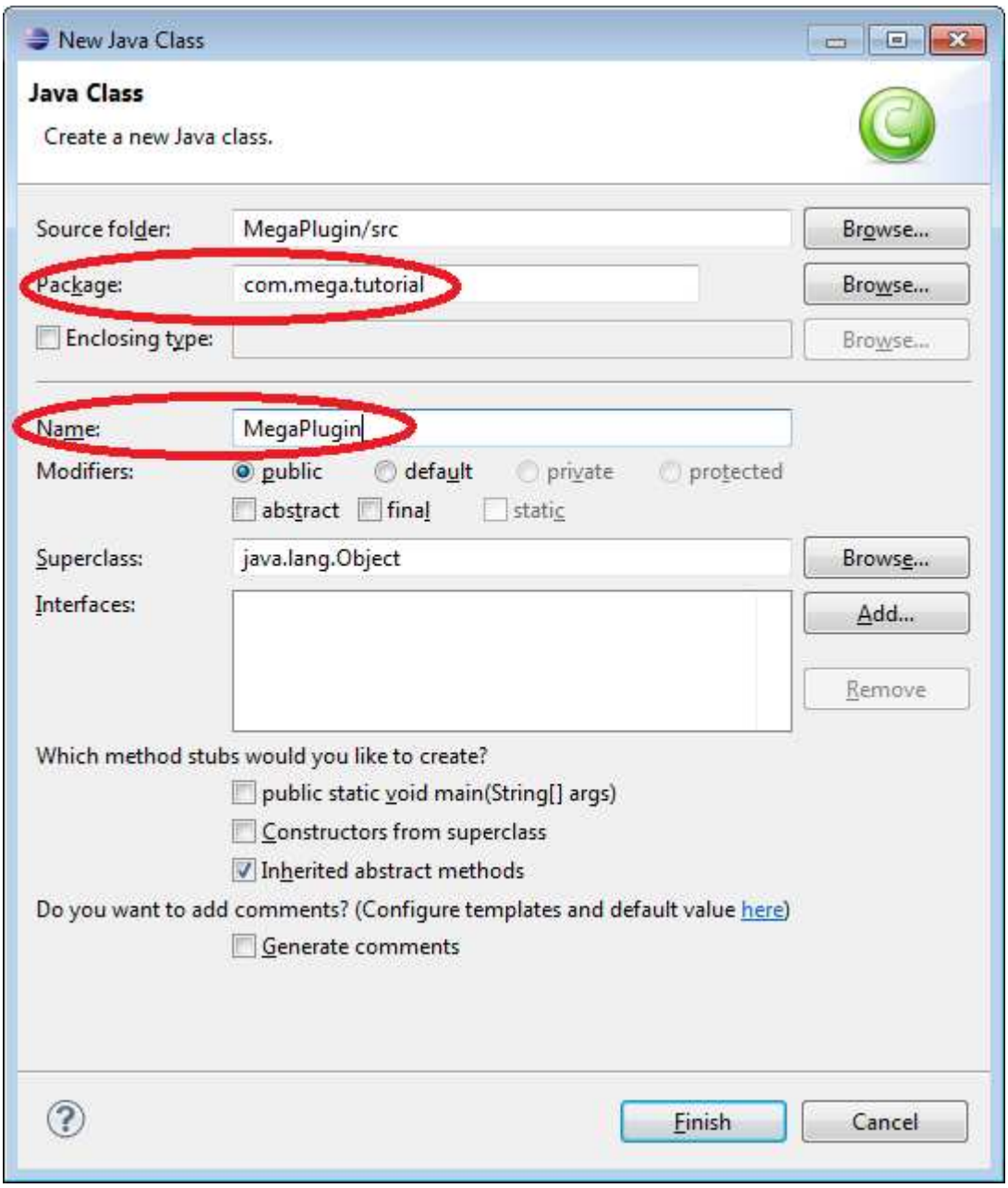
Creating implementation class of the macro

To create an implementation class of the macro:

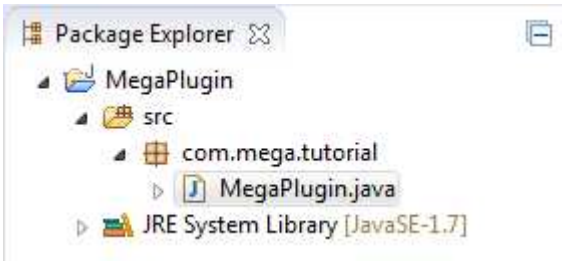
1. From the Eclipse menu bar, select **Window > Show View > Package Explorer**.
2. In the **Package Explorer** tab, expand the **MegaPlugin** project
3. Right-click the **src** folder and select **New > Class**.



4. In the **Package** field, enter "com.mega.tutorial".
5. In the **Name** field enter the class name "MegaPlugin":



6. Click **Finish**.



Note: As previously seen, the package and class created are those identified in the _ObjectFactory" parameter of the corresponding MEGA macro.

3.3.3 Implementing the macro

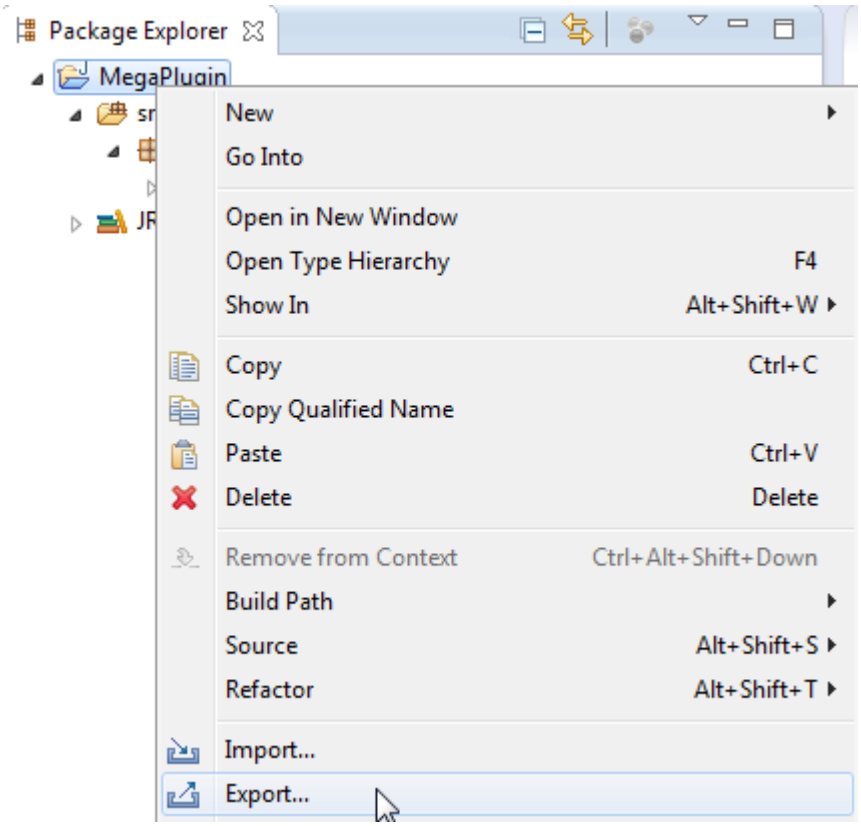
Implementation of the macro consists of a class exposing the writeInFile public method that can be called from MEGA:

```
// Package containing the implementation class.
package com.mega.tutorial;

// Need to write some text in a file.
import java.io.FileWriter;
// Need to throw exception managed by MEGA.
import java.io.IOException;

/**
 * MegaPlugin Macro implementation class.
 */
public class MegaPlugin {

	/**
	 * MegaPlugin Macro implementation class.
	 * @param   filename       Name of the file created.
	 * @param   text           Some text that will be passed from MEGA.
	 * @throws   IOException   Method called from MEGA must throw
	 *                                        IOException exception.
	 */
	public void writeInFile(String fileName, String text) throws IOException
	{
		FileWriter writer = null;
		// Open a FileWriter with file name
		writer = new FileWriter(fileName);

		// Write the text in the file
		writer.write(text);

		//Close the file
		if(writer!=null)
		{
			writer.close();
		}
	}
}
```

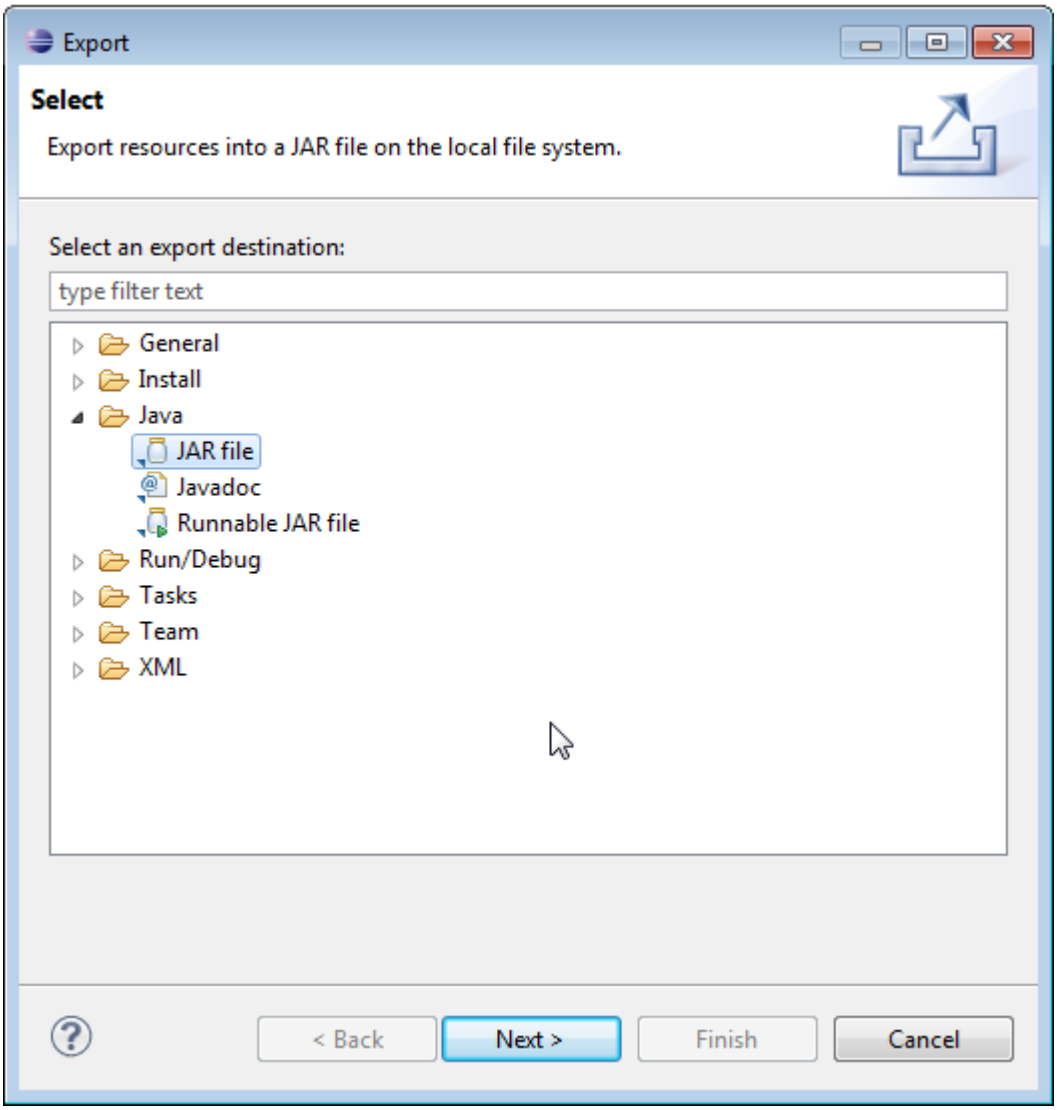
Compiling and deploying the Java component

To compile the Java component in the form of a JAR file

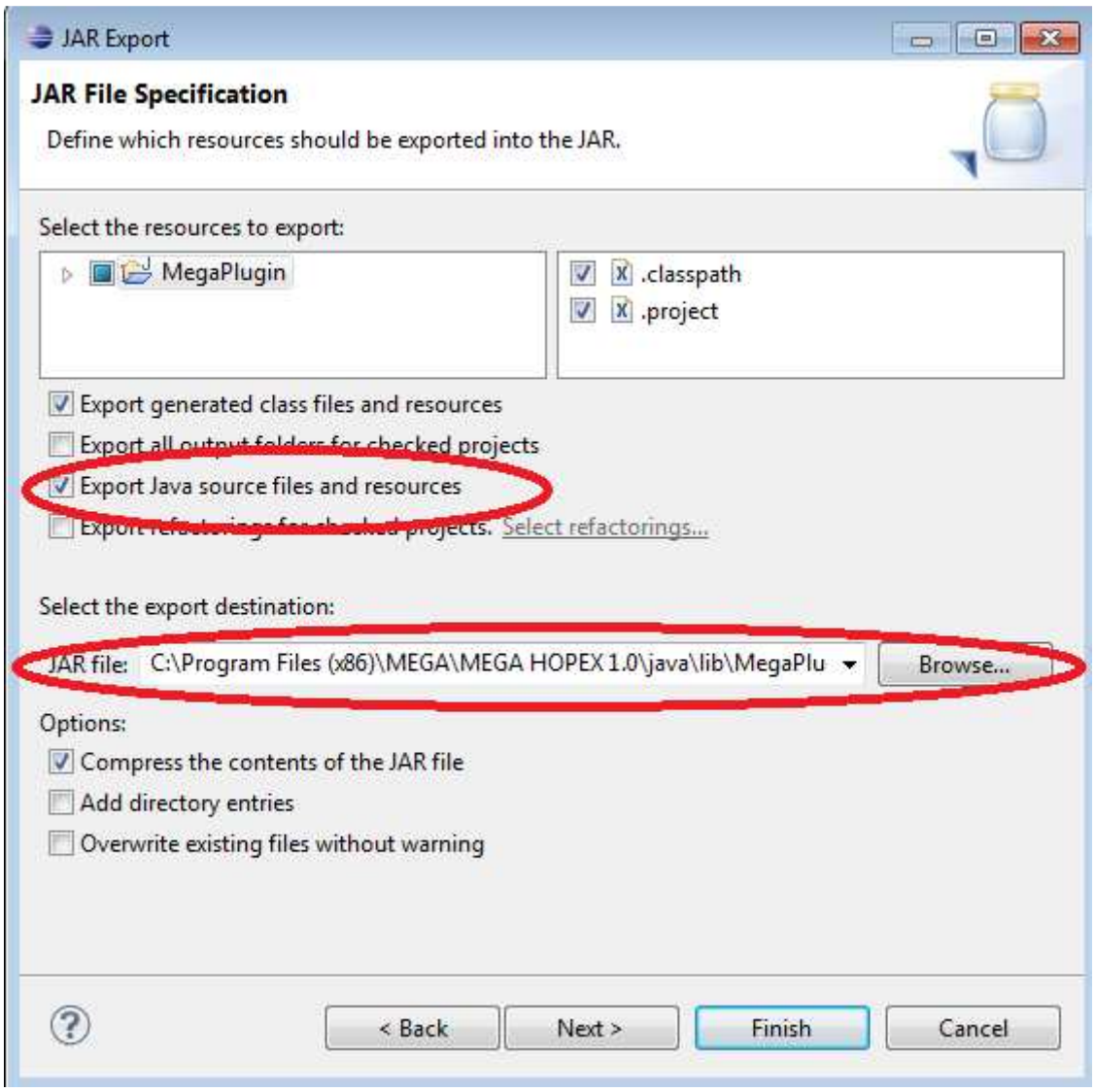
1. In the Java project, right-click MegaPlugin and select **Export**.



2. In the **Export** wizard, expand the **Java** folder and select **JAR file**.


Export
Select
Export resources into a JAR file on the local file system.
Select an export destination:
type filter text
General
Install
Java
JAR file
Javadoc
Runnable JAR file
Run/Debug
Tasks
Team
XML
< Back
Next >
Finish
Cancel

3. Click **Next**.



4. Make sure **Export Java source files and resources** is selected.
5. In the **Select the export destination** field, click **Browse** and indicate the location of the JAR file to be generated.
6. Click **Finish**.

Note 1: The JAR file must be generated (or copied after generation) in the "java\lib" directory of the MEGA installation site.

Note 2: It is important to export your Java source files. Indeed, once the project is delivered, the end user may encounter problems and call MEGA Technical Support. This will enable the Support to better diagnose the problem. Also, the customer may ask you to add additional customizations. It would be a shame to have to rewrite the full code. With this option, the code is kept and can be reused.

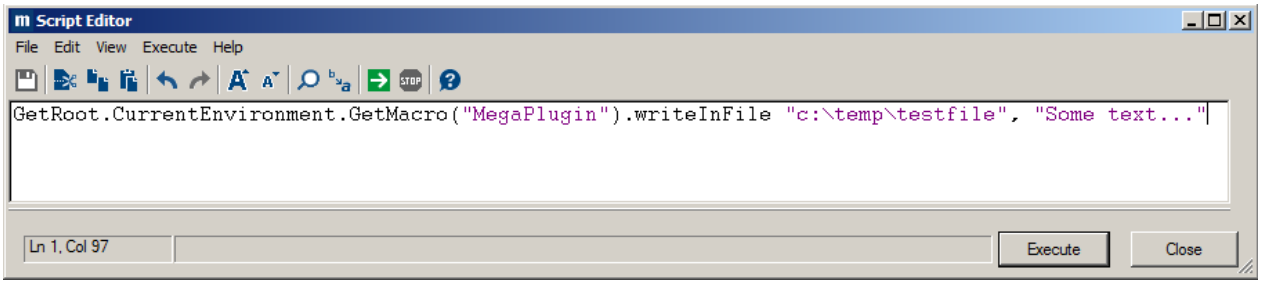
3.3.4 Executing the macro

When the JAR file has been deployed in the MEGA "java\lib" directory, the macro can be tested.

To test the macro:

1. In the MEGA script editor, perform the following command:

```
GetRoot.CurrentEnvironment.GetMacro("MegaPlugin").writeInFile "c:\temp\testfile", "Some text..."
```



3.4 Using the MEGA API in a component written in Java

This chapter is an introduction to implementation of Java MEGA APIs using the example of the creation of a "Command" that can be executed from a Business Process's contextual menu.

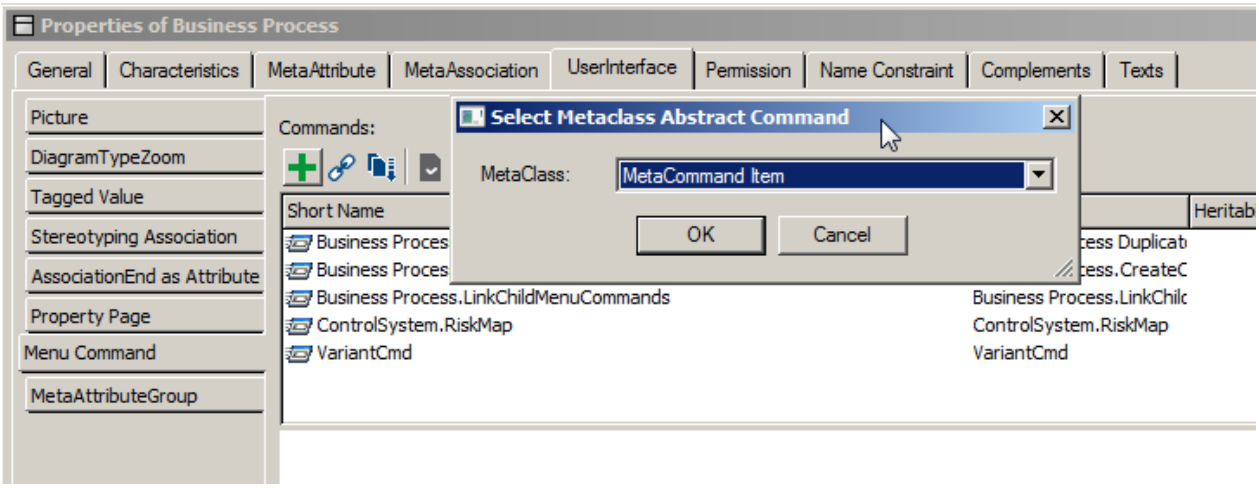
The effect of this Command is to export the hierarchy of Business Processes into a text file.

Note: The Java MEGA API is not detailed in this document. Java MEGA API documentation is provided in the MEGA installation "java\doc" directory and also accessible from MEGA menu bar (**Help > APIs > JavaDoc**).

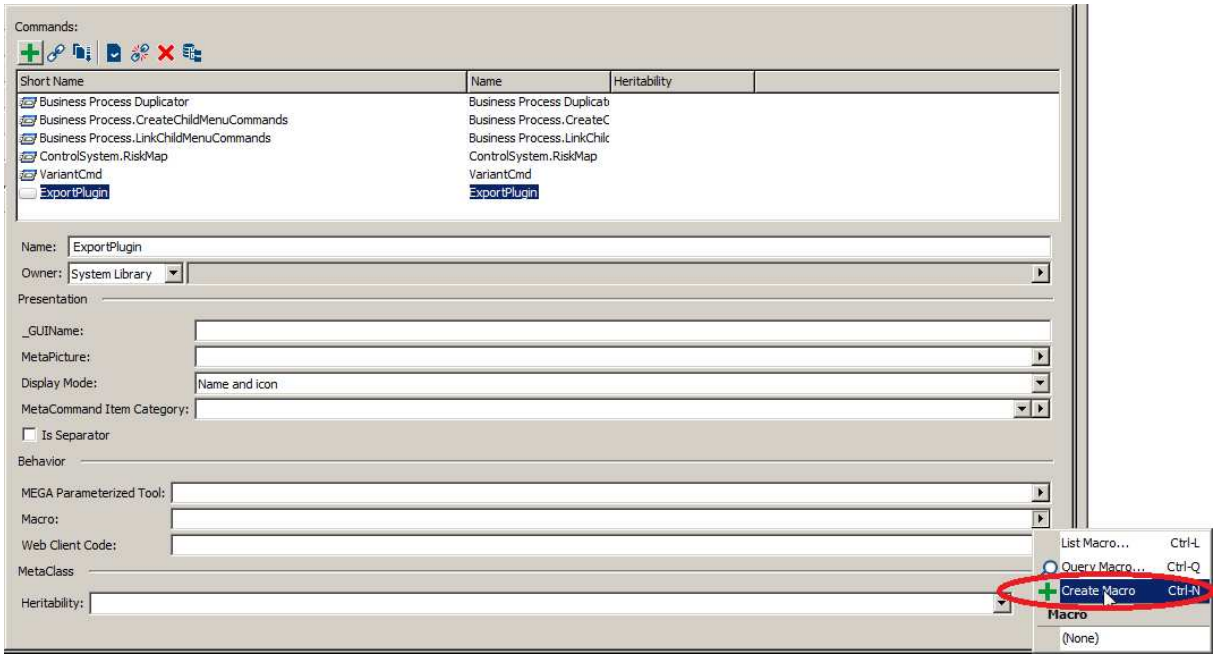
3.4.1 Creating the Command

To create the command:

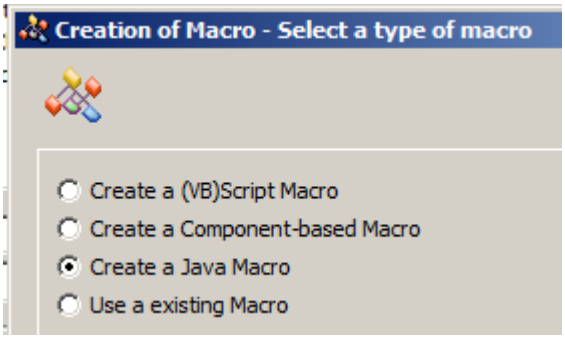
1. From **MEGA MetaStudio**, open the properties dialog box of the "Business Process" MetaClass.
2. In the **UserInterface** tab, select the **Menu Command** sub tab, and create a new MetaCommand Item named "ExportPlugin":



3. Create a macro from the "Macro" field of the MetaCommand:



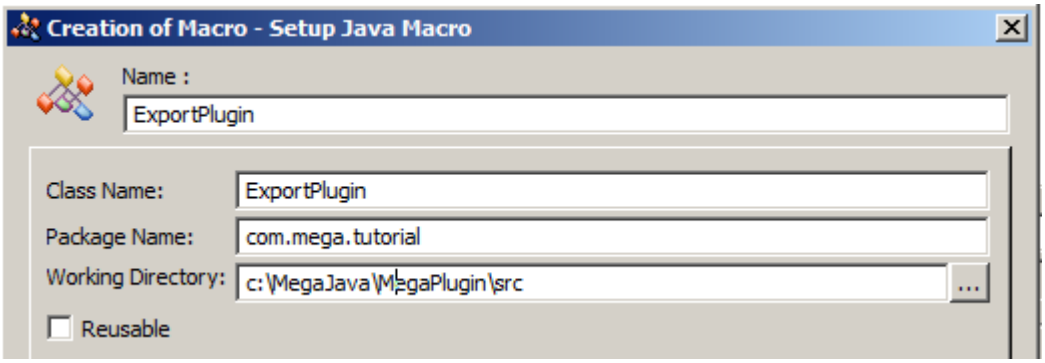
4. Select the creation of a Java macro:



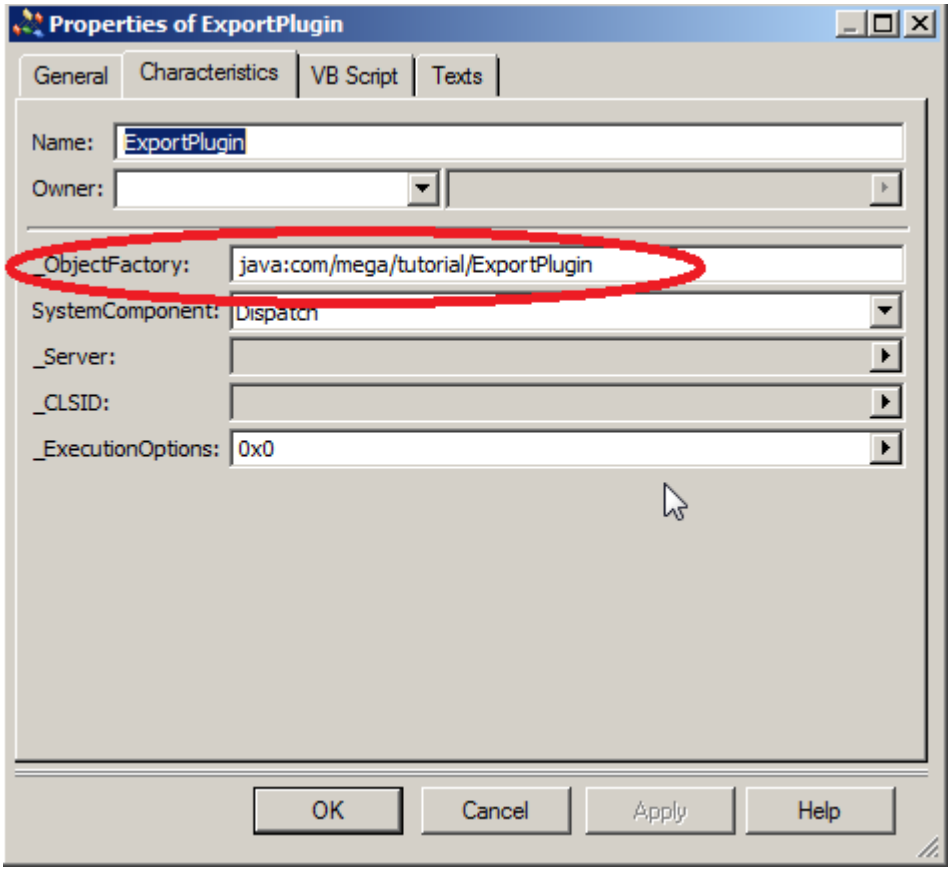
5. Name the macro "ExportPlugIn" and specify the following parameters:
   - "Class Name" identifies the Java class called, specify "ExportPlugin".

"Package Name" identifies the package of the Java class called, specify "com.mega.tutorial".

- "Working Directory" identifies the directory in which to generate the Java class initialized with the functions to be implemented.



6. Creation of the macro initializes the properties "_ObjectFactory" and "SystemComponent ". In the macro properties dialog box, the "_ObjectFactory" field is specified with the identifier of the MetaCommand implementation Java class ("java:*Package*/*Class*"):



7. The Java class is generated, according to the interface to be implemented, in the directory previously specified:

```
package com.mega.tutorial;
import com.mega.modeling.api.*;
public class ExportPlugin {
/**
```

```
    * Function called when the command is triggered from a MegaObject.
    * Example: click on a menu.
    *
    * @param mgobjSource
    *            MegaObject on which the command is applied.
    */
    public void InvokeOnObject(MegaObject mgobjSource, Object userData) {
    }
}
```

3.4.2 Configuring the Eclipse project for use of Java MEGA API

In this section, the Eclipse project « MegaPlugin » used in the chapter Creating an Eclipse project p. 26 is reused.

Referencing the Java MEGA API access component

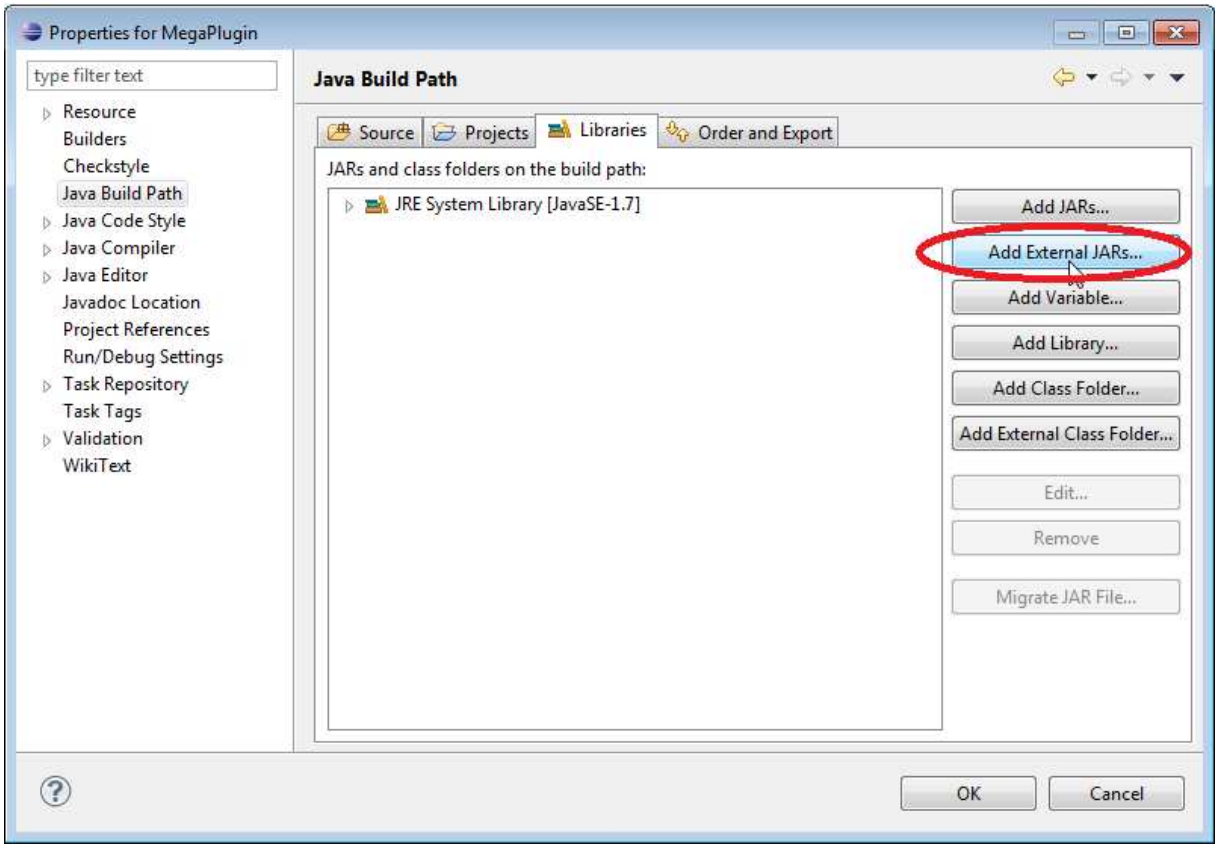
So that implementation of the MetaCommand Item can use Java MEGA APIs, the API access component must be referenced by the "MegaPlugin" Eclipse project.

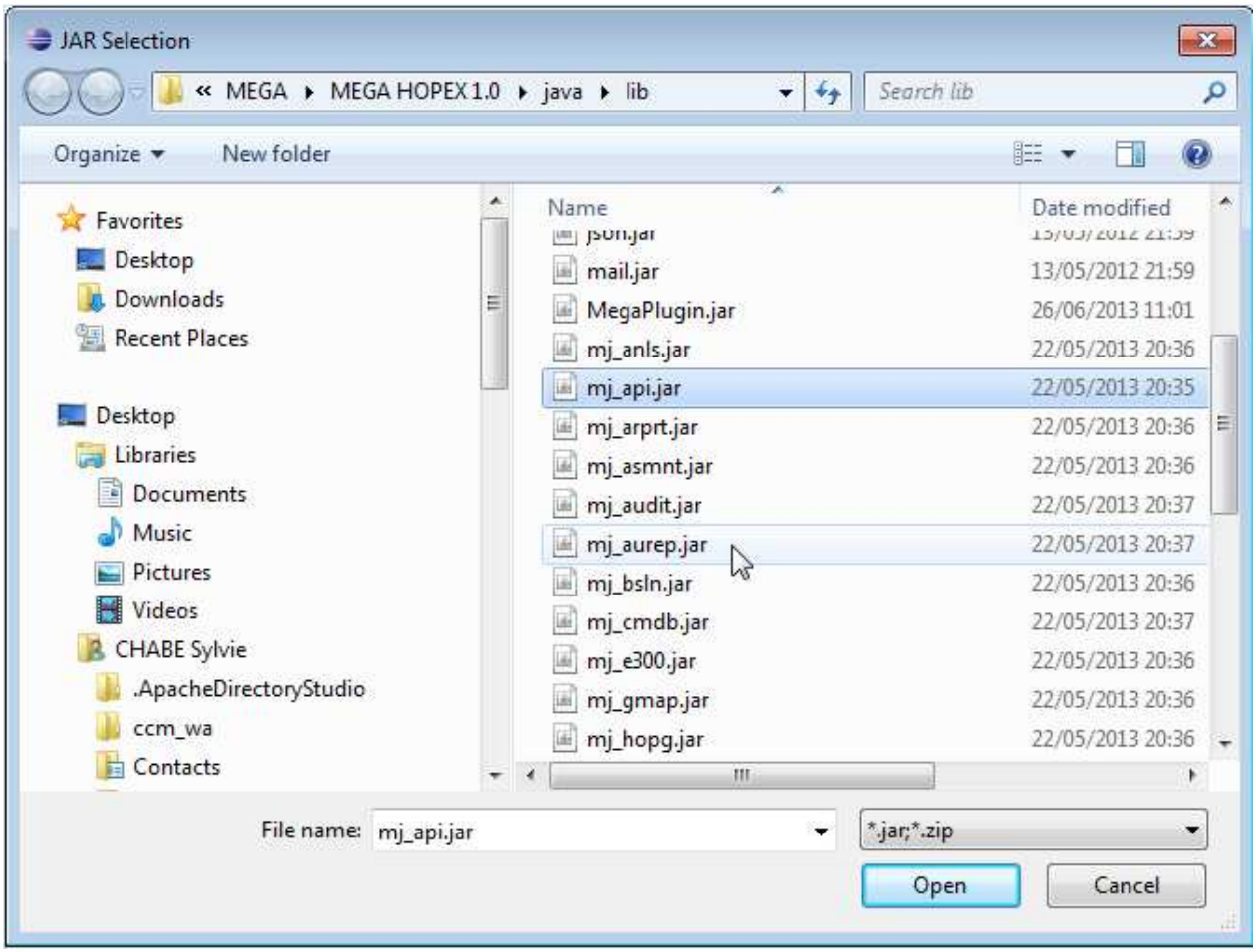
The "java\lib" directory of the MEGA installation site contains the Java components delivered by MEGA. Among these, file "mj_api.jar" provides access to Java MEGA API.

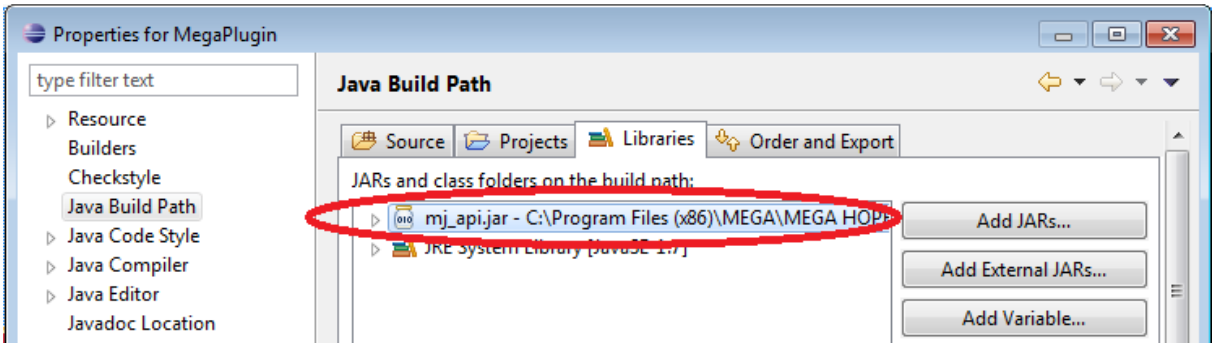
Note: As previously seen, customized Java components must also be deployed in the "java\lib" directory.

To reference the "mj_api.jar" component:

- In "Package Explorer" select "Properties" of the "MegaPlugin" project to open its properties dialog box.
- In the "Java Build Path" node, **Libraries** tab, add the reference to file "mj_api.jar":


Properties for MegaPlugin
type filter text
Resource
Builders
Checkstyle
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Project References
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
JRE System Library [JavaSE-1.7]
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
OK
Cancel


JAR Selection
« MEGA ▸ MEGA HOPEX 1.0 ▸ java ▸ lib
Search lib
Organize
New folder
Favorites
Desktop
Downloads
Recent Places
Desktop
Libraries
Documents
Music
Pictures
Videos
CHABE Sylvie
.ApacheDirectoryStudio
ccm_wa
Contacts
Name
Date modified
mail.jar 13/05/2012 21:59
MegaPlugin.jar 26/06/2013 11:01
mj_anls.jar 22/05/2013 20:36
mj_api.jar 22/05/2013 20:35
mj_arprt.jar 22/05/2013 20:36
mj_asmnt.jar 22/05/2013 20:36
mj_audit.jar 22/05/2013 20:37
mj_aurep.jar 22/05/2013 20:37
mj_bsln.jar 22/05/2013 20:36
mj_cmdb.jar 22/05/2013 20:37
mj_e300.jar 22/05/2013 20:36
mj_gmap.jar 22/05/2013 20:36
mj_hopg.jar 22/05/2013 20:36
File name: mj_api.jar
.jar;.zip
Open
Cancel



Integrating API documentation in the development environment

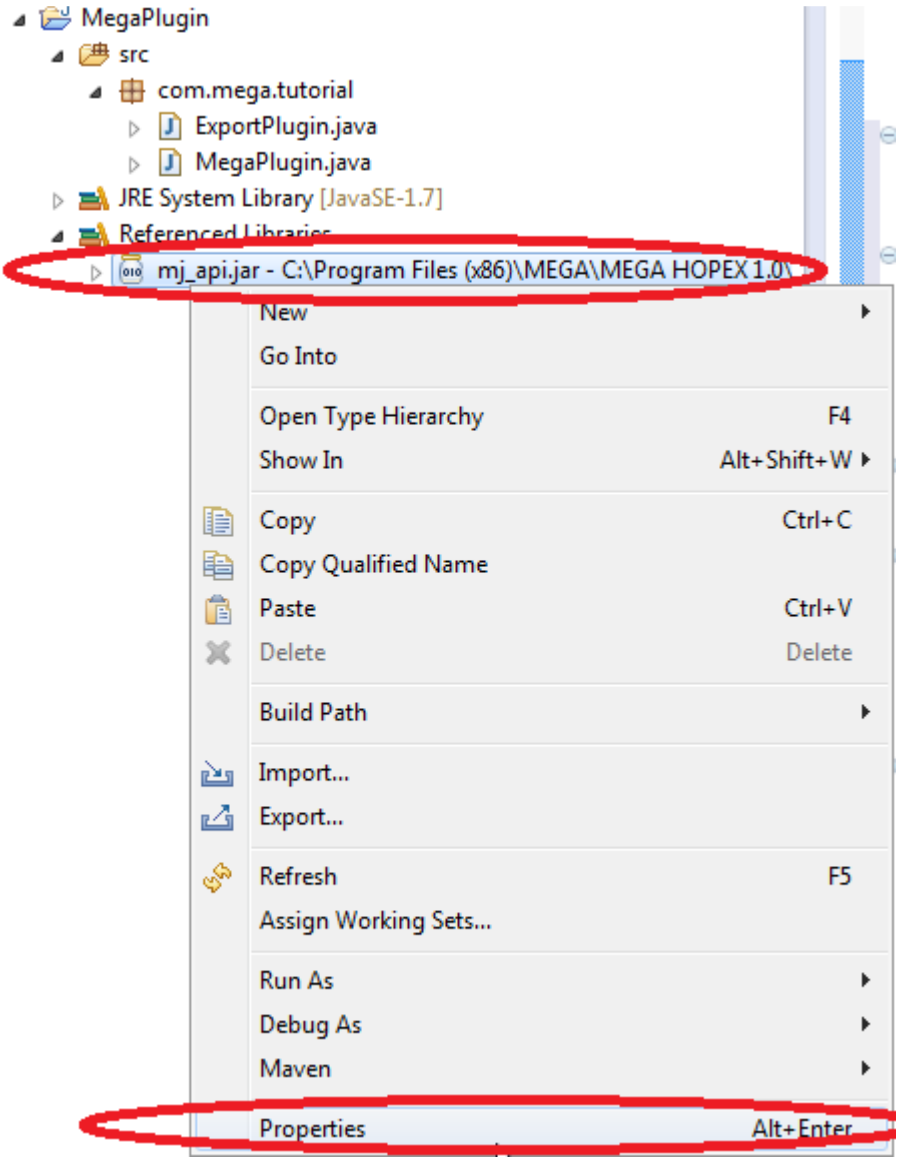
Java MEGA API documentation is delivered in the form of an archive, file "mj_api.zip" in the "java\doc" directory of the MEGA installation.

The archive contains documentation in HTML format generated by JavaDoc.

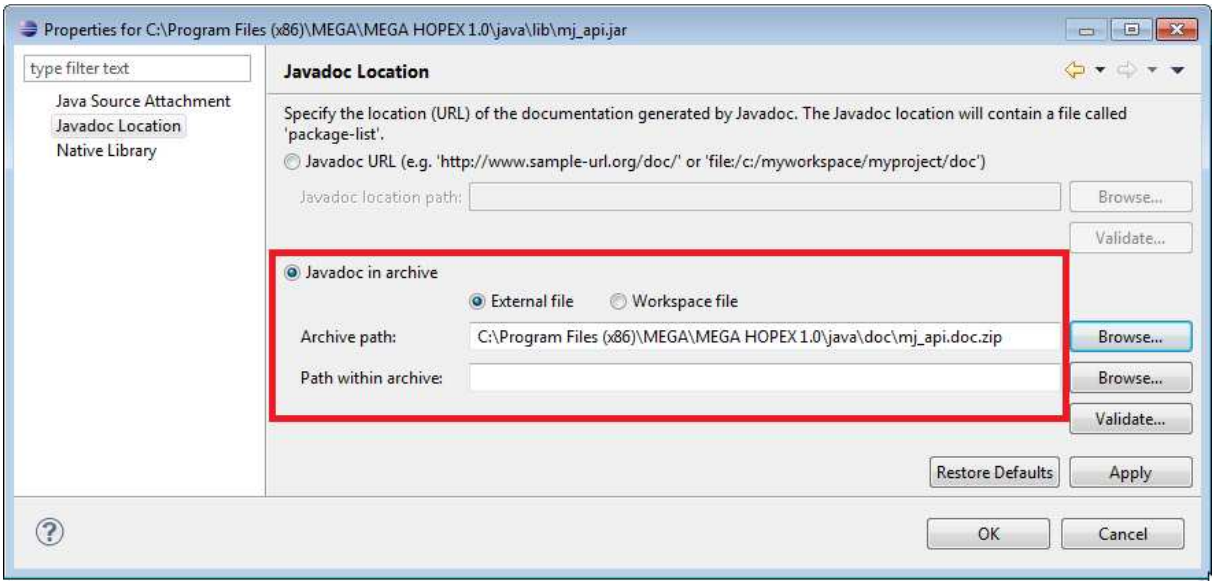
To benefit from total integration of Java MEGA API help, API documentation should be associated by referencing the file "mj_api.jar".

To associate JavaDoc help with the "mj_api.jar" component, specify the location of the archive containing help in the properties of the referenced "mj_api.jar" library:

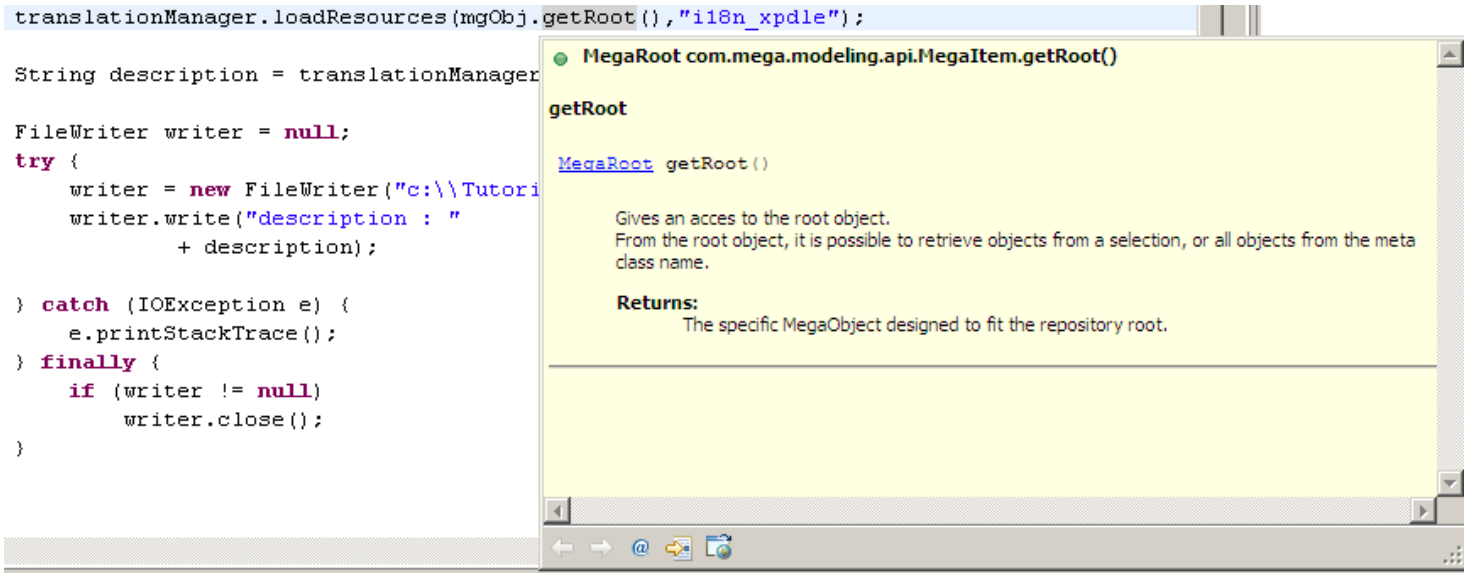
- Open the properties dialog box of library "mj_api.jar":



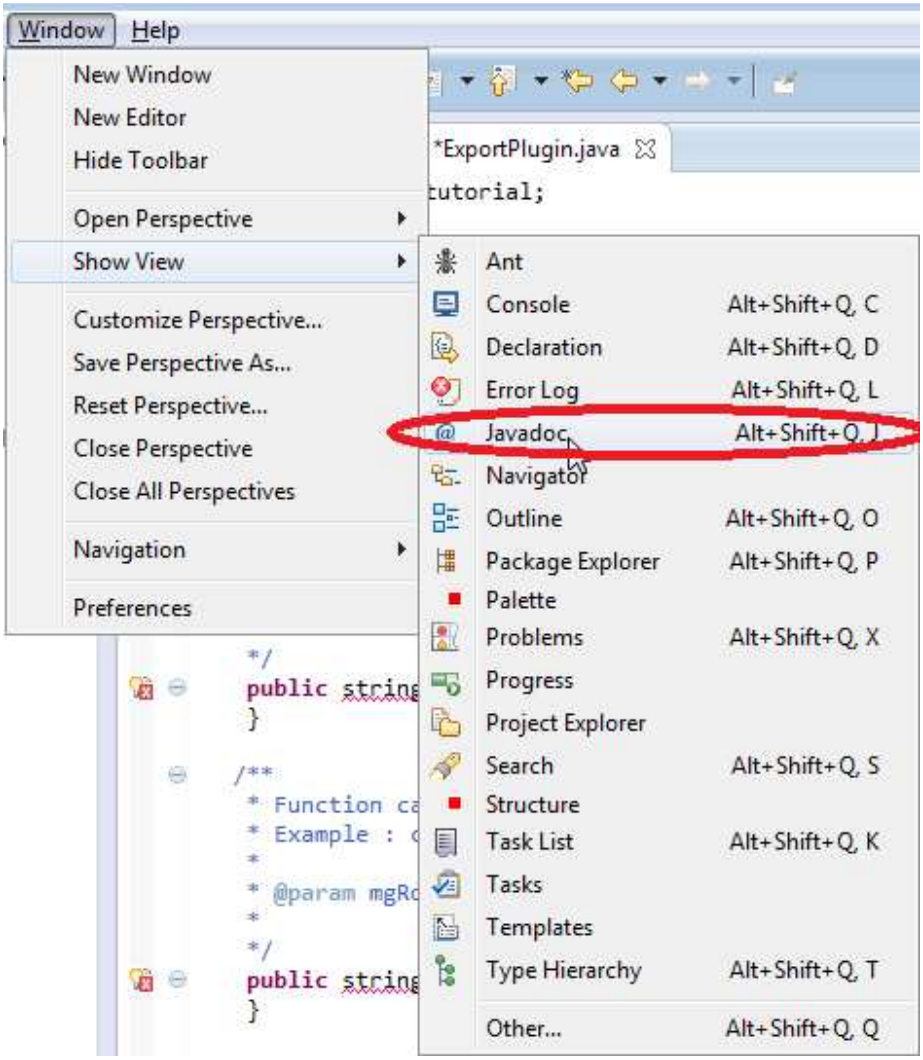
- In the node "Javadoc Location", specify the location of the JavaDoc help (MEGA API JavaDoc help is available as a compressed archive in the "java\doc" directory of the MEGA installation):



Once the JavaDoc help has been referenced, it may be used by positioning the cursor on documented keywords.



Javadoc help can be permanently displayed in a dedicated window. To display this window, select **Window** > **Show View** > **Javadoc**:



Generating identifiers for the metamodel used

MEGA APIs enable exploration and modification of models described in the MEGA repository.

The concepts (MetaClasses, MetaAssociationEnds, MetaAttributes, etc.) used through an API can be identified by their name or identifier.

Example finding sub-processes of a business process:

```
subProcesses = process.getCollection("Component");
```

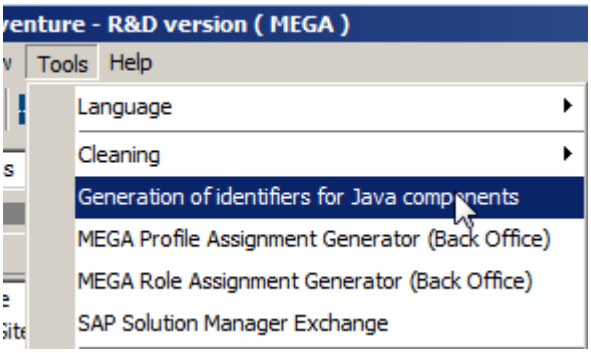
It is highly recommended to use the identifiers of concepts to ensure compatibility with languages and potential future metamodel modifications.

The example above can therefore be written as:

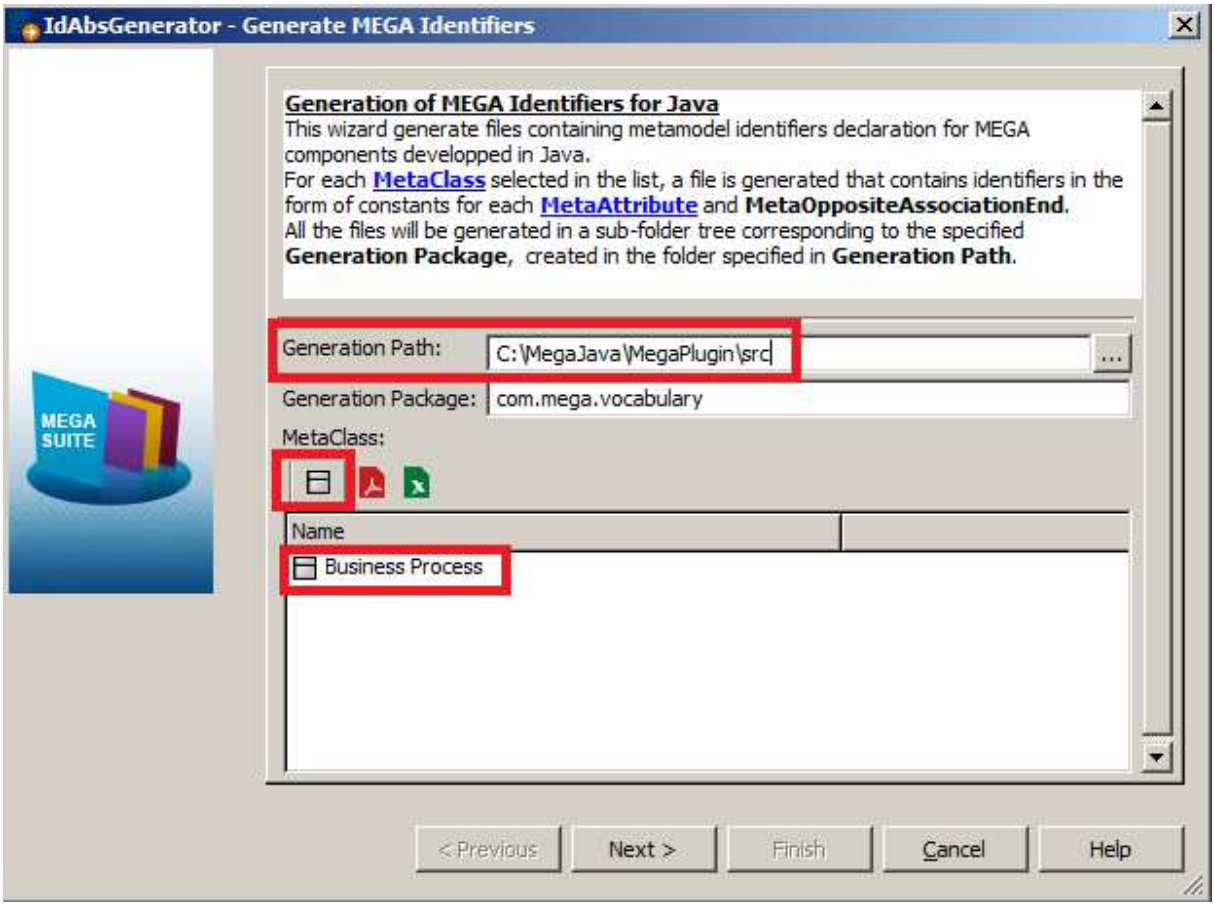
```
// using the identifier in its Absolute Identifier (IdAbs) format
subProcesses = process.getCollection("8l)gvmQ9pKE0");
// using the identifier in its hexadecimal (_HexaIdAbs) format
subProcesses = process.getCollection("ABFBAC39332500E5");
// using the identifier in its MEGA field format
subProcesses = process.getCollection("~8l)gvmQ9pKE0[Component]");
```

A wizard enables generation of classes defining identifiers to be used for a set of MetaClasses. This wizard is available with the MetaStudio product for users with "Expert" metamodel access.

To execute this wizard, select command "Generation of identifiers for Java components" accessible in the **Tools** menu of the MEGA workspace:

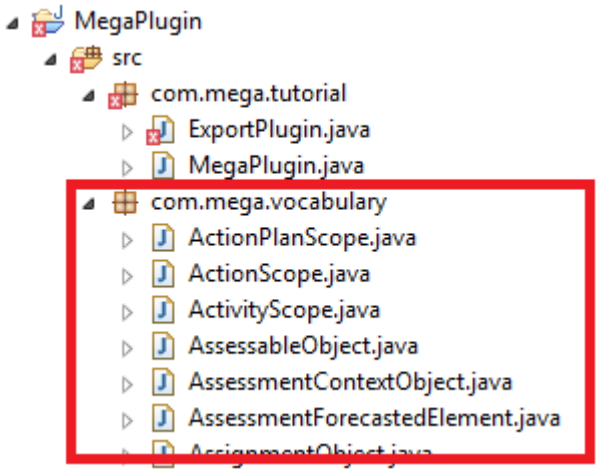


When the wizard is displayed, specify the Java classes generation path as well as the MetaClasses for which identifiers must be generated:



In our example, files will be generated in the directory containing the sources: "C:\MegaJava\MegaPlugin\src".

A package "com.mega.vocabulary" is created containing classes corresponding to declarations of selected MetaClasses as well as their inherited MetaClasses:



3.4.3 Implementing the MetaCommand Item

```
package com.mega.tutorial;

import com.mega.modeling.api.*;
import com.mega.vocabulary.*;
import java.io.FileWriter;
import java.io.IOException;


public class ExportPlugin {

        private static String filePath = "C:\\Temp\\BusinessProcess.txt";


   /**
    * Function called when the command is triggered from a MegaObject.
    * Example: click on a menu.
    *
    * @param mgobjSource
    *            MegaObject on which the command is applied.
    */
    public void InvokeOnObject(MegaObject mgobjSource, Object userData) throws IOException
    {
            FileWriter writer = null;
        MegaObject process = null;

        // Source object is a process
        process = mgobjSource;

        // Open a FileWriter with file name defined statically in the class
                writer = new FileWriter(filePath);

                // Writes the source process name
                writer.write(process.getProp(BusinessProcess.MA_GenericLocalName) + "\r\n");

            // Get the collection of sub-processes
                // Uses: MegaObject.getCollection function with the identifier
                //        of sub-process generated in the com.mega.vocabulary package
                MegaCollection subProcessesCollection = null;
                subProcessesCollection = process.getCollection(BusinessProcess.MAE_Component);

            // Search for all the sub-processes
                for (int subProcessesIndex = 1;
                        subProcessesIndex < subProcessesCollection.size() + 1;
                        subProcessesIndex++)
                {
                       MegaObject subProcess = null;

                 // Get the sub process in the collection at subProcessesIndex position
                 subProcess = subProcessesCollection.get(subProcessesIndex);
```

```
                        // Writes the sub-process name
                        writer.write("    " + subProcess.getProp(BusinessProcess.MA_GenericLocalName) +
"\r\n");
                }
                if(writer!=null)
                {
                        writer.close();
                }
    }

}
```

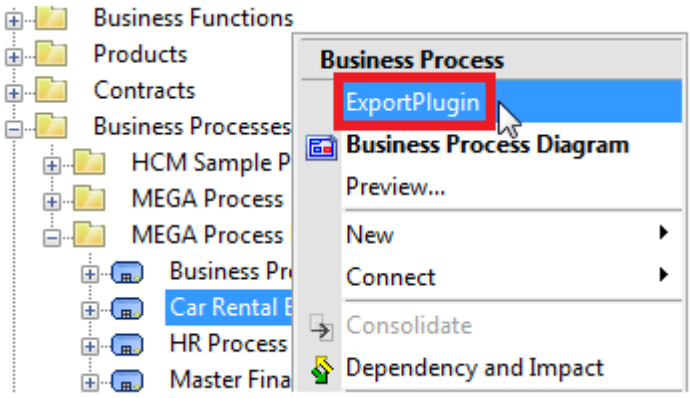
Compiling and deploying

Follow the Compiling and deploying the Java component procedure described in the Calling a component written in Java from MEGA section p. 23.

If changes are taken into account, after Java code modification, regenerate the Jar file and copy it in the MEGA java/lib directory, then exit and restart MEGA.

Using the MetaCommand Item

To execute the MetaCommand Item, click the new command "ExportPlugin" in the contextual menu of a Business Process:



The result is the creation of a file in the directory c:\temp:

```
Car Rental Business
   Provide Rental Cars
```

3.5 Debugging a Java component called from MEGA

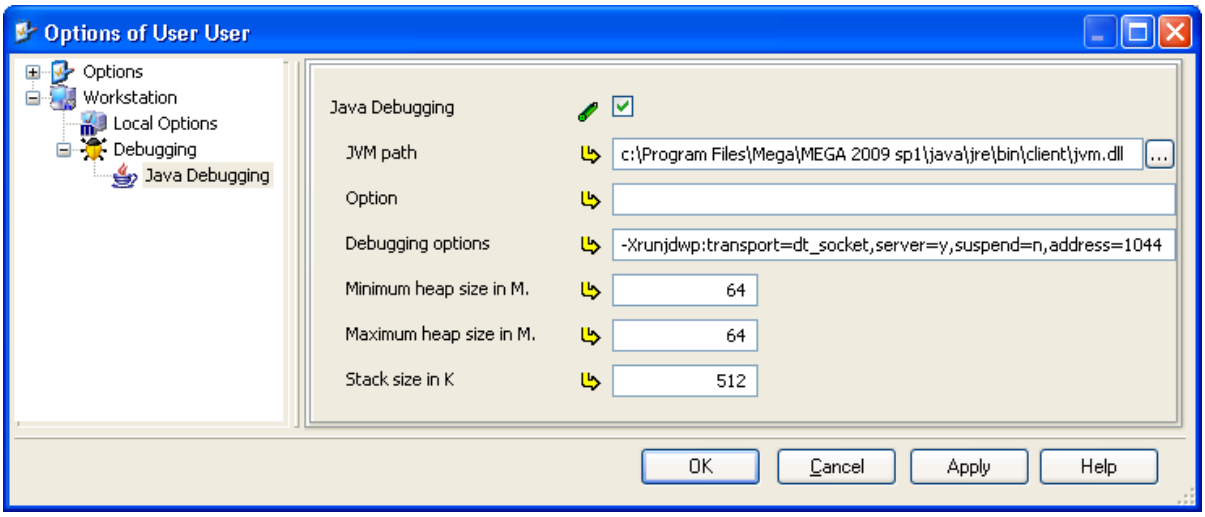
Debugging a Java component called from MEGA uses remote access mode to the Java virtual machine.

This chapter explains how to use this mode of debugging with MEGA.

Configuration

MEGA options

To activate Java component debugging, define "Java Debugging" options accessible in group "Workstation" > "Debugging":

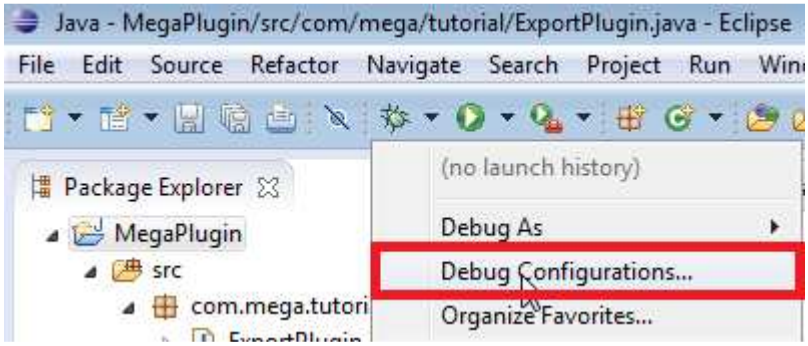


Options are:

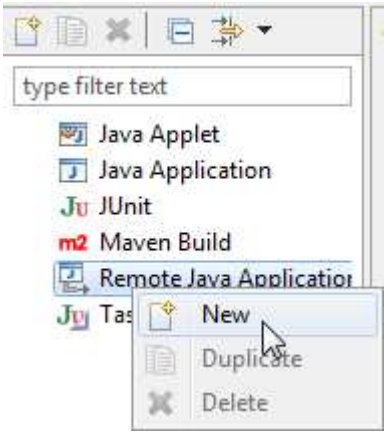
- *JVM path*: Specify the location of the Java virtual machine to be used.
- *Option*: Enables addition of virtual machine execution options.
- *Debugging options*: Debugging options.
- *Heap and stack size*: The following options enable the specification of available memory sizing.

Debugging configuration in Eclipse

From the Eclipse project, open the debugging configuration dialog box (right-click the project and select **Debug As** > **Debug Configurations**):

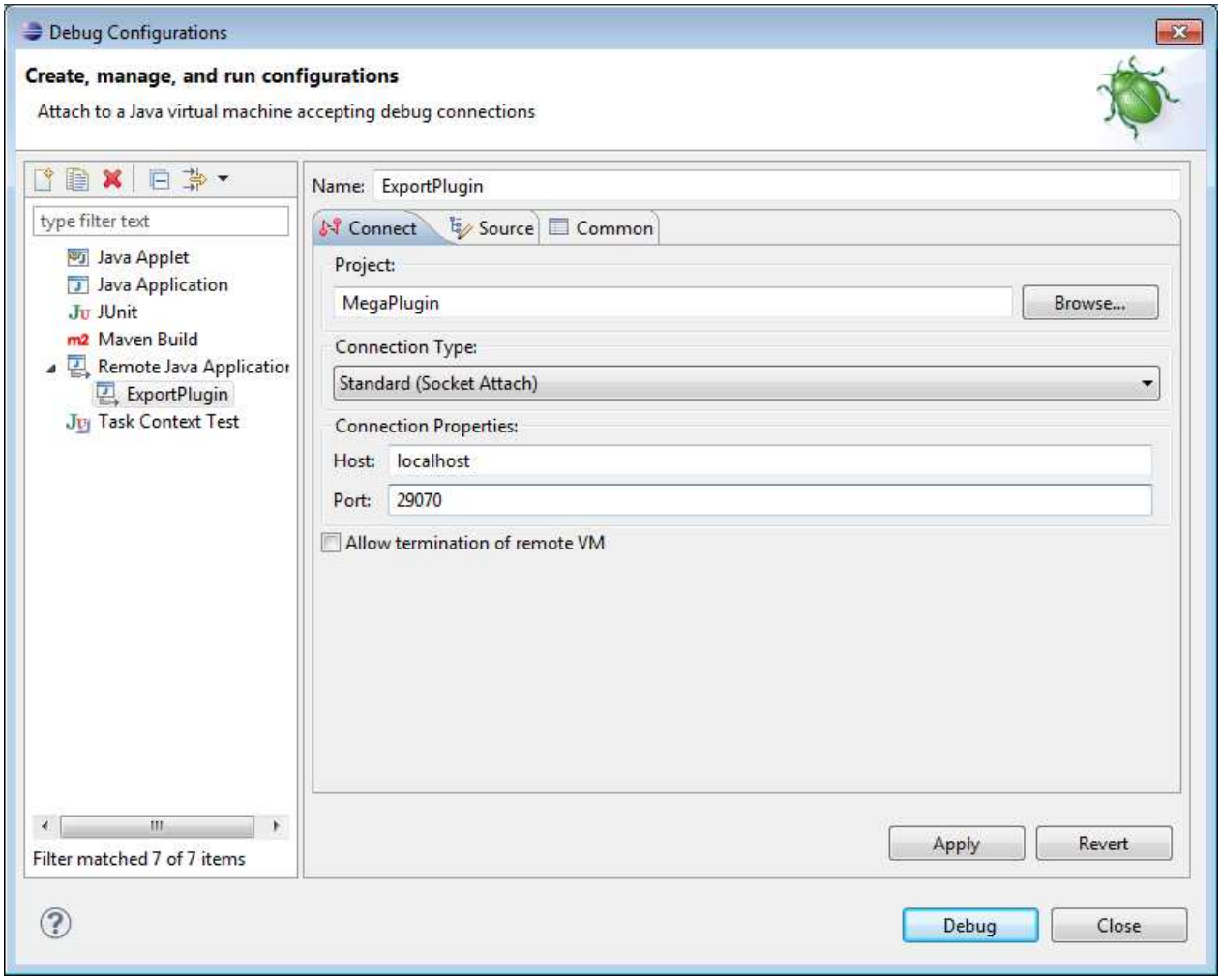


Create a new Remote Java Application debug configuration



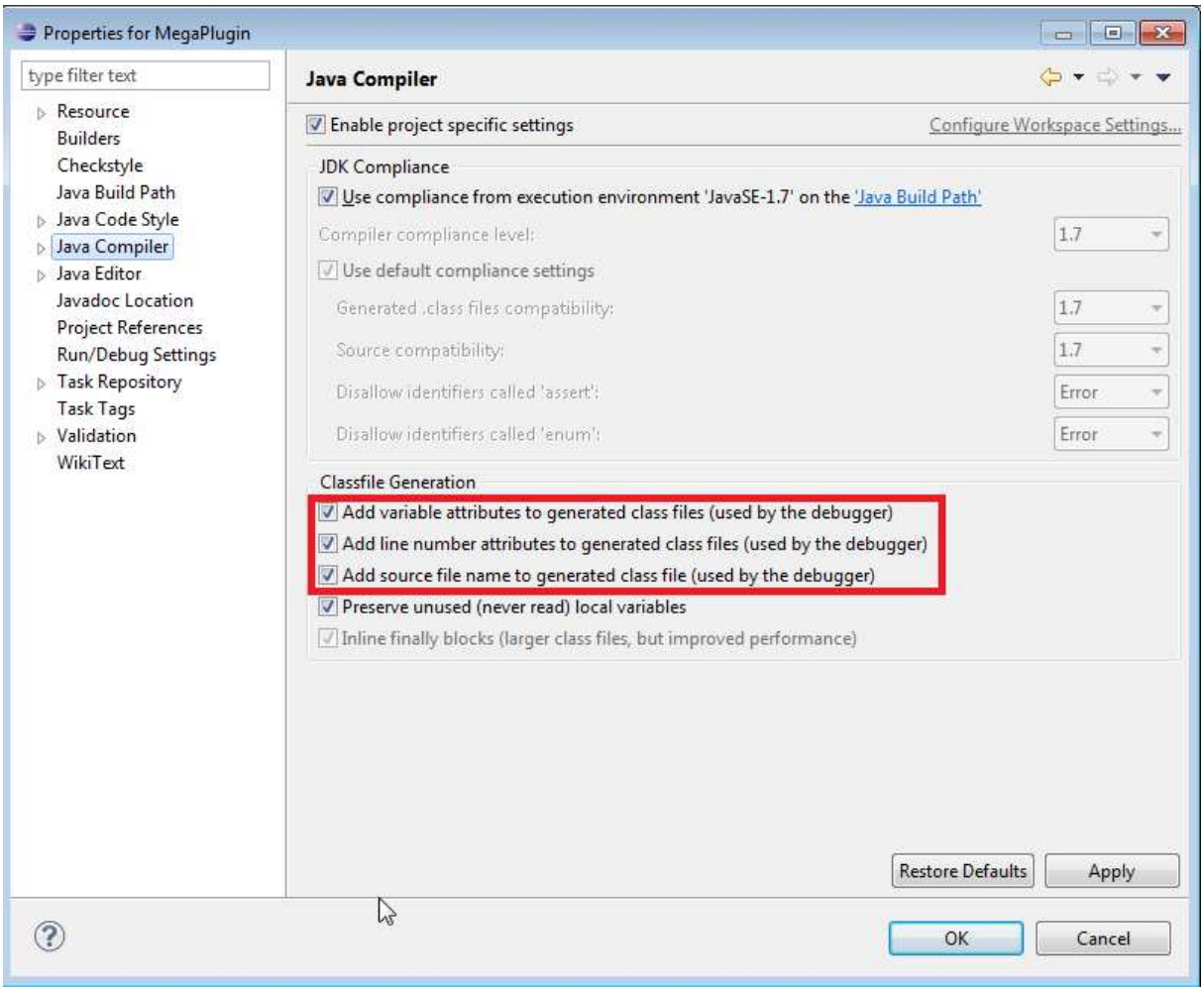
Specify parameters enabling debugging by socket attachment:

- *Connection Type*: Standard (Socket Attach).
- *Connection Properties*: host and port enabling access to the JVM used by MEGA.



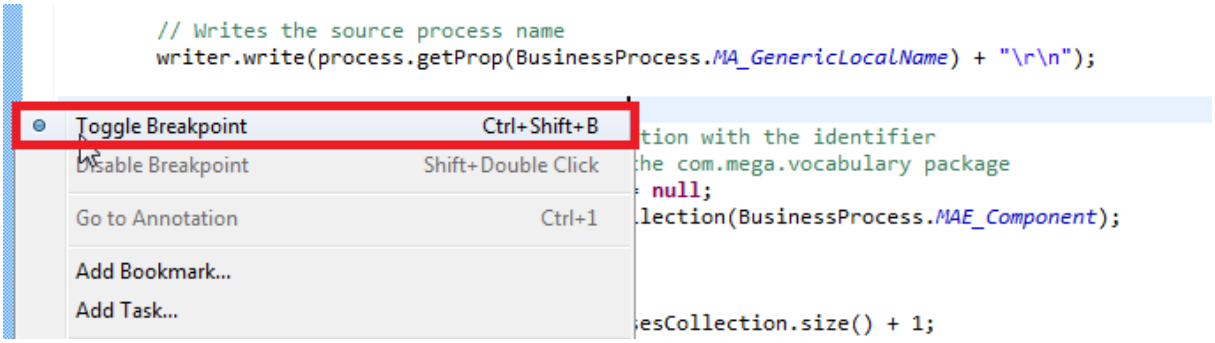
Compiling the project

Check that debugging options are activated:

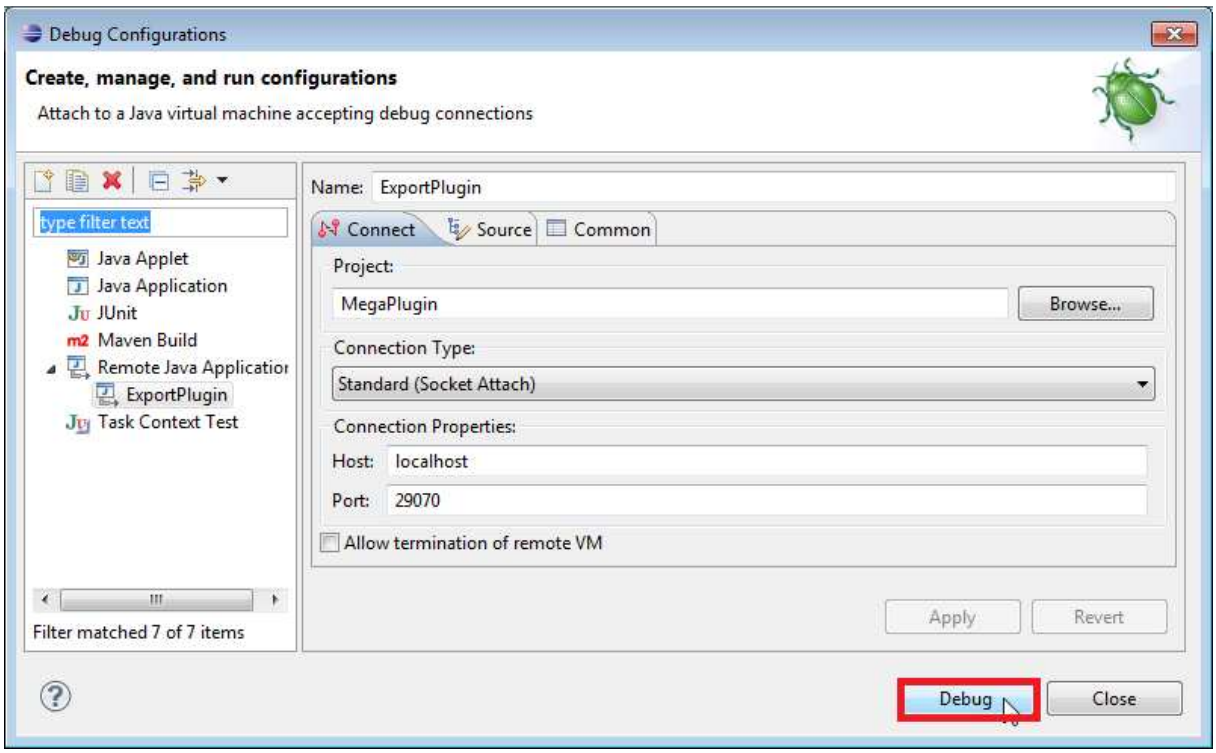


Debugging

Add breakpoints where necessary.



Execute the project in debug mode:



Note: If debugging is not correctly configured, MEGA can suddenly close. It is necessary to ensure that the correct port is used, and that the Firewall configuration authorizes access to this port (for example, try port 1044 if port 29070 does not permit debugging).

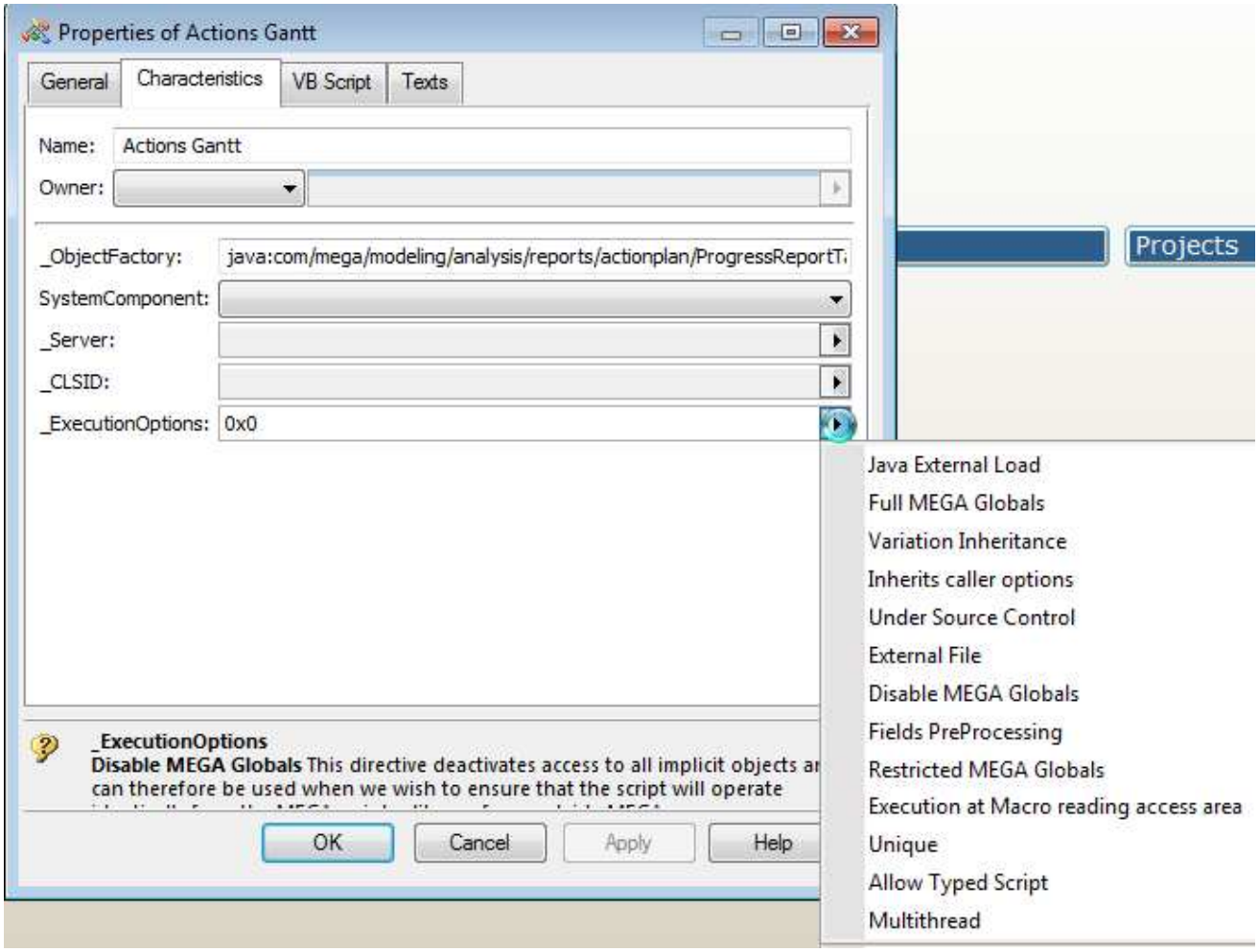
4 MACRO PARAMETERIZATION

To access macros stored in the repository:

1. Display the **Utilities** navigation window.
2. Expand the **Macros** folder.

The **Characteristics** tab of the macro properties dialog box enables definition of macro execution specificities.

The end user can parameterize the macro to define specific behaviors. The parameterization can be made on the _ExecutionOptions Macro Attribute.



This parameterization is important since it may improve performances of the full application.

The following options are available:

Full MEGA Globals: This parameter activates the global variables such as MegaRoot, MegaDB, MegaToolkit, MegaEnv, MegaSite. The access to the Mega Root is implicit. This parameter is deprecated. It is highly recommended not to use it. Setting this attribute to this property means that your macro is not compatible with MEGA Web applications.

Variation inheritance: This parameter is set if the GetCollection and GetSelection contained in the macro must retrieve both the direct objects retrieved from a MetaAssociationEnd and the objects obtained from inheritance with the variation mechanism.

Inherits Caller Options: This applies if the current macro has been called by another macro. In that case, the current macro inherits from the variation rights defined on the calling macro. As a sample, if the calling macro has the Variation inheritance value set, then the called macro has the Variation inheritance value set.

External File: The macro is stored in an external VBS file.

Disable MEGA Globals: This option defines that the use of MEGA Globals is forbidden. This is the recommended behavior. The use of Globals does not work with the web applications.

Fields PreProcessing: This option enables to use directly the fields (MetaAttribute, MetaAssociationEnd) on an object without using GetProp and GetCollection. This option is not really recommended since it is time consumed and decreases the performances. This option is now deprecated.

Restricted MEGA Globals: This option enables the use of two MEGA globals: MegaEnv and MegaToolkit. This parameter is deprecated. It is highly recommended not to use it. The use of Globals does not work with the web applications.

Execution at Macro reading access area: This option defines at which confidentiality level the macro must be executed. The macro contains code and algorithm. The MEGA Philosophy concerning confidentiality is to compute algorithm as if you can access to all the objects in order to have the good result and to hide the confidential objects only for display. Therefore, if the macro is dedicated to display data, this option does not need to be used but as soon as the macro contains algorithm, it is highly recommended to ask for the following question : "do I need to access to all the data to have a coherent result ?". If the answer is "yes", this option must be activated.

For more information, see Confidentiality section p. 206.

Unique: This parameter is used for performance purpose. If your script has no global, you may require to instantiate it only once. In that case, the script is loaded only once and executed as much as requested.

Allow Typed Script: This option gives you the availability to type the objects inside your script. For example, in standard, if you declare a function you can write:

```
Function MyFunction( oMegaObject)
Dim myObject
End Function
```

If this option is set, you can type the objects inside your code as the following:

```
Function MyFunction( oMegaObject As MegaObject )
Dim myObject As MegaObject
End Function
```

The interesting point concerning that is that you can use the intellisense on object.

Multithread: This option is available only for Java Macro. If this option is set, it means that the java component works in multhreading that is to say the component may be called at the same time by several threads.

Java External load (until 740): This parameter is for Java debugging purpose. This parameter can be set only if the Macro is a Java plug-in. It launches a new MEGA process which enables to take into account the new compiled version of the Java plug-in without restarting MEGA (Once a Java plug-in is loaded, replacing the jar file has no effect, you must stop and restart MEGA to get the effect of the new java plug-in version).

Under Source Control (until 740): This option is dedicated to MEGA R&D Team and cannot be used by the end user.

5 API USE PRINCIPLES

5.1 Coding: the right way

5.1.1 MegaRoot

You can use the **Script Editor** to write:

- scripts for testing

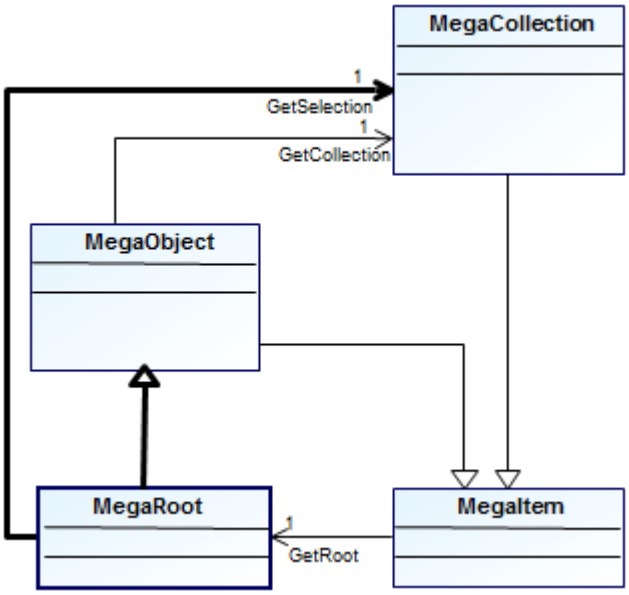
  As these scripts are not stored in MEGA you can use global variables like MegaDB, MegaRoot or GetRoot to access objects in the repository. This coding way is dedicated to testing use only.

- scripts stored as files or macros in MEGA.

Do not use MegaDB, MegaRoot, and Get Root in macro scripts stored in MEGA.

MegaRoot:

- represents the session you are connected with.
- gives access to MEGA repository data through **GetGollection** or **GetSelection**



In the following sections, **oRoot** is:

- a session with which you are connected
- either set as a parameter of the function (e.g.: `Main(oRoot)`) or you call a function that has as a parameter the MegaObjects.

Here is sample code to access the **MegaRoot** from an object or a MegaEnvironment:

```
Set oRoot = object.GetRoot


Function Sub Main(oRoot)
'code
End Sub
```

5.1.2 MegaFields

You must make reference to an object using its absolute identifier rather than its name: in this case the code is most highly optimized and resists renaming of the instance as well as change of language.

To write a query in a script you want to store in MEGA, use of MegaFields is mandatory.

An object in **MegaField** format is as follow:

```
"~xxxxxxxxxxxx[Object Name]"
```

Example for Project object:

```
~qekPESs3iG30[Project]"
```

Where:

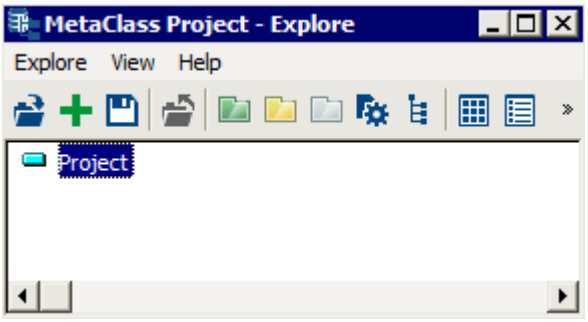
- ~ indicates the start of the MegaField
- **qekPESs3iG30** is the absolute Identifier of the object
- **[Project]** includes the object name

For more information on MegaFields see:

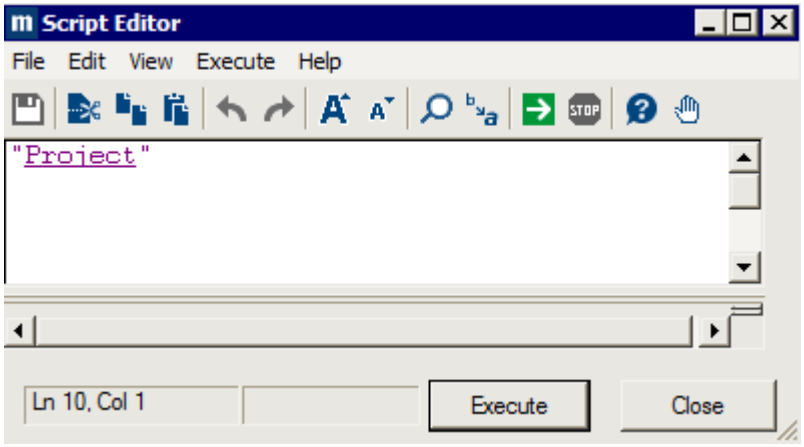
- MegaFields p. 170
- Handling Identifiers p. 199
- Navigating through data with APIs p. 217.

To enter Project MetaClass in MegaField format (i.e. with its absolute identifier) in your code:

1. From **Explore** menu bar, select **Explore > object**.
2. In the object list, select **MetaClass**.
3. In the right-pane pane select **Project**.
4. Click **OK**.



5. Right-click **Project** and select **Copy** and **Paste** it in your code.



In **Script Editor**, "Project" includes Object name (Project) and its absolute identifier (qekPESs3iG30):

```
"~qekPESs3iG30[Project]"
```

5.2 Basic Operations

Objects are entered in their MegaField formats, which include their absolute identifiers (e.g.: "~qekPESs3iG30[Project]" instead of "Project", see MegaFields section p. 53).

5.2.1 Creating an object

The following script enables creation of a project by applying the **Create** method to the **Project** object.

The name of the project created is indicated between brackets.

Example: Creation of a new project called "P1".

VB Script

```
Set oRoot = object.GetRoot
oRoot.getCollection("~qekPESs3iG30[Project]").Create ("P1")
```

Java

```
mgRoot.getCollection("~qekPESs3iG30[Project]").create("P1");
```

5.2.2 Connecting operations to a project

Example: Creation of new operations that are connected to project "P1".

VB Script

```
Set oRoot = object.GetRoot
Set oProject = oRoot.getCollection("~qekPESs3iG30[Project]").Item("P1")
Set oOperations = oProject.GetCollection("~OsUiS9B5iiQ0[Operation]")
For i = 1 To 10
oOperations.Create "ope-" & i
Next
For Each oOperation in oOperations
oRoot.Print oOperation.getProp("~Z20000000D60[Short Name]")
Next
```

Java

```
MegaObject mgobjProject =
mgRoot.getCollection("~qekPESs3iG30[Project]").get("P1");
MegaCollection mgcolOperations =
mgobjProject.getCollection("~OsUiS9B5iiQ0[Operation]");
for (int j = 1; j <= 10; j++) {
mgcolOperations.create("ope-" + i);
}
for (MegaObject mgobjOperation : mgcolOperations) {
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgobjOperation.getProp("~Z20000000D60[Short Name]"));
}
```

Code description:

```
Set oRoot = object.GetRoot
Set oProject = oRoot.getCollection("~qekPESs3iG30[Project]").Item("P1")
```

➔ retrieves the project called "P1" and its assignment to the variable "oProject".

Use the long name with "item" when manipulating namespaced objects. Example for an entity (DM) "titi" under the datamodel "DM1":

```
oEnt =oRoot.GetCollection("idabs[Entity (DM)].item("DM1::titi")
```

```
Set oOperations = oProject.GetCollection("~OsUiS9B5iiQ0[Operation]")
```

➔ retrieves all operations connected to the project "P1".

```
For i = 1 To 10
oOperations.Create "ope-" & i
Next
```

➔ creates ten operations (called "ope-1"..."ope-10") while adding them to the collection of operations connected to project "P1".

```
For Each oOperation in oOperations
oRoot.Print oOperation.getProp("~210000000900[Name]")
Next
```

➔ displays the names of the operations in the collection "oOperations".

5.2.3 Modifying a project name

Example: changing the name of project "P1" to "My Project".

VB Script

```
Set oRoot = object.GetRoot
Set oProject = oRoot.getCollection("~qekPESs3iG30[Project]").Item("P1")
oProject.GetProp("~210000000900[Name]") = "My Project"
```

Java

```
MegaObject mgobjProject =
mgRoot.getCollection("~qekPESs3iG30[Project]").get("P1");
                mgobjProject.setProp("~210000000900[Name]", "My Project");
```

Code description:

```
Set oRoot = object.GetRoot
Set oProject = oRoot.getCollection("~qekPESs3iG30[Project]").Item("P1")
```

➔ retrieves the project "P1" and assigns it to the variable "oProject".

```
oProject.GetProp("~210000000900[Name]") = "My Project"
```

➔ assigns the "My Project" value to the "Name" property of project "P1".

Note that `Setprop` is equivalent to `getProp`.

5.2.4 Displaying the names of all repository projects in a window

Example:

VB Script

```
Set oRoot = object.GetRoot
Set oAllProjects = oRoot.getCollection("~qekPESs3iG30[Project]")
For Each oProject in oAllProjects
sText = sText & oProject.getProp(“~Z20000000D60[Short Name]”) & vbCrLf
Next
MsgBox sText
```

Java

```
MegaCollection mgcolAllProjects =
mgRoot.getCollection("~qekPESs3iG30[Project]");
              String strText = "";
              for (MegaObject mgobjProject : mgcolAllProjects) {
                strText = strText + mgobjProject.getProp("~Z20000000D60[Short
Name]") + "\n";
              }
              mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", strText);
```

Code description:

`Set oRoot = object.GetRoot`

`Set oAllProjects = oRoot.getCollection("~qekPESs3iG30[Project]")`

➔ retrieves all repository projects and assigns them to the variable oAllProjects.

```
For Each oProject in oAllProjects
sText = sText & oProject.getProp(“~210000000900[Name]”) & vbCrLf
Next
```

➔ obtains the names of each project inserting a line return between each ("`vbCrLf`"). After having browsed the collection of repository projects, the variable `sText` contains the names of all projects, each on a new line.

`MsgBox sText`

➔ displays in a window the text contained in `sText` variable.

5.2.5 Assigning an attribute value to several objects

Example:

VB Script

```
Set oRoot = object.GetRoot
Set oOperations = oRoot.getCollection("~qekPESs3iG30[Project]").Item("My
Project"). getCollection("~OsUiS9B5iiQ0[Operation]")
For Each oOperation in oOperations
oOperation.Importance = "P"
Next
```

Java

```
MegaCollection mgcolOperations =
mgRoot.getCollection("~qekPESs3iG30[Project]").get("My
Project").getCollection("~OsUiS9B5iiQ0[Operation]");
              for (MegaObject mgobjOperation : mgcolOperations) {
                mgobjOperation.setProp("Importance", "P");
              }
```

Code description:

```
Set oRoot = object.GetRoot
Set oOperations = oRoot.getCollection("~qekPESs3iG30[Project]").Item("My Project").
getCollection("~OsUiS9B5iiQ0[Operation]")
```

➔ retrieves the operations connected to the "My Project" project and assigns this collection to the "oOperations" variable.

```
For Each oOperation in oOperations
oOperation.Importance = "P"
Next
```

➔ assigns the internal value "P" (for "Principal") to the importance of each operation of the "oOperations" collection.

5.2.6 Recuperating a query result

Example:

VB Script

```
Set oRoot = object.GetRoot
set myprojects = oRoot.GetSelection("Select ~qekPESs3iG30[Project] where Not
~KsUiCCB5i0)1[Diagram] ")
 For each oproject in myprojects
 oRoot.print oproject.GetProp("~Z20000000D60[Short Name]")
 Next
```

Java

```
MegaCollection mgcolAllProjects = mgRoot.getSelection("Select
~qekPESs3iG30[Project] where Not ~KsUiCCB5i0)1[Diagram]");
              String strText = "";
              for (MegaObject mgobjProject : mgcolAllProjects) {
                strText = strText + mgobjProject.getProp("~Z20000000D60[Short
Name]") + "\n";
```

```
}
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", strText);
```

Code description:

```
Set oRoot = object.GetRoot
set myprojects = oRoot.GetSelection("Select ~qekPESs3iG30[Project] where Not
~KsUiCCB5i0)1[Diagram] ")
```

➔ assigns the query result to the "myProjects" variable (this query retrieves all projects not described by a diagram).

```
For each oproject in myprojects
 oRoot.print oproject.GetProp("~210000000900[Name]")
 Next
```

➔ displays the names of all projects in "myProjects" collection.

5.2.7 Using a MegaObject or MegaCollection

The MEGA APIs provide the user with two basic types:

- the MegaObject type, which enables access to a MEGA object.
- the MegaCollection type, which enables access to a collection of objects.

An instance is an object, such as a business process, class, database, etc.

A MegaObject enables access to characteristics of an object instance.

Use the **GetProp** function, using the name as parameter.

VB Script

```
MyName = MyObject.Name
MyName = MyObject.GetProp("~210000000900[Name]") ' equivalent to the previous
line
```

Java

```
String strMyName = mgobjMyObject.getProp("~210000000900[Name]");
```

From a MegaObject, you can access each MegaCollection connected to an object instance, using the **GetCollection** function. If "myProject" is an object corresponding to a project instance, to access the collection of operations linked to this project, the code is as follows:

VB Script

```
Set myOperations = myProject.GetCollection("~OsUiS9B5iiQ0[Operation]")
For each anOperation in myOperations
do .... Next
```

Java

```
MegaCollection mgcolOperations =
mgobjMyProject.getCollection("~OsUiS9B5iiQ0[Operation]");
for (MegaObject mgobjOperation : mgcolOperations) {
 ....
}
```

The VBScript keyword **For Each** enables browsing of the content of a MegaCollection. However, other forms of search are allowed using the **Item** implicit method for the collection.

If a numeric parameter is passed to **Item**, this is treated as an ordinal position. As the standard VB **Count** function returns the number of items in the collection, you can write:

VB Script

```
numOperations = myOperations.Count
For i = 1 to numOperations
Set anOperation = myOperations.Item(i)
Set anOperation = myOperations(i) ' same thing
Next
```

Java

```
int iNumOperations = mgcolOperations.size();
              for(int j = 1; j<= iNumOperations; j++)
              {
                MegaObject mgobjOperation = mgcolOperations.get(j);
              }
```

5.2.8 Using Set in a VB Script code

Assignment of MEGA objects by command **Set** results in return of an object.

```
numOperations = myOperations.Count
Set anOper = myOperations.Item(1) 'Returns an object
Print myOperation.Item(1) 'Returns a value
```

Use of the **Set** command leads to reservation of memory space on your workstation. The use of highly recursive functions or tables of high volume can lead to saturation of this memory.

To avoid this type of problem, take care to free memory spaces reserved by unused objects.

This freeing can be carried out by assigning the value **Nothing** to your objects (for more details, see Microsoft documentation on VB Script language).

Example:

```
Set myproject= ...
...
Set myProject = Nothing
```

Use of the "Explicit Option" option is recommended, avoiding use of global variables by explicitly declaring variables in functions (global variables do not operate in MEGA web front-end context).

5.3 MEGA API Methods and Functions

MEGA proposes methods and functions relating to MegaObjects and MegaCollections.

5.3.1 Functions on MegaCollections

Accessing a MEGA repository

Any MEGA object (instance or collection) can access the repository to which it belongs by means of the **GetRoot** function.

VB Script
```
Dim Root
Set Root = MyObject.GetRoot
```

Java
```
MegaRoot mgRoot;
 mgRoot = mgobjMyObject.getRoot();
```

Collections of isolated objects

Collections obtained from the repository via the name of a MetaClass enable access to all instances of this MetaClass in the repository.

VB Script
```
Dim MyProjects
Set oRoot = object.GetRoot
Set MyProjects = oRoot.getCollection("~qekPESs3iG30[Project]")
' contains all projects in the repository.
```

Java
```
MegaCollection mgcolMyProjects = (MegaCollection)
mgRoot1.invokePropertyGet("~qekPESs3iG30[Project]");
```

You can also select objects using the **GetSelection** function.

VB Script
```
Dim MyProjects
Set MyProjects = Root.GetSelection _
("Select ~qekPESs3iG30[Project] where Name like 'A#'")
```

Java
```
MegaCollection mgcolMyProjects = mgRoot.getSelection("Select
~qekPESs3iG30[Project] where Name like 'A#'");
```

GetSelection can be called from any instance.

The **GetTypeID** function returns the identifier of the MetaClass of instances in the collection.

Collections of slave objects (connected)

Collections obtained from an instance other than the root are association collections. This is the way to obtain a collection of objects that are connected to an instance by a given association.

The original instance can be retrieved with the **GetSource** function. The **GetTypeID** function returns the identifier for the MetaAssociationEnd used.

VB Script

```
Set theSource = myOperationCollection.GetSource
Set theDuplicate = theSource.GetCollection _
(myOperationCollection.GetTypeID)
```

Java

```
MegaObject mgobjSource = mgcolOperations.getSource();
              MegaCollection mgcolOperationsDuplicate =
mgobjSource.getCollection(mgcolOperations.getTypeID());
```

The objects contained in such collections are slave objects. They can be used to access either the characteristics of the linked object or those of the MetaAssociation traversed.

VB Script

```
For each anOperation In myOperationCollection
' Name is an attribute of the operation, order is a link attribute
oRoot.Print anOperation.GetProp("~210000000900[Name]"), anOperation.
GetProp("~410000000H00[Order]")
Next
```

Java

```
String strText = "";
for (MegaObject mgobjOperation : mgcolOperations) {
  // Name is an attribute of the operation, order is a link attribute
  strText = strText + mgobjOperation.getProp("~210000000900[Name]") + " " +
mgobjOperation.getProp("~410000000H00[Order]") + "\n";
}
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", strText);
```

The **GetTypeID** function when used on a slave object returns the identifier of the MetaAssociationEnd used. It returns the same value as **GetTypeID** for the collection that provided access to the instance.

A slave object establishes a relation between three sub-objects:

- The source object, which can be obtained using the **GetSource** function (which is the equivalent function for the collection).
- The target object, which can be obtained using the **GetTarget** function.
- The association object, which can be obtained using the **GetRelationship** function.

The slave object is therefore a shortcut to the target and association objects. The above code can also be written as:

VB Script

```
For Each Ope In myOperationCollection
oRoot.Print Ope.GetTarget.GetProp("~210000000900[Name]"), Ope.GetRelationship.
GetProp("~410000000H00[Order]")
Next
```

Java

```
String strText = "";
```

```
for (MegaObject mgobjOperation : mgcolOperations) {
// Name is an attribute of the operation, order is a link attribute
strText = strText + mgobjOperation.getTarget().getProp("~210000000900[Name]") +
" " + mgobjOperation.getRelationship().getProp("~410000000H00[Order]") + "\n";
}
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", strText);
```

The target and source objects are not slave objects. The **GetTypeID** function returns the identifier for the object type and therefore is similar to **GetClassID**. The example below lists the "brothers" of a slave object (including itself).

VB Script

```
For Each Brother In _ MySlave.GetSource.GetCollection(MySlave.GetTypeID)
oRoot.Print Brother.GetProp("~210000000900[Name]")
Next
```

Java

```
String strText = "";
                MegaObject mgobjMySlave = mgcolOperations.get(1);
                for (MegaObject mgobjBrother :
mgobjMySlave.getSource().getCollection(mgobjMySlave.getTypeID())) {
                  strText = strText + mgobjBrother.getProp("~210000000900[Name]");
                }
                mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", strText);
```

Sorted collections

It is possible to obtain a collection that is sorted by one or more characteristics of the instances in the collection.

VB Script

```
Set MyOperations = MyProj.Operation ("~410000000H00[Order]", "~
Z20000000D60[Short Name]")
Set MyOperations = MyProj.Operation (1, "~ Z20000000D60[Short Name]")
Set MyOperations = MyProj.Operation _
(1, "~510000000L00[Creation Date]", -1, "~410000000H00[Order]")
```

Java

```
MegaObject mgcolMyProj = mgRoot.getCollection("~qekPESs3iG30[Project]").get(1);
   MegaCollection mgcolMyOps = mgcolMyProj.getCollection("Operation",
"~410000000H00[Order]", "~ Z20000000D60[Short Name]");
   mgcolMyOps = mgcolMyProj.getCollection("Operation", 1, "~Z20000000D60[Short
Name]");
   mgcolMyOps = mgcolMyProj.getCollection("Operation", 1,
"~510000000L00[Creation Date]", -1, "~410000000H00[Order]");
```

A positive value preceding the criterion indicates a sort in ascending order, a negative value means descending order.

Sort is in ascending order by default.

Searching a collection

The implicit **Item** function is used to find a particular instance in a collection. In addition to the name (local if applicable), the identifier, or the sequence number, an instance can be found by one of its characteristics:

VB Script

```
Set MyInstance = MyOperations("Type Operation", "A")
Set MyOcc = MyOperations(TypeOpeID, "A") ' identical to above,
' if TypeOpeID contains the identifier of the characteristic
```

If several instances match, the first one is returned.

It is also possible to find a specific position by using an object directly.

VB Script

```
Set MyOcc = MyOperations(MyOtherOcc) ' equivalent to
Set MyOcc = MyOperations(MyOtherOcc.GetID)
```

5.3.2 Functions and methods on MegaObjects

A function executes processing and returns a result. Unlike a method, parameters specified for a function must be placed between brackets.

Modifying an instance

To modify an instance, the GetProp method must be applied:

VB Script

```
MyOperations.Comment = _
"Here is the comment for my operation..."
MyOperations.GetProp("~f10000000b20[Comment]") =  _
"Here is the comment for my operation..."
```

Java

```
mgobjMySlave.setProp("~f10000000b20[Comment]", "Here is the comment for my
operation...");
```

Disconnecting a slave object

To disconnect a slave object:

➔ Apply the **Remove** method to a collection of slave objects specifying the name, identifier, or object to be connected.

The object is no longer usable after this operation.

VB Script

```
MyCollection.Remove MyObject
```

Java

```
remove();
```

Deleting an Object

To delete an object:

➔ Apply the **Delete** method to the object.

The object is no longer usable after this operation.

Be careful while using Delete on a collection, as items are shifted and the use of "for each" might not work properly.

VB Script
```
MyObject.Delete ""
```

Java
```
delete("");
```

Connecting an object

To connect an object:

➔ Apply the **Add** method to a collection of slave objects, specifying the name, identifier, or object to be connected.

VB Script
```
MyCollection.Add MyObject
MyCollection.Add MyObject.GetID
MyCollection.Add "Test"
                                    "Test" is the name of an existing instance.
```

Java
```
mgcolMyCollection.add(mgcolMyObject);
mgcolMyCollection.add(mgcolMyObject.getID());
mgcolMyCollection.add("Test");
```

Creating an object

To create an object, apply the Create method to a collection of objects obtained from the root (a simple create) or from a collection of slave objects (create/ connect).

- Without a parameter, the function creates an instance that has as its name the class name followed by a counter.
- A parameter of type String specifies the name (local if applicable) of the instance.

For any other attribute, you must specify the attribute name (or identifier), followed by the value to be assigned.

VB Script
```
MyCollection.Create
MyCollection.Create "Name1"
MyCollection.Create "Name2", "Comment", _
"Comment of Name2"
```

Java
```
mgcolMyCollection.create();
mgcolMyCollection.create("Name1");
mgcolMyCollection.create("Name2", "Comment", "Comment of Name2");
```

Accessing a new object

A new object created using an **Add** or **Create** method can be accessed as follows:

VB Script
```
set MyNewObj1 = MyColl.Create("Name1")
set MyNewObj2 = MyColl.Add ("Test")
```

Java
```
MegaObject mgcolMyNewObj1 = mgcolMyCollection.create("Name1");
MegaObject mgcolMyNewObj2 = mgcolMyCollection.add("Test");
```

Accessing the MetaClass of an object or collection

The **GetType** function allows an object or collection to be considered as an instance of a given MetaClass.

Used from a ***MegaObject***, the **GetType** function enables considering an object as a function of the MetaClass given as parameter.

VB Script
```
MyOrgProc.GetType("BPMN Process").explore runs the explorer from an object of the Organizational Process concrete MetaClass, considering it as an object of the "BPMN Owner Element" MetaClass.
```

Java
```
mgobjMyOrgProc.getType("BPMN Process").invokeMethod("Explore");
```

*Used without a parameter, operator **GetType** enables consideration of the current object as an element of the concrete MetaClass to which it belongs.*

Used from a ***MegaCollection*** of objects of different concrete MetaClasses, operator **GetType** allows you to obtain a collection restricted to instances of the MetaClass specified as parameter.

VB Script
```
oCollection.GetType("Sequence flow").explore runs the explorer on objects of the collection belonging to the "Sequence flow" MetaClass
```

Java
```
mgcolCollection.getType ("~jsV6VsHL7vJ0[Sequence
Flow]").invokeMethod("Explore");
```

If no object of the collection inherits the MetaClass given as parameter, operator **GetType** returns nothing. No error is generated.

Accessing object attributes

The **GetAttribute** function enables access to an object of MegaAttribute type to obtain type characteristics, translations if these exist, as well as the attribute value in its different formats.

VB Script
```
Print myobj.GetAttribute("Operation Type")
' Displays the attribute internal value
Print myobj.GetAttribute("Operation Type").Value("Display")
'Displays the attribute value in the language used
```

Java
```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", mgcolMyObj.getProp("Operation
Type"));
// Displays the attribute internal value
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgcolMyObj.getAttribute("Operation Type").getValue("Display"));
// Displays the attribute value in the language used
```

Attribute Format

You can obtain an attribute value in a given format. The default format is ASCII.

Available formats are: "Internal", "Ascii", "External", "Display". This is used as follows:

VB Script
```
Print MyOcc.GetProp("Operation Type", "External")
```

Java
```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", mgobjOcc.getProp("Operation
Type", "External"));
```

A default format can be specified on the instance using the **SetDefault function**.

VB Script
```
MyInstance.SetDefault("Display")
```

Java
```
mgobjOcc.setDefault("Display");
```

When defined for the root, the default format is applied to all instances.

VB Script
```
Root.SetDefault("Display")
```

Java
```
mgRoot.setDefault("Display");
```

5.4 Summary of Functions

- On MegaItems, ie. all objects:

VB Script	Java
CallFunction	public Object callFunction(Object name, Object... arg);
CallMethod	public void callMethod(Object name, Object... arg);
GetMethodID	public Object getMethodID(Object name);
GetNature	public String getNature(String format);
GetRoot	public MegaRoot getRoot();
GetSource	public MegaObject getSource();
GetTypeID	public Object getTypeID();
SetDefault	public void setDefault(String defaultValue);

- On MegaCollections:

VB Script	Java
AdviseRegister	public MegaAdviseToken adviseRegister(MegaAdviseTarget target);
Add	public MegaObject add(Object objectID);
Count	public int size();
Create	public MegaObject create();
Insert	public void insert(Object toInsert);
Item	public MegaObject get(Object index);
Remove	public void remove(Object toRemove);

- On MegaObjects:

VB Script	Java
Delete	public void delete(String options);
GetClassID	public Object getClassID();
GetCollection	public MegaCollection getCollection(Object vcolID, Object... sortCriteria);
GetCollectionID	public Object getCollectionID(Object name);
GetID	public Object getID();
GetProp	public String getProp(Object propID);
GetPropID	public Object getPropID(Object name);

VB Script	Java
GetRelationship	public MegaObject getRelationship();
GetType	public MegaObject getType(Object name);
GetTarget	public MegaObject getTarget();
Item	public Object item();
Remove	public void remove();
SetProp	public void setProp(Object propID, String value);

- On MegaAttributes:

VB Script	Java
Value	Value public String getValue();
DefaultFormat	
TextType	public Object getTextType();
DescriptionObject	public MegaObject getDescriptionObject();
GetAsPrivateProfile	public MegaPrivateProfile getAsPrivateProfile();
Translate	public MegaAttribute translate(Object language);
SourceObject	public MegaObject getSourceObject();
TestValue	public String testValue(Object value, String format);
Status	public int getStatus();
GetAttributeValue	public Object getValue(String format);

- On MegaRoot object:

VB Script	Java
BaseCanClose	public boolean isClosed();
BaseClose	public void close();
ContextObject	public MegaCOMObject contextObject(String Name);
CurrentEnvironment	public MegaCurrentEnvironment currentEnvironment();
EnterPrivilege	public Object enterPrivilege(String description);
LeavePrivilege	public void leavePrivilege();
TryPrivilege	public Object tryPrivilege(String[] description);
GetSelection	public MegaCollection getSelection(String request, Object... sortCriteria);
GetExcecutionStatus	public int getExecutionStatus();

VB Script	Java
GetObjectFromID	public MegaObject getObjectFromID(Object ident);
GetSystemRoot	
GetClassDescription	public MegaObject getClassDescription(Object classID);
GetCollectionDescription	
MegaCommit	public void megaCommit();
MegaRollback	public void megaRollback();
print Print	public void print(Object value);
SetOpenToken	public void setOpenToken(String token);
SetUpdater	

5.4.1 MEGA Operators

There are three operator types:

- "Compound": enable specification of behavior on MetaAssociations.

  Examples of "Compound" operators:

  - Owner
  - Ownership
  - Duplicate
  - Isolate
  - NameSpace
  - Navigate
  - Propose
  - Reference

- "Method": implements methods and functions available for APIs. These methods and functions can relate to a MegaObject, a MegaCollection, a MetaClass or to all MetaClasses.

  Examples of "Method" operators:

  - Edit
  - Explore

    invokeMethod("Explore")
  - IsAvailable
  - Open
  - SaveAs

- "Atomic": used for basic commands of MEGA.

  Examples of "Atomic" type operators:

  - Abbreviate
  - Modify
  - Check
  - Create
  - Disconnect
  - Description
  - Implicit
  - Import
  - Modify
  - Reorder
  - Read
  - Connect
  - Rename
  - Delete

- Translate
- Validate

To access all operators, search from the "_Operator" MetaClass.

How to write an operator

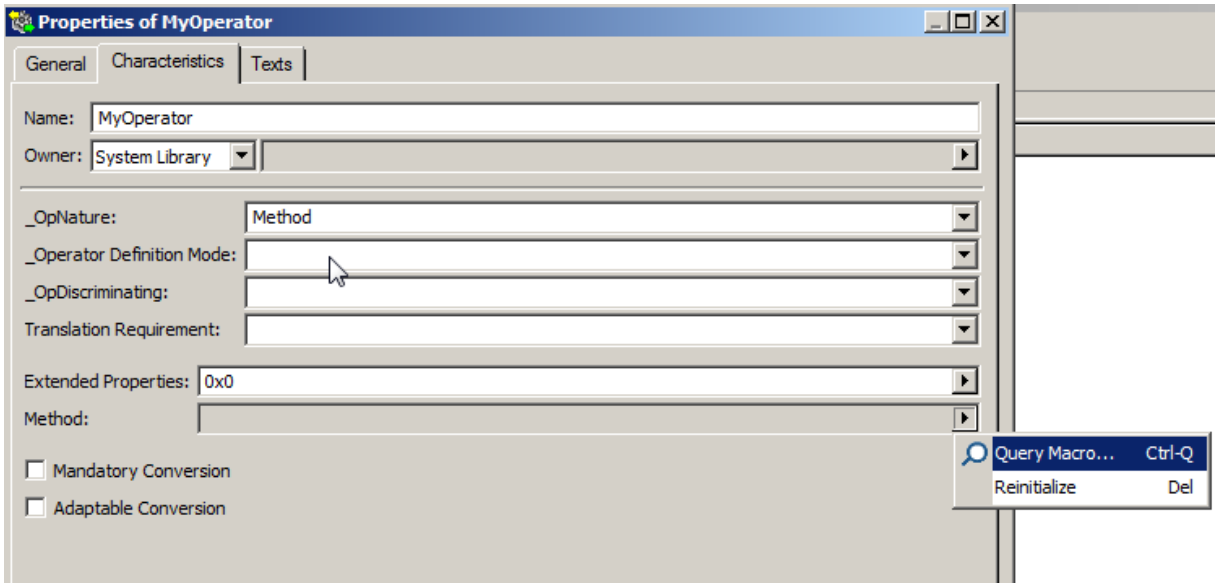
An operator is a standard function that may be called either from the root or from a specific MetaClass.

To create the operator:

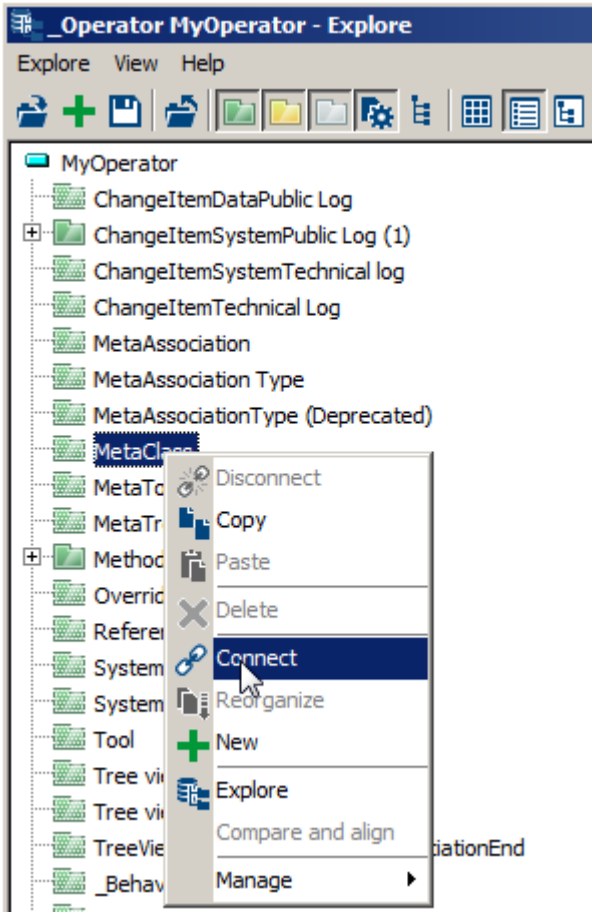
1. Open the VB Script editor and enter the following code:

```
[_Operator].Create("MyOperator").explore
[Macro].Create("MyOperator.Macro").explore
```

2. Access to the properties of the operator and connect the macro you have just created to your operator:



If your operator is specific to a MetaClass, connect the MetaClass to the operator through the MetaClass MetaAssociationEnd.



3. Navigate to the macro implementing the operator.

   For example: `MyOperator.Macro`.

4. If the operator needs to be written:
   - in Java, specify on the macro properties, the java class factory in the "_ObjectFactory" and the "dispatch" mode in the "SystemComponent" attribute.
   - in VB Script, Edit the macro.

- If the operator applies on a **MEGA Object**, the following methods must be implemented

VB Script The macro must implement the following functions:

```
'specify the number of arguments that will be passed to the operator: here 3.
Function InvokeArgCount() As Integer
  InvokeArgCount = 3
End Function
'Method called when the operator is called with 3 arguments. The first argument is the object on which the call is made (megaobject).
Sub InvokeOnObject(Obj as MegaObject, sFileName As String, strReportFilename as String , controlComponent As MegaObject)
  End Sub
```

Java The factory must defined a class containing the following functions:

```
public class MyOperator {
  /**
   * Specifies how many parameters are expected
   * if not defined, the method called has only one argument : the megaobject on which the call has been made
   */
  public int InvokeArgCount() {
    return 3;
  }
  /**
   * function called with the 3 arguments. The first argument is the object on which the call is made
   */

  public void InvokeOnObject(final MegaObject mgObjectToExport, final String strFilename, String strReportFilename, final MegaCOMObject controlComponent)
  {
    //procedure called when the operator is called
  }

}
```

Note that the InvokeOnObject method may return a value both in Java or VB Script. In that case, the java function must be defined with a return value and the VB Script method must be defined as a Function.

VB Script

```
'if you want your method to return a value, use
Function InvokeOnObject(Obj As MegaObject)
End Function
```

Java

```
//return a number
public int InvokeOnObject(final MegaObject mgObject)
{
}
```

- If the operator applies on **MegaRoot**, the following methods must be implemented. If you need to add additional arguments, you need to implement **InvokeArgCount** to specify the number of arguments.

VB Script

```
Sub InvokeOnRoot(oroot As MegaRoot)
End Function
'if you want your method to return a value, use
Function InvokeOnRoot(oroot As MegaRoot)
```

```
End Function
```

Java

```
public void InvokeOnRoot(final MegaRoot mgRoot)
{
}
Or if you want your method to return a value, use
//return an integer
public int InvokeOnRoot(final MegaRoot mgRoot)
{
}
```

- If the operator applies on a collection. In that case, the method to be implemented is **InvokeOnCollection**. If you need to add additional arguments, you need to implement **InvokeArgCount** to specify the number of arguments.

VB Script

```
Sub InvokeOnCollection(Coll As MegaCollection)
End Sub
'if you want your method to return a value, use
Function InvokeOnCollection(Coll As MegaCollection)
End Function
```

Java

```
public void InvokeOnCollection(final MegaCollection lst )
{
}
Or if you want your method to return a value, use
//return an integer
public int InvokeOnCollection(final MegaCollection lst )
{
}
```

5.4.2 Accessing MEGA API Public Objects

Public objects of MEGA are available for external VB Script code, Java code or VBA.

Public objects

Use of MEGA APIs form an external program is based on the three following public components:

MegaApplication

This component enables administration of MEGA from an external program. It is not accessible from an open session. In particular, it cannot be used to its full extent from the script editor. This is due to the fact that this component totally controls the MEGA session (connection, repository opening) and is not designed to collaborate with the MEGA workdesk. When this component is activated, it is not possible to open the MEGA workdesk or the MEGA administration tool.

In VB script, it can be created by the Basic function **CreateObject ("Mega.Application")**.

MegaCurrentEnv

This component enables access to the current MEGA session. Unlike the **MegaApplication** component, it is used to collaborate with the MEGA workspace. It enables determination of whether the MEGA session is open (using the **IsSessionOpen** function) and therefore enables an external program to use MEGA without explicitly connecting.

However, when the **GetRoot** function of this component is called when the MEGA workdesk is not open, it takes charge of the MEGA session by proposing a specific connection dialog box. In this case, it is no longer possible to open the MEGA workdesk, since the connection this proposes could be in conflict with that opened from this component.

This component can be used from the script editor and gives access to the current session.

This is to be avoided

In VB script, it can be created by the Basic function: **CreateObject("MegaRepository.CurrentEnv")**.

To obtain the **MegaApplication** public object from a **MegaCurrentEnv** object use the **MegaCurrentEnv.Site** function.

MegaToolkit

This component groups utility functions developed for use indistinctly in internal or external mode. In this way, it does not make reference to characteristics of the session in progress.

In VB script, it can be created by the **CreateObject ("Mega.Toolkit")** Basic function.

5.4.3 Accessing public objects from another MEGA object

This is recommended

To obtain the **MegaCurrentEnv** public object from a **MegaRoot** object use the **MegaRoot.CurrentEnvironment** function.

This is recommended

The **MegaToolkit** public object can be obtained from the following objects:

- **MegaCurrentEnv**, using the **MegaCurrentEnv.Toolkit** function,
- **MegaApplication** using the **MegaApplication.Toolkit** function.

6 MACROS USED IN MEGA

Here are examples of where macros (Java or VB Script) are used in MEGA.

For information on macro about...		see...
Command management	MetaCommand Manager	**HOPEX Studio – Versatile desktop** Technical Article (Configuring the desktop > Examples of macros)
	MetaCommand Item	
	MetaCommand Group	
Wizard implementation kinematic		**HOPEX Forms - MEGA Wizard Implementation – Tutorial** Technical Article **HOPEX Forms** Technical Article
Property Pages	Macro call	**HOPEX Forms - MEGA Property Pages – Tutorial** Technical Article **HOPEX Forms** Technical Article
	viewport management	**HOPEX Forms** Technical Article
Metamodel	updateTool implementation - on attributes - on links	**All about starting with APIs** Technical Article (Implementing an Update Tool in script p. 146) **HOPEX Forms** Technical Article
	calculated attributes	**HOPEX Studio - MEGA Studio** Technical Article Managing the MetaModel / MetaAttributes: - "Using VB Scripts to Calculate Characteristics"
	abstract property implementation	**HOPEX Forms** Technical Article
	collection implementation	**HOPEX Forms** Technical Article
Operators	Operator implementation	**All about starting with APIs** Technical Article (MEGA Operators p. 71)
	Writing of operator behavior on links	**Customizing perimeter** Technical Article
Requests	Writing a dynamic query	**All about starting with APIs** Technical Article (Writing a dynamic query p . 188)

For information on macro about...		see...
In diagrams	Post-processing	**JavaDoc** documentation
	Interactive Plug-in (diagram plug-in, drag and drop plug-in)	**All about starting with APIs** Technical Article (Setting up interactive plug-ins in a diagram p. 185)
Read/write dynamic access management	on Data Access Rule	Available shortly
HTML formatter	Widgets	**Creating Widgets** Technical Article
Macro call from RTF and HTML descriptors and code		**All about starting with APIs** Technical Article (Calling a macro from HTML, code and RTF descriptors p. 163)
Report content		**Reports - Writing java report chapters** Technical Article **Reports - Writing java Report renderers** Technical Article
Definition rule	Modifying password	**Administration - Supervisor** User Guide ("Modifying password definition rules")
Desktop	- Click Manager - Event management load, close, save and deactivation - Tool	**HOPEX Studio – Versatile desktop** Technical Article
Object and link display management	In diagrams via DiagramTypeXXX Condition	
Post-processing on business documents		**HOPEX Studio - Publisher** Technical Article ('Modifying document behavior")
Assessment	Plug-in	**Assessment** Technical Article
Perimeter	Behavior	**Customizing perimeter** Technical Article **HOPEX Studio - MEGA Studio** Technical Article
Workflow		**HOPEX Collaboration Manager - Workflows** user guide

For information on macro about...		see...
Specific processing to certain concept	User group management	**Administration - Supervisor** User Guide (Configuring a dynamic person group with a macro)
	Steering Calendar	**HOPEX Studio - Steering Calendar** Technical Article **SOHO Customization - Soho Assessment** Technical Report (Steering Calendar)
	Scheduler	**HOPEX Studio - Scheduler** Technical Article

7 ADMINISTRATION OF MEGA FROM APIS

The objective of administration applications is to manage transactions, environments, users, etc. MEGA administration can be carried out from the application itself or from the outside.

7.1 Introduction

7.1.1 Starting administration

To start an administration application in VBScript, you must use a **MegaApplication** type variable. You can access such a variable by using the standard ***CreateObject*** function:

```
Dim oMegaApplication
Set oMegaApplication = CreateObject ("Mega.Application")
```

When the variable has been created, you can access all the properties accessible from the **MegaApplication** class and its associated objects (environment, repository, etc.).

For more detailed information, see "API - Reference Guide".

7.1.2 Connecting to an open session

To connect to an open session to analyze or modify the current repository:

➔ use a **MegaCurrentEnv** type variable, or

➔ use the standard function **CreateObject ("MegaRepository.CurrentEnv")**.

From such a variable you can access the repository root (**MegaRoot**) and therefore all the contained objects.

Example: access to the current repository. This example displays the number of repository procedures:

```
Dim oMegaCurrentEnv
Set oMegaCurrentEnv = CreateObject (“MegaRepository.CurrentEnv”)
Dim oMegaRoot as MegaRoot
Set oMegaRoot = oMegaCurrentEnv.GetRoot
MsgBox oMegaRoot.Getcollection ("Procedure").Count
```

Connection to an already open MEGA session or manual opening

The **MegaCurrentEnv** object enables you to determine if a MEGA session is open or not by means of the **IsSessionOpen** function.

If the MEGA session is not open, the **GetRoot** function now proposes an interactive connection dialog box enabling the user to open a session.

Intrusion protection

To protect the repository against malicious intrusion, **MEGA** displays an acceptance message when calling the ***GetRoot*** function. This behavior can be cancelled in the case of a call originated by MEGA. To do this, see the reference guide, **SetOpenToken** function.

7.2 Repository Administration Tasks

You can carry out certain MEGA administration tasks using APIs.

You can execute these tasks by means of VB scripts written and stored in external .vbs files.

MEGA must be closed during execution of VB scripts carrying out administration tasks. The data regarding environments, transactions, users and repositories cannot be read when a MEGA session is open.

For a complete list of administration functions, see the "API Reference Guide" in "API annex" Technical Article.

7.2.1 Connecting to an open repository

To connect to an open repository, enter the following code in a .vbs file:

```
Set oMegaApp = CreateObject ("Mega.Application")
Set oEnvironment = oMegaApp.Environments.Item("EnvironmentFolder")
oEnvironment.CurrentAdministrator = "Administrator"
oEnvironment.CurrentPassword = "AdministratorPassword"
Set oDataBase = oEnvironment.Databases.Item("RepositoryName")
```

Replace the words in bold by the environment path, administrator name, administrator password, repository name and repository logical backup file name respectively.

Code description:

```
Set oMegaApp = CreateObject ("Mega.Application")
```

Creates an instance of the class MegaApplication and assigns it to the "oMegaApp" Variable. This class defines the data corresponding to a MEGA site.

```
Set oEnvironment = oMegaApp.Environments.Item("EnvironmentFolder")
```

From the environments defined for the application oMegaApp, the environment located in the "EnvironmentFolder" folder is retrieved and assigned to the "oEnvironment" Variable.

You must specify the path in the form in which the environment was referenced (if a UNC path was used, you must use the UNC path).

```
oEnvironment.CurrentAdministrator = "Administrator"
```

The administrator name is entered.

```
oEnvironment.CurrentPassword = "AdministratorPassword"
```

The administrator password is entered.

```
Set oDataBase = oEnvironment.Databases.Item("RepositoryName")
```

From the repositories of the selected environment, the "RepositoryName" repository is retrieved and assigned to the "oDataBase" variable.

7.2.2 Repository logical backup

Having written code for connection to the repository oDataBase, enter the following code in a .vbs file:

```
oDataBase.LogicalSave "LogicalBackupFileName")
WScript.Echo "Processing completed"
```

Code description:

```
oDataBase.LogicalSave "LogicalBackupFileName")
```

Applies the "LogicalSave" function (which carries out repository logical backup) to the selected repository, specifying the complete name of the backup file.

```
WScript.Echo "Processing completed"
```

When logical backup is completed, a window appears and displays "Processing completed".

7.2.3 Reinitializing a repository backup logfile

Having written code for connection to the repository oDataBase, enter the following code in a .vbs file:

```
Set oDataBaseLog = oDataBase.Log
oDataBaseLog.Reset
WScript.Echo "Processing completed"
```

Code description:

```
Set oDataBaseLog = oDataBase.Log
```

The selected repository logfile is retrieved and assigned to the variable "oDataBaseLog".

```
oDataBaseLog.Reset
```

The "Reset" function, which reinitializes this logfile, is applied to the repository logfile.

```
WScript.Echo "Processing completed"
```

When backup is completed, a window appears and displays "Processing completed".

7.2.4 Deleting a repository

Having written code for connection to the repository oDataBase, enter the following code in a .vbs file:

```
oDataBase.Destroy
WScript.Echo "Processing completed"
```

Code description:

```
oDataBase.Destroy
```

The "Destroy" function that deletes the repository is applied to the selected repository.

```
WScript.Echo "Processing completed"
```

When deletion of the repository is completed, a window appears and displays "Processing completed".

7.2.5 Deleting a transaction

Having written code for connection to the repository oDataBase, enter the following code in a .vbs file:

```
oTransaction.Abort
WScript.Echo "Processing completed"
```

Code description:

```
oTransaction.Abort
```

The "Abort" function that deletes the transaction is applied to the transaction.

```
WScript.Echo "Processing completed"
```

When deletion of the repository is completed, a window appears and displays "Processing completed".

7.3 Executing tasks offline

Certain time-consuming processing operations can be batch-executed using administration commands and MEGA APIs.

7.3.1 Reorganizing repositories

The script given as an example below enables reorganization of all repositories of the first environment of a site. It saves active transactions in logfiles and executes logical backup of repositories. This script is an administration script: MEGA must therefore be closed before running the script. Transactions are recreated and their logfiles reinjected.

```
Explicit Option


Dim TabTrans(256,2), sDbSave, sDbRej, STransSave, sTransRej, sAdministratorName,
sPassword
Dim oMegaApp, oDataBase, oEnvironment, oTransaction, i, j, oShell, sSystem
i=0


' TODO: replace the two values with an available MEGA user and its optional
password.


sAdministratorName = "a user name"


sPassword = "your password"
Set oMegaApp = CreateObject ("Mega.Application") Set oEnvironment =
oMegaApp.Environments.Item(1)
oEnvironment.CurrentAdministrator = sAdministratorName if sPassword <> "" then
oEnvironment.CurrentPassword = sPassword
```

```
' Stores each transaction in a logfile.
’There must be no active transaction
For each oTransaction in oEnvironment.Transactions
 If Right(oTransaction.Name,8)= "(System)" Then
 Else
  i=i+1
  TabTrans(i,1) = oTransaction.User name
  TabTrans(i,2) = oTransaction.Database.name
  STransSave = oEnvironment.path & "\Mega_usr\Trans" & oTransaction.Name & ".mgl"
  oTransaction.Database.Log.Export STransSave
  oTransaction.abort
 End If
Next

' Performs the logical save of each database except the system database.
For each oDatabase in oEnvironment.Databases
 If oDatabase.Name="SystemDb" Then
 Else
   sDbSave = oEnvironment.path & "\Mega_usr\DB" & oDataBase.Name & ".mgr"
   sDbRej  = oEnvironment.path & "\Mega_usr\DB" & oDataBase.Name & "Rej.mgr"
   oDataBase.LogicalSave
sDbSave,"meta=off,technical=off,data=On,FileOpen=Rewrite"
   oDataBase.Reset
   odataBase.Import sDbSave, sDbRej
 End If
Next

' Creates new transaction replacing the previous ones.
For j= 1 to i
  oEnvironment.Transactions.create TabTrans(j,2), TabTrans(j,1)
Next
' Imports the saved logs in the corresponding transactions. For each
oTransaction in oEnvironment.Transactions
 sTransSave = oEnvironment.path & "\Mega_usr\Trans" & oTransaction.Name & ".mgl"
 sTransRej  = oEnvironment.path & "\Mega_usr\Trans" & oTransaction.Name &
"Rej.mgl"
 If Right(oTransaction.Name,8)= "(System)" Then
 Else
  oTransaction.Database.import sTransSave,sTransRej
 End If
Next

Set oTransaction = Nothing
```

```
Set oDataBase = Nothing
Set oEnvironment = Nothing
Set oMegaApp = Nothing

MsgBox "Environment closed"
```

7.3.2 Generating documents

To create a document using a VB script:

➔ Apply to a MEGA object the **NewDocument** method.

This method takes as a parameter the document template from which you create the document.

The **NewDocument** method returns the created document.

Example:

```
Set oRoot = object.GetRoot
Set objDoc = oRoot.GetCollection("~pj)grmQ9pG90[Business Process]").Item("~2F6SN4tSpyh0[Purchasing]").NewDocument("Business Process documentation")
```

Creates a document on the "Purchasing" business process from the "Business Process Description" document template and assigns it to the variable "objDoc".

To update document links:

➔ Apply to the document the **RefreshDocument** method without a parameter.

Example:

```
Set RefreshStatus = objDoc.RefreshDocument()
```

This method returns an object with value:

- 1 when document update is in progress
- 0 if document update is completed.

Two document update operations cannot be started simultaneously.

To detach a document from MEGA:

➔ Apply to the document the **DetachDocument** method. This takes as parameter the complete file name (example: C:\My documents\Document.doc).

Example:

```
objDoc.DetachDocument("C:\My documents\Document.doc")
```

This method detaches the current document, cuts document links and creates a file independent of MEGA.

7.3.3 Generating Web sites

To generate a Web site:

➔ Apply to the Web site the **GenerateWebSite()** function. Example:

```
Set oRoot = object.GetRoot
Set oWebSiteList = oRoot.GetCollection("Web Site")
For each oWebSite in oWebSiteList
 oWebSite.GenerateWebSite()
Next
```

In this example, Web sites existing in **MEGA** are retrieved and we apply to each one the **GenerateWebSite()** function.

VB scripts also enable generation of the CHM file corresponding to a Web site. Apply to the Web site the **GenerateCHM()** function.

Example:

```
Set oRoot = object.GetRoot
Set oWebSiteList = oRoot.GetCollection("Web Site")
For each oWebSite in oWebSiteList
  oWebSite.GenerateCHM()
Next
```

In this example, Web sites existing in **MEGA** are retrieved and you apply to each one the **GenerateCHM()** function.

8 COMMUNICATION BETWEEN MEGA AND THE OUTSIDE

The MEGA application offers various possibilities of communication with the outside, for administration tasks for example.

8.1 API Scripts and .NET

Use .NET applications only with Administration APIs. For Dispatch call, see "Late binding" information on Microsoft Help and Support website:
http://support.microsoft.com/kb/302902/en-us

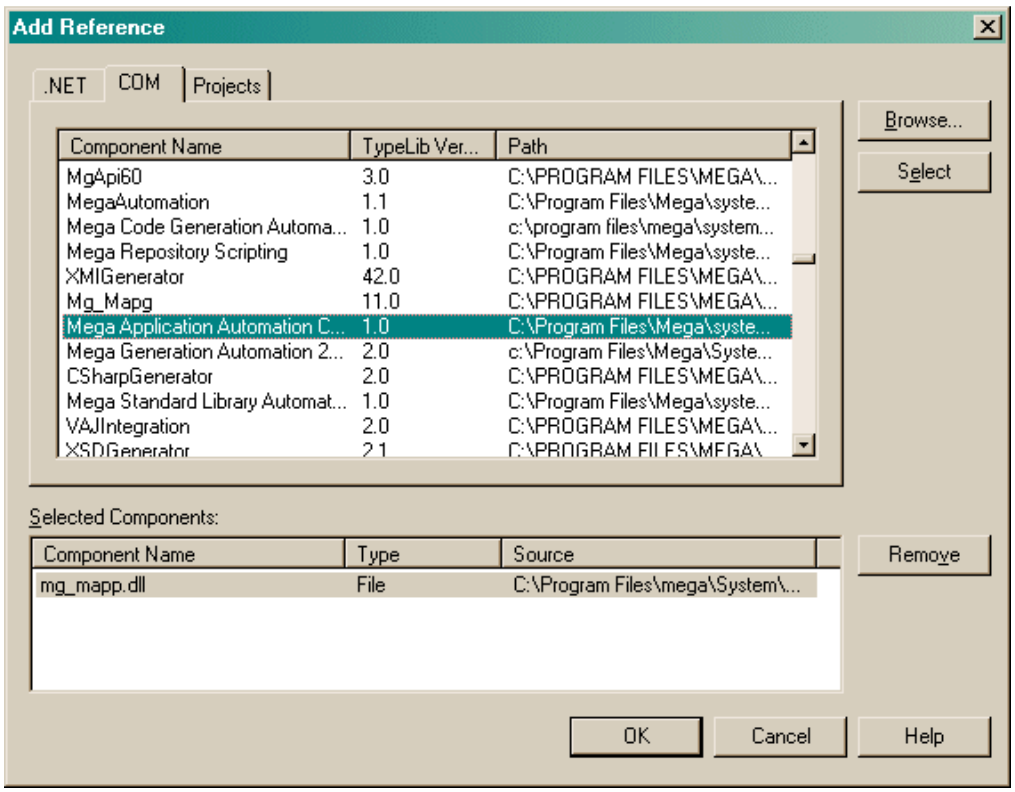
Access and update APIs of the MEGA repository are presented in the form of COM components. They can be integrated in a .NET application. The user can develop .NET compatible components accessing MEGA repository data. This section explains the principle used to implement a .NET application accessing data in a MEGA repository. It also raises points relating to constraints linked to choice of .NET language.

8.1.1 Implementation principle

Whatever the language chosen for your .NET application (Visual Basic.NET, C++. NET, C#, etc.), you can integrate a COM component and use objects and functions exported.

In Visual.NET:

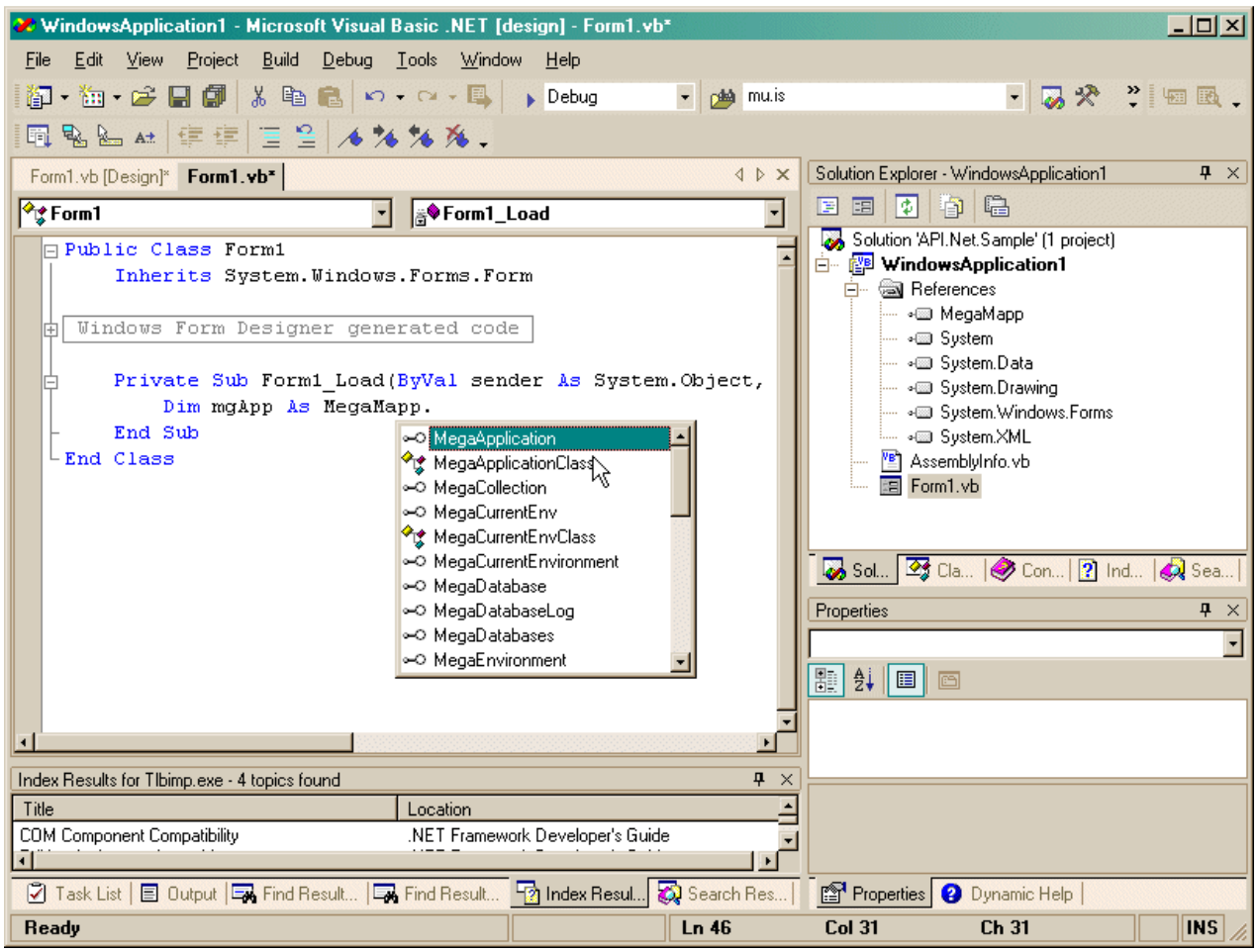
1. Select **Project > Add Reference**.
2. In the **Add Reference** dialog box, select the **COM** tab.
3. Select the **MEGA Application Automation component** component or query the DLL in **< folder MEGA> \system\mg_mapp.dll**.
4. Add this dll to the list of selected components.



When a reference has been created for the component, API objects are accessible. A namespace has been created automatically and is called MegaMapp.

To access API classes such as MegaApplication, MegaCollection, etc., prefix the class name with this namespace: **MegaMapp**.

Intellisense now handles display of available classes as shown in the following figure.



COM component integration mechanism

Life cycle management of COM and .NET objects is not handled in the same way. For COM objects, the application handles reference creation (by QueryInterface::AddRef()) and reference deletion (by QueryInterface::Release()). In the case of a .NET object, it is the framework that handles freeing of objects.

To ensure total integration of the MEGA API component, Visual.NET automatically creates an intermediate .NET DLL. The role ofthis intermediate dll (interop.MegaMapp.dll) is to encapsulate each COM object exported and to handle its life cycle. From the user point of view, the operation is transparent and it can use objects of the API like any other .NET object.

Strong naming

To avoid implementation or compatibility problems caused by updating of certain DLLs, developments can be carried out using strong names. This system allows you to assign to each application object an assembly comprising this name and a unique key.

For more details, refer to **Strong-Named Assemblies** of MSDN.

For this type of development, all components inserted in the project must be signed with an identification key. Therefore, the encapsulation DLL generated by Visual.NET (interop.MegaMapp.dll) is not assigned to a key. It can not therefore be inserted in this type of project.

To alleviate this problem, the encapsulation DLL should be generated manually using the tlbimp.exe utility provided with the .NET framework.

It is this utility that is used to create the encapsulation dll automatically, but all options are not used.

On a command line, enter the sequence:

```
TLBIMP COMdll.dll /out:Netdll.dll /keyfile:keyfile.snk /
namespace:MEGA /asmversion:x.0.0.0
```

Where parameters are as follows:

- `COMDll.dll`: name of the COM DLL to be converted. It can be complemented by the complete name of the folder (use quotes if the name includes spaces).
- `Netdll.dll`: name of the encapsulation DLL
- `keyfile.snk`: name of the file containing the unique key assigned to the application. This file can be generated using the sn.exe utility provided with the .NET framework.
- `namespace: MEGA`: "MEGA" is the name of the encapsulation DLL namespace. In this case, the API objects will be accessible by prefixing class names with "MEGA"
- `asmversion`: defines the version number of the generated DLL.

8.1.2 Language characteristics

In theory, all .NET languages are interchangeable. Classes can be created in VB.NET and overload definitions in C# or C++.NET.

In practice, there are differences between each language.

Optional parameters

Languages like C# do not allow the use of optional parameters. However, this is possible in VB.NET.

The MEGA API has many functions of which parameters are optional. These are either options (MegaDatabase.HistoryReset()), or sort or assignment parameters

(MegaRoot.getSelection(), MegaObject.getCollection(), MegaCollection.Create()). Finally, to enable use of APIs in languages like C#, optional parameters can be indicated as empty character chains. In this case, it can be become complicated to provide the six optional parameters of functions Create() or getCollection(). In addition, it is recommended that an intermediate class be defined exporting underlying functions using different overloads (1 to 6 parameters if necessary).

Unexplained functions

MEGA APIs accessed from VBA or VBScript enable use of a certain number of functions undeclared in the type library. This is the case for functions derived from operators like MegaObject.Extract().

This mechanism is possible in VB6 or VBScript using the tardive (late binding) link and implementation by each COM object of the Dispatch() interface (see MSDN on this subject).

In VB.NET this mechanism is also possible using the Option Strict Off directive. However, highly typed languages (such as C# or Java) do not allow access to undeclared functions from an untyped variable. We must pass to COM objects by a reference.

8.2 VBA Application Example (Visual Basic for Applications)

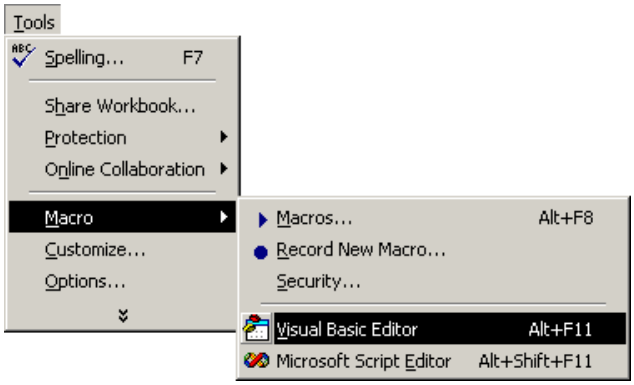
Example of integration between Microsoft Excel® and MEGA

This example shows how to access MEGA from Excel and retrieve data contained in a MEGA repository that can be used by Excel.

In this case, we shall create a macro from Excel. This macro will access MEGA, retrieve the names of procedures contained in the current repository and display them in an Excel spreadsheet.

To create a new macro in Excel:

1. From Excel menu bar, select **Tools > Macro > Visual Basic Editor**.

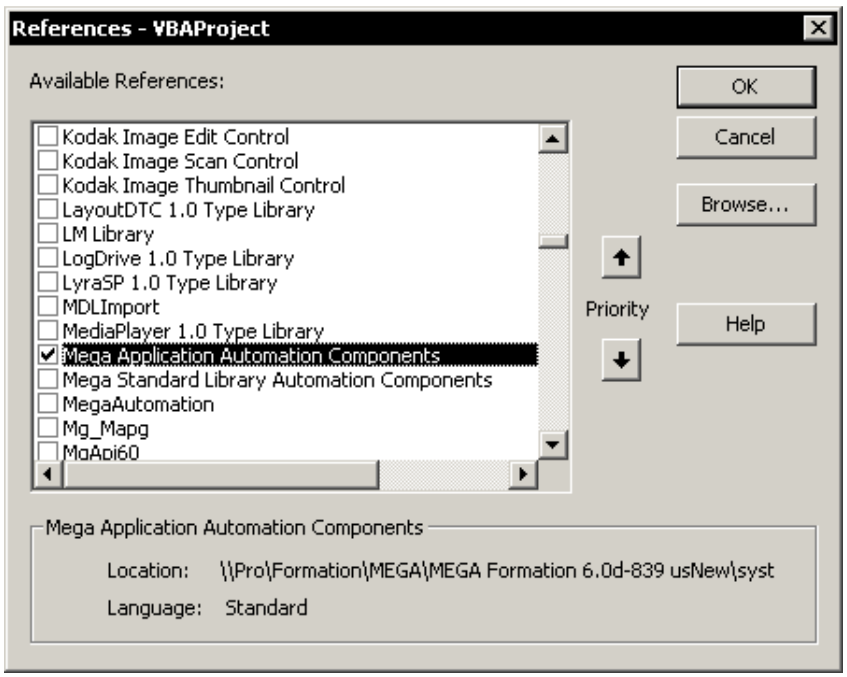


The Visual Basic editor opens.

To use **MEGA** APIs and benefit from Intellisense (display of names of classes, functions, parameters, etc.) create a reference for a **MEGA** component (DLL) in the VBA project.

To create a reference for this component:

1. In the Visual Basic editor menu bar, select **Tools > References**.
2. In the **VBAProject References** dialog box, select the **Mega Application Automation Components** component.



*If the component does not appear in the list, click **Browse** and add it from the "System" directory of the **MEGA** installation folder (the DLL file to be added is called **mg_mapp.dll**).*

3. Enter the macro code in the editor:

```
Sub ImportProcedure()
Dim oMegaCurrentEnvironment As New MegaCurrentEnv
Dim oMegaRoot As MegaRoot
Set oMegaRoot = oMegaCurrentEnvironment.GetRoot
Dim oProcedure As MegaObject
Dim nRow As Integer nRow = 1
For Each oProcedure In oMegaRoot.GetCollection("Procedure")
Range("B" & nRow).FormulaR1C1 = oProcedure.GetProp("Name")
nRow = nRow + 1
Next
End Sub
```

Code description:

```
Sub ImportProcedure()
```

Contains the name of the code used.

```
Dim oMegaCurrentEnvironment As New MegaCurrentEnv
```

The oMegaCurrentEnvironment variable is declared and assigned an instance of the class MegaCurrentEnv corresponding to the current MEGA environment.

```
Dim oMegaRoot As MegaRoot
```

The oMegaRoot variable of type MegaRoot is declared (corresponding to the root of a MEGA repository).

```
Set oMegaRoot = oMegaCurrentEnvironment.GetRoot
```

The root of the repository open on the current environment is retrieved (oMegaCurrentEnvironment.GetRoot) and is assigned to the variable oMegaRoot.

```
Dim oProcedure As MegaObject
```

The oProcedure variable of type MegaObject is declared (corresponding to a MEGA object).

The remainder of the code retrieves the collection of procedures and displays the name of each procedure in a different cell of the same Excel spreadsheet column.

```
Dim nRow As
```

variable nRow of integer type is declared.Integer nRow = 1

```
For Each oProcedure In oMegaRoot.GetCollection("Procedure")
Range("B" & nRow).FormulaR1C1 = oProcedure.GetProp("Name")
nRow = nRow + 1
Next
End Sub
```

For each procedure of the collection of repository procedures, the procedure name is inserted in a cell of column B, beginning with cell B1.

Counter nRow enables passage from one cell to another.

To execute this macro:

1. In the Visual Basic editor toolbar, click  .

   A message asks if you accept the process that will try to access the **MEGA** repository.

2. Click **Yes**.

   The names of the repository procedures are displayed in the Excel spreadsheet.

The same principle is used to access **MEGA** from other VBA compatible applications (Word or PowerPoint for example).

9 TOOLKIT

9.1 Metamodel

9.1.1 Accessing an attribute translation using APIs

To access an attribute translation using APIs, you need to use the MegaAttribute component.

To get the "Att" MegaAttribute component, enter:

VB Script Either:

```
set myMAtt = myObject.GetProp("Att", "Object")
```

or:

```
set myMAtt = myObject.GetAttribute("Att")
```

```
MegaAttribute mgattMA = (MegaAttribute) myObject.getProp("Att", "Object");
```

Java

```
MegaAttribute mgattMA = myObject.getAttribute("Att");
```

After, you can refer to the MegaAttribute interface.

Example:

VB Script

```
'MegaContext(Fields)
set myConcept = GetObjectfromID("~ezHXsMuE3fG0[Analisi]")
set myAtt = GetObjectfromID("~ezHXsMuE3fG0[Analisi]").GetAttribute("Name")
print "Current language: " & myAtt.DescriptionObject.GuiName & " : '" & myAtt & "' or:
" & myAtt.Value
print "In Spanish: " & myAtt.~n97OO26RrO00[Espanol].DescriptionObject.GuiName & " : '"
& myAtt.~n97OO26RrO00[Espanol] & "' or: " &
myAtt.Translate("~n97OO26RrO00[Espanol]").Value
print "In GUI Language: " & myAtt.GuiLanguage.DescriptionObject.GuiName & " : '" &
myAtt.GuiLanguage & "' or: " & myAtt.Translate("GuiLanguage").Value
' in 2007 SP1:
for each transl in myAtt.DescriptionObject.Translations
set myTranslation = myAtt.Translate(transl.LanguageID)
print "Traduction in " & myTranslation.DescriptionObject.GuiName & " : " &
myTranslation.Value
next
```

Java

```
MegaObject mgobjConcept = mgRoot.getObjectFromID("~ezHXsMuE3fG0[Report]");
                MegaAttribute mgattAtt = mgobjConcept.getAttribute("Name");
                mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", " Current language: " +
mgattAtt.getDescriptionObject().invokePropertyGet("GuiName") + " : " +
mgattAtt.getValue());
                mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", "In Spanish: " +
((MegaAttribute)
```

```
mgattAtt.invokePropertyGet("~n97OO26RrO00[Espanol]")).getDescriptionObject().invokePro
pertyGet("GuiName") + " : " +
mgattAtt.translate("~n97OO26RrO00[Espanol]").getValue());

              mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", "In GUI Language: " +
((MegaAttribute)
mgattAtt.invokePropertyGet("GuiLanguage")).getDescriptionObject().invokePropertyGet("G
uiName") + " : " + mgattAtt.translate("GuiLanguage").getValue());


              //in 2007 SP1:

              for (MegaObject mgobjTransl : (MegaCollection)
mgattAtt.getDescriptionObject().invokeFunction("Translations")) {

                MegaAttribute mgAttMyTranslation =
mgattAtt.translate(mgobjTransl.invokeFunction("LanguageID"));

                mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", "Translation in " +
mgAttMyTranslation.getDescriptionObject().invokePropertyGet("GuiName") + " : " +
mgAttMyTranslation.getValue());

              }
```

9.1.2 Accessing the metamodel description using APIs

Do not consult metamodel elements the same way you would do for any MEGA concept, i.e. by directly working with "MetaClass", "MetaAssociation", "MetaAttribute", "MetaAssociationEnd" MetaClasses and related MetaAssociations.

The code written that way is hardly ever efficient, and too dependent on the MEGA metamodel modeling style.

The modeling style is changing over time and takes on some of the implicit data – particularly the fact that conventional MetaAttributes are not necessarily associated with MetaClasses.

Moreover, note that the arrival of abstract MetaModel (with MEGA 2007 and even more with its evolutions in MEGA 2009) makes the use of MetaModel native data more and more complex.

To do this, a virtual abstraction layer has been developed, which uses in particular the compiled Metamodel, in this way enabling optimized and precise access to Metamodel concepts.

This layer does not exactly correspond to Metamodel concept implementations, and has been designed to simplify access to data – possibly calculated – which a developer may require to dynamically discover the Metamodel.

Technically, this abstraction layer can be used by APIs by means of MegaObjects and MegaCollections, therefore in exactly the same way as the native objects they represent.

So as to avoid confusion, we try to use a vocabulary for concepts handled in this abstraction layer different from that used in the native Metamodel. In addition, this vocabulary can reference that of APIs as required.

We shall first explain the model object of this abstraction layer. This model has two main entry points, which we shall name ClassDescription and CollectionDescription.

Objects of ClassDescription type enable object MetaClass discovery. Each MegaObject used in APIs makes available its class description by means of the GetClassObject function. A quick method of making contact with such a description and exploring it:

VB Script `myMegaObject.GetClassObject.Explore`

Java
```
mgobjMegaObject.getClassObject().invokeMethod("Explore");
```

Objects of CollectionDescription type enable description of the interface of a MegaCollection. Each MegaCollection used in APIs makes available this description by means of the GetTypeObject function; we can explore such a description with the following script line:

VB Script
```
myMegaCollection.GetTypeObject.Explore
```

Java
```
mgcolMegaCollection.getTypeObject().invokeMethod("Explore");
```

Or:

VB Script
```
Set oRoot = object.GetRoot
oRoot.GetCollection("~KekPBSs3iS10[Diagram]").GetTypeObject.Explore
```

Java
```
mgRoot.getCollection("~KekPBSs3iS10[Diagram]").getTypeObject().invokeMethod("Explore");
```

Note that *GetTypeObject* function is also available on a *MegaObject*. In that case, the returned *CollectionDescription* usually corresponds to the description of the collection from which we got the MegaObject (it is not necessary true for collections built from heterogeneous objects).

From version 2007 SP1, we can directly access these descriptions without passing via an already existing MegaObject or MegaCollection, by means of the following functions available on the MegaRoot object:

VB Script
```
myObject.getRoot.GetClassDescription(ClassID)
myObject.getRoot.GetCollectionDescription(CollectionID)
```

Java
```
mgobjMegaObject.getRoot().getClassDescription(ClassID)
mgobjMegaObject.getRoot().getCollectionDescription(CollectionID)
```

ClassID represents here the absolute identifier of a MetaClass or MetaAssociation – this can be supplied in the form of a field. Regarding CollectionID, this can be the identifier of a MetaClass, MetaAssociation, MetaAssociationEnd or Selection.

In earlier versions, or in more generic code, you access the collection of ClassDescriptions of MetaClasses via collection:

VB Script
```
myObject.getRoot.GetCollection("~Ffs9P58kg1fC[ClassDescriptions]")
```

Java
```
mgobjMegaObject.getRoot().getCollection("~Ffs9P58kg1fC[ClassDescriptions]")
```

The following introduces the object Model corresponding to ClassDescriptions and CollectionsDescriptions.

Concept availability start dates are indicated in brackets.

ClassDescription:

Main properties:

- **Name**: name in current language
- **GUIName**: name in the user interface language
- **Rpbid**: absolute identifier in base64 (you can use GetID to get the identifier in internal format)

- **Level**: concept technical level, corresponding to MetaAttribute Technical Level
- **Visibility**: bit field defining the object visibility (0x20000:Extension)
- (2007) **Abstraction**: abstraction level (1 for abstract class, 0 for concrete class)
- (2009) **Location**: MetaClass location (S:System D:Data L:External)

Main collections:

- **Description**:

  List that includes only one item: the CollectionDescription corresponding to the ClassDescription

- **Pages**:

  Property page list defined on the object, including implicit pages (see PageDescription)

- **Groups**:

  Property group list defined on the object, including implicit groups (see GroupDescription)

- **UpperClasses**:

  UpperClass list, seen as *ClassDescription*

- **LowerClasses**:

  LowerClass list, seen as *ClassDescription*

- (2007) **CommandAccessor**:

  MetaCommand list explicitly defined on the class (see *CommandDescription*)

CollectionDescription:

Main properties:

- **Name**: name in current language
- **GUIName**: name in the user interface language
- **Rpbid**: absolute identifier in base64 (you can use GetID to get the identifier in internal format)
- **Level**: concept technical level, corresponding to MetaAttribute Technical Level
- **Visibility**: bit field defining the object visibility (0x20000:Extension)
- **Order**: order number
- **Major**: indicates that the collection corresponds to a major MetaAssociationEnd
- **Cardinal**: indicates the collection maximum multiplicity (1, U, or N)
- **LType**: absolute identifier of the link type of the collection

- **Permission**: update permission via the user interface for this collection objects
- **Mandatory**: indicates the collection minimum multiplicity (0 or 1)
- (2007) **PhysicalClassID**: collection native MetaClass identifier; this can be a MetaAssociationEnd, a MetaAssociation, a MetaClass or a Request, or can be not defined
- (2007) **Opposite**: opposite MetaAssociationEnd identifier
- (2007) **Association**: MetaAssociation identifier
- (2007) **TargetClassID**: source MetaClass identifier (of the collection)
- (2007) **SourceClassID**: target MetaClass identifier
- (2007) **Abstraction**: abstraction level (1 for abstract class, 0 for concrete class)
- (2009) **Location**: Association location (S:`System` D:`Data` L:`External`)
- (2009) **SourceTypeID**: source MetaClass identifier, abstract in the case of a generic association
- (2009) **TargetTypeID**: target MetaClassidentifier, abstract in the case of a generic association
- (2009) **AliasID**: collection alias identifier, if this exists
- (2009) **RootID**: generic association identifier if we are on an alias.

Main collections:

- **Properties**: list of the properties (see *PropertyDescription*)
- **Collections**: list of the collections, seen as CollectionDescription
- **ExternalRefs**: list containing, if it exists, the CollectionDescription corresponding to the standard link to external reference
- **Characters**: list containing, if it exists, the CollectionDescription corresponding to the standard link to keyword
- **ImageFormats**: list of image formats defined for this collection (see *ImageFormatDescription*)
- (2009) **Concretes**: list of collections corresponding to concrete classes accessible from a generic association
- (2009) **MainProperties**: list of the main properties (i.e. those that are not translations)

From MEGA 2007, functions are available on CollectionDescriptions:

VB Script
```
Function CollectionDescription.GetOperatorBehavior(OperatorId) As String
```

Java
```
Function CollectionDescription.callFunction("GetOperatorBehavior",
OperatorId) As Integer
```

This function enables to know the operator behavior given as a parameter according to the MetaAssociationEnd matching the collection

The returned value corresponds to the Behavior: (65:Abort 83:Standard 76:Link 68:Deep)

VB Script
```
Function CollectionDescription.IsSuperClassOf(ClassId) As Boolean
```

Java
```
Function CollectionDescription.callFunction("IsSuperClassOf", ClassId) As
Boolean
```

VB Script
```
Function CollectionDescription.IsSubClassOf(ClassId) As Boolean
```

Java
```
Function CollectionDescription.callFunction("IsSubClassOf", ClassId) As
Boolean
```

VB Script
```
Function CollectionDescription.IsClassAvailable(ClassId) As Boolean
```

Java
```
Function CollectionDescription.callFunction("IsSubClassOf", ClassId) As
Boolean
```

These functions enable management of correspondence between concrete classes and abstract classes.

IsClassAvailable enables testing whether a corresponding class object can be inserted in the collection

VB Script
```
Function CollectionDescription.Specializations As MegaCollection
```

Java
```
Function CollectionDescription.callFunction("Specializations") As
MegaCollection
```

In MEGA 2007, enables listing of collections corresponding to concrete classes accessible from a generic association. Replaced in 2009 by the collection:

VB Script
```
CollectionDescription.Concretes
```

Java
```
Function CollectionDescription.invokePropertyGet("Concretes") As
MegaCollection

Function CollectionDescription.SameFamily(CollectionId) As Boolean
```

VB Script

Java
```
Function CollectionDescription.callFunction("SameFamily") As Boolean
```

True if both collections correspond to the same generic MetaAssociation.

Collections accessible from these two types of description lead us to define the following sub-descriptions:

PropertyDescription: description of a property: a property can be a MetaAttribute, TaggedValue or 'LegAttribute (MetaAssociationEnd seen as property)

Main properties:

- **Name**: name in the current language

- **GUIName**: name in the user interface language
- **Rpbid**: absolute identifier in base64 (use GetID to get the identifier in internal format)
- **Level**: concept technical level, which corresponds with the MetaAttribute Technical Level
- **Order**: order number
- **Abbreviation**: property abbreviation (short name)
- **Type:** basic type of the property
- **Format**: property format, corresponding to MetaAttribute Type attribute(X:String 9:Numerical 1:Boolean S:Short L:Long D:Date A:Text B:BinaryText Q:Binary H:64bits F:DoubleFloat)
- **Tabulated**: External format of the property, corresponding to the MetaAttribute Format attribute (S:Standard F:Enumeration T:EnumerationOpened D:Duration P:Percent E:Double O:Object Z:Signed)
- **Length**: property internal length
- **ASCIILength**: Length of property in its ASCII representation
- **ExternalLength**: Length of property in its external representation
- **Occurrence**: indicates that the property (in H Format) represents a MEGA object
- **TextFormat:** identifier of text format managing the property
- **Translatibility:** indicates that the property is translatable (0 or 1)
- **LCID:** identifier of language in which the property is restored
- **Index**: indicates that the property is an index (U:Unique S:UniqueCaseSensitive N:NonUnique)
- **FromLink**: indicates that the property comes from the MetaAssociationEnd (0 or 1)
- **Permission**: bit field characterizing the property UIPermission
- **Mandatory**: indicates that the property is mandatory (0 or 1)
- **Visibility**: bit field characterizing the object visibility (0x20000:Extension 0x1:OutOfList 0x4:Administration 0x40000:Localization 0x10000:NamePart 0x200000:HasDefault 0x400000:DefaultButton)
- **UpdateToolID:** binary attribute containing the identifier of the component managing update of the property. Indicates that the property is not updated by standard method
- **Substitution:** identifier of the attribute substituting the property for this MetaClass
- **OnCreation:** bit field indicating behavior of the property at object creation.

This field is restored in the form of a character and we should test asc(value) (0x1:DetailedBehavior 0x2:UpdatedOnCreation 0x10:Mandatory 0x20:UpdatableOnlyDuringCration)

- (2007) **PhysicalClassID**: identifier of the native MetaClass motivating the property; this can be a MetaAttribute, TaggedValue, MetaAssociationEnd… or can be unspecified
- (2007) LanguageID: identifier of the language (when the property is a translation) (2007) **Heritability**: indicates that the property is heritable
- (2007) **DefaultValue:** property default value
- (2007) **RootID**: identifier of the root property, in the case of a translation

Main collections:

Values: lists defined tabulated values (see **AttributeValueDescription**)

AttributeClasses: when the property is an Occurrence, lists the potential MetaClasses of this occurrence (when defined) in the form of a ClassDescription

(2009) **Translations:** property translation list

AttributeValueDescription:

Describes a tabular value defined for a property

Main properties:

- **Name**: name in the current language
- **GUIName**: name in the user interface language
- **Rpbid**: absolute identifier in base64 (use GetID to get the identifier in internal format)
- **Level**: concept technical level, which corresponds with the MetaAttribute Technical Level
- **Order**: order number
- **Value**: ANSI value of the tabular value

Main collections:

- **ImageFormats** lists defined image formats for this tabulated value (see *ImageFormatDescription*)
- ***PageDescription***: describes the property page; This page can be implicite, i.e. it does not correspond to a MEGA object (example: Administration page).

Main properties:

- **Name**: name in the current language
- **GUIName**: name in the user interface language

- **Rpbid**: absolute identifier in base64 (use GetID to get the identifier in internal format)
- **Level**: concept technical level, which corresponds with the MetaAttribute Technical Level
- **Order**: order number
- **LType**: identifier of the page type (defines the tab in which it is supposed to appear)
- **Group:** identifier of the group that motivates the page
- **Guid**: identifier of the macro that implements the page
- (2007) **Heritability**: indicates the page heritability (0 or 1)

GroupDescription

Description of a property group. If the group is implicit, it does not correspond to a MEGA object.
Main properties:

- **Name**: name in the current language
- **GUIName**: name in the user interface language
- **Rpbid**: absolute identifier in base64 (use GetID to get the identifier in internal format)
- **Level**: concept technical level, which corresponds with the MetaAttribute Technical Level
- **Order**: order number
- **LType**: identifier of the group type (defines the tab in which it is supposed to appear)
- (2007) **Heritability**: indicates the group heritability (0 or 1)

Main collections:

- **Properties**: group property list (see PropertyDescription)
- **Pages**: group associated page (see PageDescription)

ImageFormatDescription

Main properties:

- **Name**: name in the current language
- **GUIName**: name in the user interface language
- **Rpbid**: absolute identifier in base64 (use GetID to get the identifier in internal format)
- **Level**: concept technical level, which corresponds with the MetaAttribute Technical Level
- **Abbreviation**: standard file extension
- DefaultUsed: indicates that ImageFormat is not explicitly defined and that the file used is the default file.

(2007) CommandDescription

Main properties:

- **Name**: name in the current language
- **GUIName**: name in the user interface language
- **Rpbid**: absolute identifier in base64 (use GetID to get the identifier in internal format)
- **Level**: concept technical level, which corresponds with the MetaAttribute Technical Level
- **Order**: order number
- **Heritability**: indicates the commandaccessor heritability (0 or 1)

Comments regarding ClassDescription, XXXDescription, "Name" and other properties

XXXDescriptions are MegaObjects and not MetaClasses.

You cannot use usual MetaAttribute identifiers with these objects.

For example, a search on a MetaClass name returns an error:

VB Script
```
Print mgobjMyObject.getRoot.getclassdescription("~gsUiU9B5iiR0[Organizational
Process]").getprop("~210000000900[Name]")
```

Java
```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgobjMegaObject.getRoot().getClassDescription("~gsUiU9B5iiR0[Organizational
Process]").getProp("~210000000900[Name]"));
```

How to do this?

Find the identifier of the "Name" property (different from the "Name" MetaAttribute) of MegaObject ClassDescription.

To do this, examine the ClassDescription object type:

VB Script
```
mgobjMyObject.getRoot.GetClassDescription("~gsUiU9B5iiR0[Organizational
Process]").GetTypeObject().Explore
```

Java
```
mgobjMegaObject.getRoot().getClassDescription("~gsUiU9B5iiR0[Organizational
Process]").getTypeObject().invokeMethod("Explore");
```

You can therefore find the "Name" property and copy/paste in the form of a MegaField.

In this way you can use the ClassDescription:

VB Script
```
print mgobjMyObject.getRoot.getclassdescription("~gsUiU9B5iiR0[Organizational
Process]").getprop("~oKdcP5epmcfC[Name]")
```

Java
```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgobjMegaObject.getRoot().getClassDescription("~gsUiU9B5iiR0[Organizational
Process]").getProp("~oKdcP5epmcfC[Name]"));
```

Note above that the « Name » property identifier is different from the « Name » MetaAttribute one.

9.2 Property Pages

9.2.1 Accessing the description of an object Property Pages

Availability: from 2005 release

This component obtains a description by code of properties pages of an object.

Retrieving the list of pages and tabs of an object

The propertiesdialog method of a MegaObject can now be used as a function and returns a properties page description component.

```
Set PageDescription = mgObject.PropertiesDialog("Description")
```

This component is a page enumerator, and for this purpose it includes Count and Item standard methods. It returns MegaPropertyPage objects.

These objects are identified by the CLSID of the page, enabling their query in the enumerator.

All visible object pages are accessible via this enumerator.

The script below enables simple test of component on the first MetaClass:

VB Script

```
set ppcol = MetaClass.item(1).propertiesdialog("Description")
print ppcol.count
for each ppi in ppcol
print ppi.getID & ppi.parentID & ppi.level & " : " & ppi.name
if ppi.level > 1 then print " Parent Is " & ppcol.item(ppi.parentID).name
set cnt = ppi.Component
if not cnt is nothing then
cnt.Content.Explore
end if
next
```

Java

```
ComObjectProxy mgcomPpcol = (ComObjectProxy)
mgRoot.getCollection("~P20000000c10[MetaClass]").get(1).invokeFunction("propertiesdialog",
"Description");
int iSize = (Integer) mgcomPpcol.invokeFunction("count");
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", iSize);
for (int j = 1; j <= iSize; j++) {
                MegaPropertyPage mgobjPpi = new MegaPropertyPage((MegaCOMObject)
mgcomPpcol.invokeFunction("item", j));
                mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", mgobjPpi.getID() + " " +
mgobjPpi.parentID() + " " + mgobjPpi.level() + " : " + mgobjPpi.name());
                if (mgobjPpi.level() > 1) {
                  ComObjectProxy mgC = (ComObjectProxy) mgcomPpcol.invokeFunction("item",
mgobjPpi.parentID());
                  String strA = (String) mgC.invokeFunction("name");
```

```
      mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", "Parent Is " + strA);
    }
    MegaPropertyPageComponent cnt = mgobjPpi.component();
    if (cnt != null) {
      cnt.content().invokeMethod("Explore");
    }
  }
```

Interface de l'objet

```
MegaPropertyPage.GetID as String
```

CLSID of the page, in form {00000000-0000-0000-0000-000000000000}

```
MegaPropertyPage.Name as String
```

Page name

```
MegaPropertyPage.Order as Long
```

Page order number

```
MegaPropertyPage.Default as Boolean
```

Indicates if the page is the default page (visible at first activation)

```
MegaPropertyPage.ParentID as String
```

CLSID of the parent page if current page is a subtab

```
MegaPropertyPage.Level as Long
```

Page subtab level: 1 indicates main page, 2 page presented in subtab, …

```
MegaPropertyPage.IsTab as Boolean
```

Indicate if page is itself a tabbed box. In this case it is the parent of all pages it contains

```
MegaPropertyPage.Component as MegaObject
```

Page description, if supported by the component implementing the page. If not, is Nothing

Mega 2009:

`MegaPropertyPage.TypeID`

Identifier of the page type

`MegaPropertyPage.SourceID`

Identifier of the page source that can be:

- Identifier of underlying MetaPropertyPage when this exists
- Identifier of MetaAttributeGroup when this is not associated with a page – if the page comes from a MetaAttributeGroup
- Identifier of page type (for Tabs containing pages, and for generic pages)
- Another identifier in other cases

Page component

Pages implementing it (currently only standard pages) make available a auto-description accessible by the method. Component of the MetaPropertyPage object.

This object is an explorable standard MegaObject (as the script example shows).

It has the following attributes:

`Name as String ~oKdcP5epmcfC[Name]:` element title

`Nature as String ~VxJRO58xtMuC[Nature]:` Control nature, conforming to configuration syntax:

```
"Static"
"Edit";
"Text"
"MegaEditCheck"
"ComboBox"
"DropDownList"
"CheckBox"
"3StateCheckBox”
"ComboBoxMenu"
"DropDownListMenu"
"ComboLinks"
"StaticMenu"
"EditMenu"
"RemotingDropDownListMenu"
"ListView"
"TreeView"
"ActiveX"
"HelpComment"
"Schedule"
"Button"
```

Order as Long ~5gs9P58ye1fC[Order]

SourceID as Variant ~Dgs9P58(e1fC[SourceID]: characterizes the property displayed by the control. Usually, the absolute identifier of a MetaAttribute.

Style as Long ~)0nfa1WYhchD[Style]: bit field characterizing the control

```
#define ACWISTYLE_NOTITLE 0x0010000
#define ACWISTYLE_TITLEUP 0x0020000
```

```
#define ACWISTYLE_DEFAULTED 0x0040000
#define ACWISTYLE_BOTTOMALL 0x0080000
#define ACWISTYLE_BORDERED 0x0100000
#define ACWISTYLE_EXCLUDED 0x0200000
#define ACWISTYLE_DISABLED 0x0400000
#define ACWISTYLE_HIDDEN 0x0800000
#define ACWISTYLE_DYNAMIC 0x1000000
#define ACWISTYLE_REMOTING 0x2000000
#define ACWISTYLE_CLIPLEFT 0x002
#define ACWISTYLE_CLIPRIGHT 0x004
#define ACWISTYLE_CLIPTOP 0x008
#define ACWISTYLE_CLIPBOTTOM 0x010
#define ACWISTYLE_CLIPMLEFT 0x020
#define ACWISTYLE_CLIPMRIGHT 0x040
#define ACWISTYLE_CLIPMBOTTOM 0x080
#define ACWISTYLE_CLIPMTOP 0x100
#define ACWISTYLE_NOVCLIP 0x1000
#define ACWISTYLE_NOHCLIP 0x2000
```

Group as String ~0es9P5eQf1fC[Group]: name of the Group in which the control has been placed

Options as String ~T2zVa10))bZD[Options]: options defined for the control in configuration

Read Only as Boolean ~3es9P5ORf1fC[Read Only]

Width as Long ~ths9P5OOf1fC[Width]: control width (in dialog Units)

Height as Long ~uhs9P5eOf1fC[Height]: control height (in dialog units)

Left as Long ~whs9P58Pf1fC[Left]: if specified, control left margin (in dialog units)

Top as Long ~)hs9P5OQf1fC[Top]: if specified, position of top of control relative to group (in dialog units)

Kind as Short ~a2zVa1W00cZD[Kind]: precise nature of control

ControlID as Binary ~v0nfa10XhchD[ControlID]: CLSID of component implementing the control

MapID as Variant ~PDfkighQx400[MapID]: absolute identifier of MAP of control (see Parameterization)

LinkID as 64Bits ~V20000000z50[LinkID]: if the control is obtained from a link, absolute identifier of the MetaAssociationEnd.

ObjectID as 64Bits ~qM44Q5OUd4xC[ObjectID]: absolute identifier of the object of which the control displays a property

ID as Variant ~jKdcP5OomcfC[ID]: Control internal identifier

Child controls of a control are accessible by the Child Collection (« AttributeControl »)

~7fs9P58ig1fC[AttributeControl]

9.3 Accessing MEGA Object menus using APIs

The menu of a MEGA object or object collection is accessible via the **MegaCommandManager** component.

The **CommandManager** operator enables access to this component, from a MegaObject or MegaCollection (therefore from a MegaItem).

This component describes the standard menu of the object or collection, as it appears for example in the explorer (the collection menu appears when you click a folder)

Methods of this component are:

```
MegaCommandManager.Object As MegaItem
```

Returns the MegaObject or MegaCollection on which the CommandManager has been invoked

```
Sub MegaCommandManger.TrackPopup(Optional Line As Integer,Optional Column As Integer,Optional Options As String)

Function MegaCommandManger.TrackPopup(Optional Line As Integer,Optional Column As Integer,Optional Options As String) as Long
```

Displays the object pop-up menu. This method can therefore be called only in interactive mode.

Line and Column indicate the absolute position on the screen of the top left-hand corner of the menu. If they are not specified, the position of the mouse cursor at the time of call is used.

Two options can appear in Option (any separator):

- `"NoProperties"`: the "Properties" command is not added to the menu
- `"ShowOnly"`: the menu is displayed but the possibly selected command is not called

When the function is called, the index of selected command is returned, or 0 if no command was selected, or -1 if the "Properties" command was selected.

```
Sub MegaCommandManager.Invoke(CommandID As Variant)
```

Execute the command.

CommandID can be the name of the command or its index. If several commands have the same name, the first one is invoked.

```
Sub MegaCommandManager.InvokeStandard(CommandID As Variant)
```

From 2007 release

This method allows invocation of a standard menu command. Managed commands are:

- `"Copy"`: "Copy" command

- "Destroy": "Destroy" command
- "Unlink": "Unlink" command
- "Explore": "Explore" command

You can invoke the following generic commands on objects:

- "Open": default command (open a menu if there are several commands)
- "AddToFavorites": "add to favorites" command

You can invoke the following commands on collections:

- "Paste": "Paste" command
- "Create": "Create" command in in-place mode
- "QueryCreate": "Create" command in interactive mode
- "Link": "Link" command
- "ReOrder": "Reorder" command

This method can be used with or without a return parameter.

If there is a return parameter (here myResult), it is 'True' if the command exists for the CommandManager and has therefore been launched. Otherwise, the command does not exist in this context.

If there is no return parameter and the command does not exist, the method triggers an error.

You can explicitly access to the list of commands that are defined in the Menu:

MegaCommandManager.Commands As MegaCollection

Returns the exhaustive list of object commands, including deactivated and invisible commands, and explicit sub-menus.

MegaObjects contained in the resulting collection include the commands – These are not MEGA occurrences.

Their properties are as follows:

CommandItem.Name As String:

Command name (in the language of the current environment or site)

CommandItem.Index As Long:

Command internal number. This number is not stable and should not be used with an instance of CommandManager other than that with which the CommandItem was obtained.

CommandItem.Style As Long:

Bit field characterizing the command style and visibility. Significant bits are:

- for the styles:

POPUP 0x001 (1) Indicates that the command is a pop-up menu
CHECKBOX 0x002 (2) Indicates that the command is a check box: the corresponding menu element can appear with a check mark
RADIOBUTTON 0x08 (8) Indicates that the command is a radio button: the corresponding menu element can appear with an exclusive check mark: a single

element can be marked among the RadioButtons of the same group (see CommandItem.Group below)

- for the status:

CHECKED 0x100 (256) Indicates that the element (CheckBox or RadioButton style) is marked in the context of the occurrence.

DISABLED 0x400 (1024) Indicates that the element is grayed and cannot be active in the context of the occurrence.

CommandItem.Category As Long:

Bit field specifying category of the element as defined in the corresponding CommandAccessor. Significant bits are:

DESCRIPTION 0x0010 Object description data entry commands (Open, Zoom, Flowchart,...)
ENTRY 0x0020 Object characteristics data entry commands (Attributes, Operations, Navigability->, Cardinality->)
ACTION 0x0004 Object action/activation commands (Generate, Derive, Prototype, Quantify...)
DOCUMENTATION 0x0040 Object Publication/Documentation commands (Document, Reference,...)
DISPLAY 0x0002 Object presentation commands (View, Format, Drawing)
CONTAINER 0x0080 Object commands in container (Cut, Copy, Disconnect)
ADMIN 0x0008 Object administration commands (Explore, Delete, Compare,...)
ADMINEX 0x0400 Commands Manage >
LINK 0x0200 Commands Connect >
NEW 0x0100 Commands New >

EXPORT 0x0800 Tool/Export menus in desktop

INPUT 0x0001 Enumerator commands

REVERSE 0x1000 Tool/Reverse menu commands in desktop
LOCALISATION 0x2000 Menu commands relating to languages – managed in desktop and diagram

STANDARDCMD 0x10000 Standard commands

STANDARDOPEN 0x20000 Opening command

NOENUMLISTVIEW 0x100000 Filtre: Commands not available outside ListViews
ENUMLISTVIEW 0x200000 Filtre: Commands specific to ListViews

CommandItem.Order As Long: Command order number. This number is used to sort commands of the same category when displaying the menu.
CommandItem.Group As Long: Indicates command group. The value returned is 0, except for elements of RadioButton type and elements included in an explicit sub-menu (that is not derived from a category). In this case the Group value corresponds to the index of the pop-up element under which the commands appear.

9.4 Managing scanners

9.4.1 Using or not using the scanner library

Scanner library enables optimized metamodel browsing, using cache.

The system repository access number is lower: the first information request feeds the cache while for the second same information request the cache is only read.

Very important:

This cache library enables only to browse **system repository** data and **system repository** metamodel.

Metamodel browsing principle:

We consider that we start from an idabs (absolute identifier) of a concrete object.

Scanner functions enable recovery of information of the object itself. (Current availability of these functions depends on language)

The library enables recovery of objects (and their attribute values) located at the other end of a link.

Link attributes can also be recovered.

Browsing can continue from the context received (MegaCollectionScannerContext).

To keep benefit of the cache

When scanners are used, we want to benefit from use of the cache. During browsing, we do not need to access the system repository. System repository access is very costly.

During browsing we should avoid accessing the system repository, calling only functions not contained in the scanner library so as to avoid losing the benefit of the cache.

When to use scanner functions:

The three following points are mandatory:

- we want to read system repository data
- this data does not change frequently. If it changes every day, it is of no use.
- check that MetaClasses and links are eligible for browsing. Not all system repository MetaClasses are eligible. In case of doubt, ask.

We must be able to position the **"Extended Properties" attribute of** MetaClasses on "compiledTechnicalData" or "compiledMetaData"

(Properties / Characteristics / Advanced / Extended properties: "compiledTechnicalData")

Attention: Ask before modifying.

What can influence the choice to use or not use:

- Quantity of data to recover. The more you want to recover, the greater the interest.
- Frequency of use. The more frequent the use, the greater the interest.

Limitations

Today, (27 February 2012) scanner library functions are not yet complete:

- varchars cannot be recovered by this library (the function has been requested and should arrive)
- there are limitations in recovery of texts (sizes, languages, efficiency). Check for required use.
    - Translatable names are not in cache.
    - Parameterization texts are not in cache.
    - Comments are not in cache.
- The calculated attribute should not be read.

When is the cache active?

It is active when the metamodel is compiled, and only on data carried by MetaClasses that have the Extended Properties attribute: "compiledTechnicalData" or "CompiledMetaData"

It is also preferable that links browsed have this same property.

Attention:

- If the MetaClass does not have the "compiledTechnicalData" attribute, the scanner library returns information on objects, but the cache is not used.
- If the MetaClass belongs to the "data" repository and not to the "system" repository, the scanner library does not return information on objects.
- If the metamodel is not compiled, the scanner library returns data, but this will not be in cache.

Forbidden:

Do not use scanners to read calculated attributes. This information cannot be stored in a cache as it has to be calculated each time.

9.4.2 Theoretical operating principles (without language dependence)

Starting point

The starting point is the idabs (absolute identifier) of a concrete object.

It is the starting point of the MetaModel browsing.

Create a reception class

Create a new class that includes an **OnItem** function.

This OnItem function will be invoked each time an object at the other end of a link is found.

Configuring browsing path

Use a browsing scanner function, with the following parameters:

- starting idabs: [IDOBJ_DEPART]
- the idabs of the link you want to browse [IDLINK]
- the class you just created and that includes the OnItem function
- the idabs of the properties you want to read in the objects on the other end of the link
- the idabs of the properties you want to read on the link

Starting browsing

Call the browsing scanner function

The library searches for objects at the other end of the link [IDLINK] from the indicated start object [IDOBJ_START].

The OnItem function is called for each object found.

In this OnItem function, we can recover information on the received object. This information will however be limited to attributes declared at the time of configuration.

Continuing browsing

In an OnItem function, we can restart a browsing scanner function to continue from the found object, indicating a browsing link from this object.

Implementing a scanner

See Using scanners in Java p.113.

9.4.3 Using scanners in Java

Using scanners to browse the metamodel and data of the system repository.

Introduction

The scanner library enables metamodel browsing, optimized by use of caches.

The number of accesses to the system repository is consequently reduced.

Metamodel browsing principle:

- We consider that we start from an idabs (absolute identifier) of a concrete object.
- The library enables to retrieve objects (and their attribute values) located at the other end of a link.
- Link attributes can also be retrieved.
- Browsing can continue from the context received (MegaCollectionScannerContext).

Prerequisites:

Read the following sections:

- Using or not using the scanner library p. 111

- Theoretical operating principles (without language dependence) p. 112.

API:

Location: **api.jar**

Classes:

```
com.mega.modeling.api.util
- MegaResources
- MegaCollectionScannerContext
```

MegaResources:

NameFormat GUIName

To obtain the GUIName to display it, you can use NameFormat.guiShortName.

NameFormat.guiName contains a path, with a path separator.
An inheritance phenomenon exists.

For example, if you request a guiName and this is empty, this returns the name.

To deactivate inheritance, use keyword **direct**

Example: to obtain the guiname without inheritance, request `NameFormat.directGuiShortName`

CollectionScanMode

Methods used by scanCollection.

We should indicate a search method: synchronous/asynchronous/...

Using synchronous is simpler.

So that objects will be returned sorted, use **CollectionScanMode.ordered.**

Objects will be sorted according to the "Order" attribute of the object.

Since it is a bit field, you can add options.

Example: `scanMode= CollectionScanMode.ordered + CollectionScanMode.synchrone`

scanCollection

You can indicate a search for information in parallel using CollectionScanMode.parallel. This will run multiThreads. However, before using the multithread, always ask if it is really justified, since it will require additional tests.

Use the CollectionScanner interface as parameter => create functions that have this interface.

The attList parameter contains a parameter list, that is a megaField separated by the required separator (comma, semi-colon,..)

The function recognizes megafields.

Megafields

Searches are always executed with MegaPath ids (MegaObject.megaField)

Reminder: megaField format: ~string1[string2]

Parallelism:

When the multithread is used, use synchronize in java to obtain the information.

CollectionScanner

The Object endId parameter signifies the ID of the MetaAssociationEnd (MAE)

Abort in onIntem function

If you know you will receive only a single object, you can call abort or cancel when the object has been received.

Instancing a MegaResources

```
MegaResources mr = megaRoot.currentEnvironment().resources();
```

scanCollection parameters

- 1st parameter:
  - either the identifier of the MegaObject from which you start (myMegaObject.getID())
  - or an idabs
  - or a moniker

    A moniker is a string including a reference to an object or an address of an object, or a path to a file, or possibly information on the object. The moniker format depends on where it is used in the code. Information included in the moniker varies.

    This can be, for example: a tilde character (~) followed by an identifier that allows retrieving an object. This identifier can be, for example, an idabs in Base64, or a path, or any identifier type.
  - or a megafield
  - or use the Context when we execute onItems in cascade: onitem called from a path which already exists in an onitem. This context is received as parameter of OnItem.
- 2nd parameter:
  - identifier of the link (MAE) to which you want to go.
- Last parameter:
  - **attlist**

    List of attributes you want to place in the cache.

    Take care, you should not indicate titles (name/guiname) in this list, since it should not hide the language. A second language cache will be called, but not here. Here it is a browser cache.

    To indicate that an attribute is on target, we should add ':T' after the attribute.

    An attribute without ':T' signifies that it is a link attribute.

    Search for a title (name, guiname, etc...) should be done in onitem.

OnItem:

Called for each object located at the other end of the MAE link

On the Context object (MegaCollectionScannerContext) received as parameter of OnITem, we can call the following functions:

- **`property`** provides information on the link property
- **`targetProperty`** provides information on the property of the object at the other end of the MAE.

This finds the value of attributes previously indicated in attList.

The MegaCollectionScannerContext received enables restart of scancollection from the current object to continue browsing in the OnItem function.

To retrieve a title in onItem:

1. Declare MegaResources in the scanner.
2. The scanner builder receives MegaResources.
3. In onItem; use name of MegaResources + context and NameFormat to obtain a title

We therefore have:

```
public class MyScanner implements CollectionScanner {
//Declaration of MegaResources in the scanner
private MegaResources m_megaResources;
public MyScanner(final MegaResources megaResources) {

this.m_megaResources = megaResources;
}
}
/** receipt of an object at each call on OnItem and of its properties */
public void OnItem(final MegaCollectionScannerContext context, final Object
endId) {
String strResult = this.m_megaResources.name(context, NameFormat.guiName);
}
```

See NameFormat at the end of this section.

To retrieve a comment on OnItem

Proceed as in the previous example, to access megaResources in OnItem.

Then use the comment function, indicating a CommentFormat

```
String strResult = this.m_megaResources.comment(context, CommentFormat.standard);
```

To retrieve the MetaCLassIdAbs in OnItem

To determine which object type OnItem will return, you can request its MetaClass.

Note: there is no point in declaring this attribute in attList. It is an attribute that can always be requested.

```
public void OnItem(final MegaCollectionScannerContext context, final Object endId) {

String strIdabsMetaClass =
context.targetProperty(VocGenericObjectSystem.MA_ObjectMetaClassIdAbs);
}
```

VocGenericObjectSystem is the class of vocabulary of GenericObjectSystem. It is generated from Mega.

```
public static final String MA_ObjectMetaClassIdAbs = "~d20000000T60[X]";
```

To retrieve a varchar type field for link attributes

If you want to retrieve an attribute declared in Mega as a varchar, you should use the specialized function **text**.

Declare the attribute in attlist.

```
String monVarChar = context.text(megaField, field content type);
```

(at March 2012):

- this only functions for link attributes.
- field content type is not used

To retrieve attribute of an object knowing only its idabs

This is particularly useful for the browsing start object, for which we have not yet browsed a link.

```
String a1 = ObjectIdabs // idabs of the object belonging to the system repository
String a2 = MA_AttributeIdabs; // required attribute idabs
StringBuffer s = new StringBuffer(); // Request result
MegaRoot().currentEnvironment().resources().basedObj.invokeMethod("GetSystemObjectProp", a1, a2, s);
if ((s != null)) {
...
}
```

MegaCollectionScannerContext

Retrieving objects linked by MAE with functions of MegaCollectionScannerContext:

- targetProperty: property of the object on the opposite side of the link
- property: property of the link

Issues resolved by this:

- system repository queries in cache
- requested fields (not values) in cache
- browsed links in cache
- speed of execution
- enables dynamic recalculation of information, according to system language, authorizations, metamodel modifications.

Warnings

When the subject of browsing is small or there is minimum processing, parallelism should not be used as it executes too rapidly.

No result will be returned.

NameFormat

```
public interface NameFormat {

public static final String name = null;

public static final String localName = "LocalName";

public static final String shortName = "ShortName";

public static final String abbreviation = "Abbreviation";

public static final String guiName = "GuiName";

public static final String guiLocalName = "GuiLocalName";

public static final String guiShortName = "GuiShortName";

public static final String guiAbbreviation = "GuiAbbreviation";

public static final String directName = "DirectName";

public static final String directLocalName = "DirectLocalName";

public static final String directShortName = "DirectShortName";

public static final String directAbbreviation = "DirectAbbreviation";

public static final String directGuiName = "DirectGuiName";

public static final String directGuiLocalName = "DirectGuiLocalName";

public static final String directGuiShortName = "DirectGuiShortName";

public static final String directGuiAbbreviation = "DirectGuiAbbreviation";

}
```

9.4.4 VB Script examples

The following is a vbscript scanner example browsing the link "Desktop" to "Desktop Scope"

Prerequisite

Read the following sections:

- Using or not using the scanner library p. 111
- Theoretical operating principles (without language dependence) p. 112.

Description

The function **GetScopes**() returns a table of Scopes

This example illustrates a method of retrieving information found by the scan.

- When browsing in **OnItem**, information is placed in a Scope (Class Scope) object
- Each Scope object is placed in a table attribute of the scanner class (**ScannerScopes.**mgScopes()
)
- The **GetScopes** function starts the scanner and retrieves the table.
- The **main** starts **GetScopes**, retrieves the table then displays the retrieved information.

Warning

The following code is suitable for small lists.

For large lists, do not use a table but a collection.

Code

```
'MegaContext(Fields,Types)
Option Explicit

Class Scope
public metaclassidabs
public idabs
public guiname
public scopeType
End Class

Class ScannerScopes
Public mgResource
Public mgToolkit
Public mgbTrouve
Public mgIdAttribute
Public mgScopes()
public nbscopes

Public Sub OnItem(Context, Id)
mgbTrouve =true
nbscopes = nbscopes +1
redim preserve mgScopes(nbscopes)

dim unScope
Set unScope = new Scope

unScope.idabs = mgToolkit.getString64FromID(Id)
unScope.metaclassidabs = Context.targetProperty("~d20000000T60[IdAbs of the
object metaclass]" )
unScope.scopeType = Context.targetProperty("~FYWVgKOgEHmN[Scope Type]" )
unScope.guiname = mgResource.name(Context, "GuiName")

set mgScopes(nbscopes)=unScope

'Context.Abort
End Sub
End Class

Function GetScopes(mgRoot As MegaRoot)

Dim mgScanner
Set mgScanner = New ScannerScopes
Set mgScanner.mgResource = mgRoot.CurrentEnvironment.Resources
Set mgScanner.mgToolkit = mgRoot.currentEnvironment().toolkit()
```

```
dim idDepart
dim idLink
dim listAttributes

idDepart = "~)wIimDhiEftG"
idLink = "~wBYIJ8NgEfDE[Desktop Scope]"
listAttributes ="~FYWVgKOgEHmN[Scope Type]" & ":T"

mgScanner.mgbTrouve = false
mgScanner.nbscopes=0
mgScanner.mgResource.ScanCollection idDepart ,idLink , mgScanner ,1 ,
listAttributes

if(mgScanner.mgbTrouve ) then
GetScopes= mgScanner.mgScopes
else
GetScopes= null
end if

end function

Sub Main()

mgobjMyObject.getRoot.print "GetScopes"
dim mgScopes 'tableau de scopes
mgScopes= GetScopes(GetRoot())

dim i

mgobjMyObject.getRoot.print ""

if(not isnull(mgScopes )) then
for i=1 to ubound (mgScopes)
mgobjMyObject.getRoot.print mgScopes(i).guiname
mgobjMyObject.getRoot.print mgScopes(i).metaclassidabs
mgobjMyObject.getRoot.print mgScopes(i).idabs
mgobjMyObject.getRoot.print mgScopes(i).scopeType
mgobjMyObject.getRoot.print ""
next
end if

End Sub
```

9.5 Others

9.5.1 MEGA TextStream an alternative string concatenation

Availability: From MEGA 2009 SP1 CP4

This component is based on the Microsoft ITextStream interface, used in particular by the FileSystemObject component.

- MEGA *TextStreams* can be opened in writing access or reading access; mixed mode however is not supported.
- MEGA *TextStreams* can correspond to character strings: it is possible to instance a *TextStream* in reading access using a character string, and to instance a *TextSteam* in writing access, which will produce a character string output.
- MEGA *TextStreams* offer the possibility of reading or writing files, natively offering UTF8 and ANSI conversion (writing in UNICODE being possible).

Files opened in read-only can use the following functions of the ITextStream interface:

```
TextStream.AtEndOfStream As Boolean
```

indicates that the reader has reached the end of the stream.

```
TextStream.AtEndOfLine As Boolean
```

indicates that the reader is positioned at the end of a line.

```
TextStream.Read(nbChar As Integer) As String
```

reading the number of requested characters in the stream. If line end markers are encountered during reading, they are not transformed and are therefore returned as-is.

```
TextStream.ReadLine As String
```

reading stream to the next line (character CR) OR end of file. Line break characters do not appear in the returned string.

```
TextStream.ReadAll As String
```

reading integrality of stream in a string; not to be used if the file is potentially large…

```
TextStream.Skip(nbChar As Integer)
```

skip the number of requested characters in the stream and continue reading

```
TextStream.SkipLine As String
```

read forward in stream to the next line (character CR) or end of file.

```
TextStream.Line As Integer
```

number of lines read (including line to read: value cannot be less than 1)

```
TextStream.Column As Integer
```

number of columns read in the last line (including column to read: value cannot be less than 1)

Files opened in read-only can use the following functions of the ITextStream interface:

```
TextStream.Write(Text As String)
TextStream.Write(Text As String) As Integer 'Script Only
```

writing string in stream. Use in the form of function does not generate an exception in the case of failure when writing, but returns a not null value in the case of writing failure:

```
TextStream.WriteLine(Text As String)
TextStream.WriteLine(Text As String) As Integer 'Script Only
```

this value corresponds to the OLE (HRESULT) code corresponding to the error. Note that this form is not defined in the ITextStream interface and is therefore reserved for Script or Late-Binded VB use.

writing the character string in the stream, and adding a line end marker. Use in the form of function does not generate an exception in the case of failure when writing, but returns a not null value in the case of writing failure: this value corresponds to the OLE (HRESULT) code corresponding to the error.

```
TextStream.WriteBlankLines(nbLines As Integer)
```

writing requested line end marker number.

```
TextStream.Line As Integer
```

number of lines written (first being number 1).

```
TextStream.Line As Integer
```

number of columns of last line written. Systematically equals 1 in the case of use of WriteLine or WriteBlanlLines.

For all TextStreams:

```
TextStream.Close
TextStream.Close As Integer 'Script Only
```

The Close function can be used only if the file has been explicitly opened by the component (see function Open described below). Use in the form of function does not generate an exception in the case of failure when writing, but returns a not null value in the case of writing failure: this value corresponds to the OLE (HRESULT) code corresponding to the error.

TextStream creation and opening:

In Script you can create an explicit TextStream with CreateMegaObject function:

```
Set myStream = MegaToolkit.CreateMegaObject("MegaTextStream")
```

Then open this TextStream according to its usage.

Opening a Stream corresponding to a file:

```
Sub MegaTextStream.Open(
Source As Object,
Optional Mode As String,
Optional Format As String)

Function MegaTextStream.Open(
Source As String,
Optional Mode As String,
Optional Format As String) As String
```

Source corresponds to the file name. It can be ignored in specific opening modes detailed above.

Mode: can be **"Write"**, **"Write,Create"** or **"Read"**.

In **"Write,Create"** mode, file creation is possible, else the file must correspond to another existing file.

The default mode is **"Read"** mode.

For files opened in writing access mode, the following keyword can be added: "**NoLineFeed**", indicating that the line end sequence is represented by a CR character alone, not followed by LF when it is generated by functions **WriteLine** or **WriteBlankLines**.

For files open in creation mode, you can add the following key words:

"Temporary": indicates that the file should be created with a "temporary" tag.

"Archive", indicating that the file should be created with the "archive" tag **"WithBOM"**, indicating that format definition characters (UTF8 or UNICODE) should be written in the file header.

Format: by default, format is undefined for reading access files (the reader considers that the format is ANSI, or UNICODE in obvious cases), and ANSI in the current page code for writing access files. It is possible to specify format: In this case, we should also include the mode, the format necessarily being the third parameter of the method.

Format corresponds to the numerical value of LCID in the case of an ANSI file; negative numerical values defined for formats unicode (-1), binary (-3) and utf8 (-4) can also be used, as well as the following keywords: "UNICODE" "BINARY" "UTF8"

When **Open** is used in the form of a function, it does not produce an exception in the event of failure at file opening: The character string returned is empty if opening is successful; in the case of failure, the character string indicates the reason for the error.

Opening a stream corresponding to a character string in reading access:

This operation invokes a method using a *TextStream* as entry system when we have a character string available. In this case, we should not initialize the *TextStream* with the **Open** method, but with the **MapString** method.

```
MegaTextStream.MapString(Input As String)
```

This function also enables reading MEGA text in the form of a TextStream, and therefore to be able to read a text line-to-line.

VB Script

```
myTextStream.MapString myObject.GetProp("Comment", "Display")
Do While Not myTextStream.AtEndOfStream
Print "Line " & myTextStream.Line & " : " & myTextStream.ReadLine
Loop
```

Java

```
MegaCOMObject mgobjMyTextStream = (MegaCOMObject)
mgobjMegaObject.getRoot().currentEnvironment().toolkit().createMegaObject("MegaT
extStream");
mgobjMyTextStream.invokeMethod("MapString", mgobjMegaObject.getProp("Comment",
"Display"));
while ((Boolean) mgobjMyTextStream.invokePropertyGet("AtEndOfStream") == false)
{
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", "Line " +
mgobjMyTextStream.invokePropertyGet("line") + " : " +
mgobjMyTextStream.invokePropertyGet("ReadLine"));
}
```

Opening a stream corresponding to a character string in writing access:

This function enables creation of a stream managed in live memory and, after writing processing, recovery of its content in the form of a character string.

To do this, we specifically use the **Open** method in the following way:

```
myTextStream.Open 0, "BSTRMODE"
```

The stream is also opened in writing access mode: call on **Write**XXX functions enables optimized concatenation of the value written in an allocated character string.

On completion of the writing operation, we can recover the written string using the **GetString** function.

```
myOutput = myTextStream.GetString
```

This function does not duplicate the string; the string used in the stream is transmitted as-is to the variable and the stream is reinitialized with an empty string, in which we can again write.

This function is particularly optimized and adapted to a generation. For example, consider the two following examples:

VB Script

```
Function TestBSTRNative
TestBSTRNative = ""
For i = 1 To 10000
TestBSTRNative = TestBSTRNative & " Writing of the line number " & i & VbCRLF
Next
End Function

Function TestBSTRStream
Set myTextStream = CurrentEnvironment.Toolkit.CreateMegaObject("MegaTextStream")
myTextStream.Open 0, "BSTRMODE"
For i = 1 To 10000
myTextStream.WriteLine " Writing of the line number " & i
Next
TestBSTRStream = myTextStream.GetString

End Function
```

Java

```
public String TestBSTRNative() {
    String strTestBSTRNative = "";
    for (int i = 1; i <= 10000; i++) {
    strTestBSTRNative = strTestBSTRNative + " Writing of the line number " + i +
"\n";
    }
    return strTestBSTRNative;
  }


  public String TestBSTRStream(final MegaRoot mgRoot) {
    MegaCOMObject mgobjMyTextStream = (MegaCOMObject)
mgRoot.currentEnvironment().toolkit().createMegaObject("MegaTextStream");
    mgobjMyTextStream.invokeMethod("Open", 0, "BSTRMODE");
    for (int i = 1; i <= 10000; i++) {
      mgobjMyTextStream.invokeMethod("WriteLine", " Writing of the line number "
+ i);
    }
    return (String) mgobjMyTextStream.invokeFunction("GetString");
  }
```

The TestBSTRStream function, while restoring an identical text, is much faster (x50 on a developer machine).

Opening a stream on a volatile temporary file.

It is possible to create a serializedstream on a temporary file, automatically destroyed at freeing of the component.

This system enables script processing to handle generations that risk exceeding machine memory capabilities.

Such a stream is used in two steps:

First, we open it in writing access mode; we can then generate data. To do this, we use the **Open** method in a specific way:

VB Script
```
myTextStream.Open "", "Write,Create,Temporary,DeleteOnClose","UNICODE"
```

Java
```
mgobjMyTextFile.invokeMethod("Open", "", "Write,Create,Temporary,DeleteOnClose",
"UNICODE");
```

On completion of the writing phase, we call the **SetReadMode** method (reserved for this type of stream) which enables switch to reading mode.

We can then use it as a stream in reading mode; however we can no longer write in it.

Finally the **Close** function destroys the temporary file. If it is not called, the file is destroyed when the component is freed.

We can write the example above using a stream on temporary file:

VB Script
```
Function TestBSTRTempo(mgRoot as MegaRoot)
Set myTextFile =
mgRoot.CurrentEnvironment.Toolkit.CreateMegaObject("MegaTextStream")
myTextFile.Open "", "Write,Create,Temporary,DeleteOnClose","UNICODE"
For i = 1 To 10000
myTextFile.WriteLine "Writing of the line number " & i
Next
myTextFile.SetReadMode
TestBSTRTempo = myTextFile.ReadAll
End Function
```

Java
```
public String TestBSTRTempo(final MegaRoot mgRoot) {

MegaCOMObject mgobjMyTextFile = (MegaCOMObject)
mgRoot.currentEnvironment().toolkit().createMegaObject("MegaTextStream");

mgobjMyTextFile.invokeMethod("Open", "", "Write,Create,Temporary,DeleteOnClose",
"UNICODE");

for (int i = 1; i <= 10000; i++) {mgobjMyTextFile.invokeMethod("WriteLine", "
Writing of the line number " + i);

}

mgobjMyTextFile.invokeMethod("SetReadMode");

     return (String) mgobjMyTextFile.invokeFunction("ReadAll");

     }
```

This function executes slower than the **TestBSTRStream** function (around 20%), but in any case much faster than **TestBSTRNative**.

It should be noted that this implementation is of no great interest in this example, since in any case we must provide allocation of the return character string, here in the **ReadAll** function: In this context, it is not possible to manage a stream with insufficient memory…

10 How to

10.1 Import/Export

10.1.1 Using MEGA Import/Export command options (2005)

Availability: 2005 and previous releases
(later release extensions are not listed)

import / export shared options

`ServerMode = On/Off` (with or without GUI) default: Off

`Meta = On/Off/Full` META occurrences are included - default: On

`Data = On/Off/Full` DATA occurrences are included - default: Full

`Mega = On/Off` MEGA occurrences are included - default: Off

`Technical = On/Off/Full` Technical occurrences are included - default: Full

`MgxFilter = Meta|Data|Technical|HistoricOff|TextOff|KernelOnly|TranslationOff|DateOff|UserOff|HistoricDisable|LogActivityOff None -` default: None

HistoricDisable and LogActivityOff options are 2005 Release specific

HistoricOff option is no longer supported with 2005 Release

`CommandFormat = MGE/MGR/MGL/MGA/XML -` default: MGR

Export specific options

`FileOpen = Rewrite/Append File` File opening mode - Default: Rewrite

`SavableOnly = On/Off/Default -` Default: Default

`LoggableOnly = On/Off/Default -` Default: Default

`DumpObjectLogs = On/Off/Default` Log export - Default: Default

This option is of no use in 2005 Release.

`DumpObjectsOfMerging = On/Off/Default -` Default: Default

XML import specific option

`XmlReport = None|Errors|Warnings|Commited|Skipped`

Import specific options

`Validate = Never/Standard/AtEnd/AtEndOnSuccess -` default: Standard

`ControlIdAbs = On/Off -` default: depends on CommandFormat

`ControlGraph = On/Off -` default: depends on CommandFormat

`Logability = On/Off` Log Activation - default: On

`ReprocessStd = On/Off` Standard Reprocessing - default: Off

`ReprocessUsr = On/Off` User Reprocessing - default: Off

`RepositoryActivity = On/Off` (2005 Release only) - default: Off

`ObjectLog = On/Off/Default` - default: Default

This option is of no use in 2005 Release.

CheckBase option

`CheckMode = Physical/Logical/Total` - default: Total

`CheckTransaction = On/Off` - default: Off

SaveAsCommand specific option (backup deducted from ChangeItems) (2005 Release only)

`DisabledCommandGeneration = On/Off` - default: Off

10.1.2 Accessing the Desktop Context using APIs

Availability: from MEGA 2005 SP3

This component enables an API program to interact with the desktop.

To access to the desktop:

VB Script
```
Set myDesktopContext = CurrentEnvironment.DesktopContext (optional Root As MegaRoot)
```

Java
```
MegaCOMObject myDesktopContext = (MegaCOMObject) mgobjMegaObject.getRoot().currentEnvironment().invokePropertyGet("DesktopContext", mgobjMegaObject.getRoot());
```

- Can be Nothing if there is no DesktopContext
- If the MegaRoot is not specified, any access to MEGA repository leads to the message « An external program tries to access MEGA… »
- If the MegaRoot is specified, the DesktopContext can depend on the MegaRoot.

The desktop context enables:

- Actions (activate, deactivate) on desktop client areas
- Creation of specific client areas (currently only browsers can be created)
- Access to current desktop element (the present version does not enable notification of a change of current object)

DesktopContext is a component conforming to the following object model:

DesktopContext:

VB Script
```
DesktopContext.CurrentObject As MegaObject
```

Java
```
MegaObject mgobjCurrentObject = (MegaObject) myDesktopContext.invokePropertyGet("CurrentObject");
```

Enables access to or modify the current object of the desktop

VB Script
```
DesktopContext.GetRoot As MegaRoot
```

Java
```
MegaRoot mgRootExpl = (MegaRoot)
myDesktopContext.invokePropertyGet("GetRoot");
```

Enables access to the desktop repository.

VB Script
```
DesktopContext.Clients As DesktopClients
```

Java
```
MegaCOMObject myDesktopClients = (MegaCOMObject)
myDesktopContext.invokePropertyGet("Clients");
```

Enables access to the collection of the items of the client area.

DesktopClients:

- `DesktopClients.Count As Integer`

Number of desktop clients

- `DesktopClients.Item(Index As Variant) As DesktopClient`

Enables access to a specific client. Index can be:

the index (Integer from 1 to Count)

the tab identifier (can be entered in Field format)

the tool identifier (can be in entered in Field format)

the tab name

This function returns the first client that verifies the index.

- `DesktopClients.Create(Nature As String, Id As Variante, InitialName As String) As DesktopClient`

Enables to add a new item in the desktop client area:

`Nature`: nature of the new item (Example: « WebBrowser »)

`Id`: tool identifier (can be in entered in Field format). Via this unique identifier, you can reuse an already existing client area

`InitialName`: tab initial name

DesktopClient

`DesktopClient.Name As String`

Enables client name consultation or modification in clients bar

`DesktopClient.GetID As Variant`

Enables determination of client identifier

`DesktopClient.Flags As Integer`

Enables client configuration consultation or modification (use not currently public)

`DesktopClient.Type As Variant`

Enables determination of client type (diagram, browser, other…)

`DesktopClient.ToolID As Variant`

Enables determination or modification of tool identifier. A specific client can be found using this property. The property can be updated using a field

`DesktopClient.Activate`

Enables client activation (bring to surface)

`DesktopClient.Component As Object`

Enables access to client implementation. In the case of a WebBrowser, this component is the browser itself, supporting the `IWebBrowser2` interface, which among other things enables definition of a URL for the browser…

10.2 Launching MEGA Tools from APIs

10.2.1 Interactive tools

Access to an element menu

`MegaItem.CommandManager As Object:`

See Accessing MEGA Object menus using APIs p. 108.

Exploring an element

VB Script `MegaItem.Explore :`

Java `MegaItem.invokeMethod("Explore");`

This command launches MEGA explorer on any MegaItem (MegaObject or MegaCollection).

When calling a MegaCollection, the root is a field (might be red if the collection does not correspond with a MetaAssociationEnd).

Access to the description of an object property pages

VB Script `MegaObject.PropertiesDialog(« Description ») As Object :`

Java `MegaObject.callFunction("propertiesdialog", "Description");`

See Accessing the description of an object Property Pages p. 104.

Opening an object property pages

VB Script
```
MegaObject.PropertiesDialog(optional option1 As String,optional option2 as
String,…) :
```

Java
```
MegaObject.invokeMethod("PropertiesDialog", String optional option1, String
optional option2,…);
```

The following options are optional and can be entered in any order:

DefaultSize=w,h: property window default size

Position=x,y: property window position in the page

CommentSize=h: help area hight size

ActivePage={clsid}: initially active page. You can obtain the page identifier from the object description mentioned above.

Open the macro editor

VB Script
```
MegaObject<Macro>.OpenMacro
```

Java
```
mgRoot.getObjectFromID("~vh1nrtiV95P0[DoDAF Exchange
Utilities]").invokeMethod("OpenMacro");
```

Interactive deletion

VB Script
```
MegaItem.QueryDelete(optional Options As String,optional impactList As String) :
```

Java
```
mgRoot.getObjectFromID("~vh1nrtiV95P0[DoDAF Exchange
Utilities]").invokeMethod("QueryDelete");
```

This function runs interactive deletion on the object or collection relating to the MegaItem.

If present, the *options* parameter can contain the following keywords:

```
SilentMode: the user interface is not activated; deletion is launched on all
objects, including propagated objects
NoCheckDeletable: deletion rights are not checked; all objects that can be
deleted physically are deleted.

In Mega 2005, a deleted objects backup system can be activated or deactivated

BackupHidden: hide backup view

BackupActive: backup activation

BackupInactive: backup deactivation
```

When deletion of an object can impact an important object (a diagram for example), an indicator is displayed in the interface. It is possible to define a list of objects for which we do not want to display impact; to do this, we must concatenate in the *impactList* parameter the absolute identifiers (in the form of fields) of these objects. In particular, this system allows that, at deletion in a diagram, impacts relating to this specific diagram are not included in the interface.

Interactive query

VB Script
```
MegaRoot.RequestQuery(optional metaClasses As Variant) As MegaCollection
```

Java
```
MegaCollection mgcolQuery = (MegaCollection)
mgRoot.invokeFunction("RequestQuery");
```

This function runs interactive query and returns the collection thus selected.

If the query is cancelled by the user, the result is *Nothing*.

The *MetaClasses* parameter enables definition of the MetaClasses to which the query should relate.

If it is absent, the query relates to all MetaClasses accessible to the user.

If specified with the name, identifier or field representing a MetaClass, query is restricted to these.

If we want to run a query related to several MetaClasses, we should specify as parameter a MegaCollection containing a list of these MetaClasses.

Interactive object selection

```
MegaCollection.SelectQuery(optional title As String,optional monoSel As Boolean)
As MegaCollection
```

Running the selection box enables selection of one or several objects of the start collection, and restores the collection of selected objects.

Title of the box is *title* if the argument is specified.

If *monoSel* is *True*, the box allows selection of only one object. Titles of these boxes should be specified.

The collection restored could be empty.

Running the object creation wizard

```
MegaCollection.InstanceCreator As Object :
```

See "*Courseware – MEGA wizard implementation*" Technical article.

Launching a generic wizard

```
MegaObject.CallFunction(«~AfLYxbu47b00[WizardRun]») As Object :
```

See "*Courseware – MEGA wizard implementation*" Technical article.

To be documented

MegaObject.Edit: opens object editor

MegaObject<Diagram>.Open: opens diagram

ProgressBarDlg: launches the progress bar

Simulate: launches the simulation tool

10.2.2 Batch Tools

Access to rules and regulations API

```
ApplyRegulation; ApplyRule; ApplyTest,RuleAppliableIs, RegulationActivate,
TestApply :
```

See Accessing rules and regulations using APIs p. 190.

Evaluating a condition on an object

```
MegaObject.ConditionEvaluate(condition) As Boolean
```

The condition conforms to the syntax of conditions used in rules, properties pages or commands

Creating duplicate of an object

```
MegaObject.CreateDuplicate(copyName As String,subObjectPrefix As String)
```

Creating first variant of an object

```
MetaObject.CreateVariant As MegaObject
```

Testing if object is filtered in the current session of MEGA

```
MegaObject.IsAvailable As Boolean
```

For a MetaAssociationEnd the test is carried out on the MetaAssociation.

Importing a command file

```
MegaRoot.MegaImport (importfilename as string, rejectfilename as string,
optionalparameters as string) as integer
```

Saving an element

```
MegaItem.SaveAs
```

See Import/Export section p. 127 and see also MEGA documentation regarding import/export options.

Exporting content of changeItem or changeItem collection

```
MegaItem.SaveAsCommand
```

Function not delivered. To obtain it, import the corresponding **AlignToBase** macro.

To create it directly; internal macro:

```
_objectfactory = "CompareMethodObjectCreate", _server = GBMF
```

Enables comparison of elements in two repositories and generation of alignment file.

```
MegaItem.CallFunction(« AlignToBase » , otherBase, updateFileName As
String,optional optionList As String,optional deleteFileName As String)
```

If MegaItem is a MegaRoot, we compare complete repository.

If MegaItem is a MegaObject, we compare this object (and its content). If a MegaCollection, we compare objects of the collection

otherBase: identifier or name of the repository to be compared

updateFileName: name of the file to be generated

deleteFileName: name of file containing deletion commands, if we want to handle these separately.

optionsList: list of options. Options included in a single character string, separated by commas, and in any order. These options are:

UpdateExtract: generates extraction update only, not that of complete repository

Reverse: generates reverse alignment

Ignore=item1 {;item2…} does not compare cited attributes. We can include as many Ignore= options as there are attributes; these are grouped, separated by semi-colons. Conventional attributes can be cited using keywords

Date;Creator;Authorization;Comment;Order. Others are included either by name, or their absolute identifier in the form of a field.

Used as a function, the return value indicates if the operation has been suitably executed.

Used as a method, returns an error if the operation has failed

To be documented:

MegaObject.Protection As MegaLock; accesses component managing object protection and lock

DetachDocument: detaches a MEGA Report(MS Word) object

MegaObject<Diagramm>.Drawing: accesses diagram drawing

ExecuteDescriptor: enables to execute a MEGA descriptor

GenerateCode: generates a Code descriptor

GenerateFmt: generates an HTML Formator

GetMetaClassPicture: retrieves the picture of a specific MetaClass

GetObjectPicture: retrieves the picture of a specific object

GetPropPicture: CDB

InheritedSelect: CDB

InitializeDocument: reinitialize a report(MS Word) content

NewDocument: creates a new Report (MS Word) from an object and a dedicated Report(MS Word) Template

WebSiteDescription: gives access to the APIs of the web site description

WebSiteTemplateDescription: gives access to the APIs of the web site Template description

10.3 Comparing and aligning (CompareTool API)

CompareTool is a MEGA object enabling management of object comparison between the current transaction and a target repository in VBScript. Comparison enables creation of an update file designed to align the target repository. This functionality is available with the "MEGA Supervisor" module.

For detailed information see "Customizing perimeter" Technical Article.

10.4 Exporting (ExportTool API)

ExportTool is a MEGA object enabling management of standard export of an object or collection of objects in VBScript or Java. This functionality is available with the "MEGA Supervisor" module.

For detailed information see "Customizing perimeter" Technical Article.

10.5 Launching an automatic macro while publishing

The system described below enables launch of automatic processing when dispatching a MEGA transaction.

Availability: MEGA 2005 SP 4 CP 7.0

MEGA 2007 CP 9 ?

MEGA 2009 SP 1 CP 1.0

In particular, this system can be used:

- to execute compliance updates before dispatching
- to run synchronizations with third-party tools when we are sure that the transaction will not be discarded.

In principle: we manage a list of identified 'Jobs' in a transaction

- by the macro to be called
- by a specific parameter.

This list of jobs can be accessed from a MegaRoot by the function:

VB Script
```
Set Jobs = MegaRoot.DispatchJobs
```

Java
```
MegaDispatchJobManager mgdjmJobMgr = new
MegaDispatchJobManager(mgRoot);
```

A job is inserted in this list by:

VB Script
```
Jobs.Insert Macro , Parameter
```

Java
```
mgdjmJobMgr.insert(macro, Parameter);
```

Where:

- **Macro** is the identifier of the macro to be executed for the job, or a character string containing the macro name or field
- **Parameter** is any character string serving as parameter for the Job

*The pair (**Macro**,**Parameter**) is unique in a transaction: if we insert the same pair twice, the second insertion is ignored.*

We can browse the list already recorded for this repository in this transaction and consult characteristics of each job

VB Script

```
dim Job
For i=1 to Jobs.count
Job=Jobs.item(i)
Print Jobs.item(i).GetID ' unique identifier assigned to the job at its creation
Print Jobs.item(i).MacroID ' identifier of the macro to be executed for the job
Print Jobs.item(i).Parameter ' job parameter
Next
```

Java

```
for (int j = 1; j <= (Integer) mgdjmJobMgr.basedObj.invokeFunction("count");
j++) {
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", ((MegaCOMObject)
mgdjmJobMgr.basedObj.invokeFunction("item", j)).invokeFunction("GetID"));
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", ((MegaCOMObject)
mgdjmJobMgr.basedObj.invokeFunction("item", j)).invokeFunction("MacroID"));
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", ((MegaCOMObject)
mgdjmJobMgr.basedObj.invokeFunction("item", j)).invokeFunction("Parameter"));
}
```

This list remains empty if you are not in transaction.

To remove a job from the list:

VB Script

```
Jobs.Remove Macro , Parameter
```

Java

```
mgdjmJobMgr.remove(macro, Parameter);
```

This function does not generate an error when the pair (Macro, Parameter) does not correspond to a job in the list.

Note: In Mega 2009, we can insert jobs concerning only the system repository; this list is accessible by MegaRoot.GetSystemRoot.DispatchJobs

At dispatch:

The macro corresponding to each job is instanced. If the same macro is used for several jobs, as many instances as there are jobs will be created.

It is therefore possible to use globals in these macros. Just before dispatch: we call the following Method in the Macro. This method is necessarily present in the Macro

```
Sub RunJobBeforeDispatch(Root As MegaRoot,Parameter As String)
' work with dispatch data.
' it is not possible at this stage to execute updates that will be taken into
account in the transaction.
End Sub
```

Dispatch then takes place. If dispatch is successful, call the following method:

```
Sub RunJobAfterDispatch(Root As MegaRoot,Parameter As String)
' good point to execute Commit on third-party DBMS, if required.
```

```
' if a global has been defined in RunJobBeforeDispatch, it can be used here.
' MEGA updates however should not be executed in this function.
End Sub
```

If dispatch is successful, call the following method:

```
Sub RunJobOnFailedDispatch(Root As MegaRoot,Parameter As String)
' good point to execute Commit on third-party DBMS, if required.
' if a global has been defined in RunJobBeforeDispatch, it can be used here.
' MEGA update execution is not recommended in this function, which has not been tested (to do this, a dispatch must fail...)
End Sub
```

The following call has been added:

```
Sub RunJobEnd(mgobjRoot As MegaRoot,strParameter As String)
End Sub
```

At dispatch, the user interface is locked to prevent any user modifications.

Implementation of this sub is called after user interface unlocking. This allows jobs to propose a user interface, which would otherwise be locked.

Note: RunJobEnd is called whether dispatch is successful or not. To determine if dispatch has been successful, use a global update RunJobAfterDispatch or RunJobOnFailedDispatch, depending on the case.

The following function has been added:

```
Function RunTestJobBeforeDispatch(mgobjRoot As MegaRoot, strParameter As String) As String
End Function
```

This function is called before dispatch (after click on Dispatch button) in the repository but unlike the RunJobBeforeDispatch procedure, it prevents dispatch if necessary.

If the function returns "" (empty string) then dispatch will be started; if not, the user returns to the transaction after acknowledgement (responsibility of macro to display explanatory message).

10.6 Invoking an object creation wizard using APIs

Objective: Enables creation of an object using its interactive creation wizard.

This function uses the InstanceCreator method, available on a MegaCollection. This method returns (if appropriate) a MegaInstanceCreator component.

```
Set myCreator = myRoot.GetCollection("~QrUiM9B5iCN0[Org-Unit]").InstanceCreator
[java : MegaInstanceCreator myCreator = new MegaInstanceCreator(myCollection) ]
```

This component enables launch of the object creation wizard offering the possibility of modifying its behavior, in particular:

- By redefining wizard launch mode by means of the mode property – In practice, 3 modes are possible
  - mode = 2: non-interactive launch – no window opens, only specific processing of creation is launched, possibly dependent on supplied parameters. Creation by this mode fails when a required parameter is missing, or when the specific object creation wizard does not implement this function.

- mode = 1: standard creation mode with entry of name and all parameters.
- mode = 0: "inplace" mode. this mode does not allow name entry and is adapted to creation from a listview, in which the name will be subsequently edited. Depending on the case, no dialog box is opened. If however a required parameter is missing, the interactive wizard may nevertheless be launched.

- By specifying creation context parameters (attributes or cookies).
- (Mega 2009) By inserting additional processing and pages to the wizard.

The interface of this component is an extension of the MegaWizardContext interface (in particular the Mode property mentioned above is a property of this interface). Additions are:

```
create As Variant
```

This function launches the wizard. It returns the identifier of the object created (or possibly reused) and 0 if the wizard has been discarded.

It is not currently possible to continue using the component after call on the Create function.

```
getRoot As MegaRoot
classID As Variant
```

Obtains wizard target MetaClass.

In Mega 2009, the following methods have been added:

VB Script
```
addTriger(TriggerId As Variant,Optional Options As String)
```

Java
```
public void addTriger(final Object trigger, final String position)
```

This method enables insertion of a trigger, of which we specify the identifier here. This trigger enables modification of wizard behavior, and in particular the addition of an initialization or processing code. By default, this trigger is called last (that is after the triggers defined for the wizard). However, with the "OnHead" option, it is possible to arrange that this trigger be called first.

VB Script
```
addPage(PageId As Variant,Optional Options As String)
```

Java
```
public void addPage(final Object page, final String position)
```

this method enables insertion of an additional page in the wizard. PageId is the identifier of the MetaPropertyPage that we want to insert. The option enables specification of the position of the page, and it can take the following values:

`Preliminary:` the page is presented in first position. It does not propose entry of name.

`Preparatory:` the page is presented after the preliminary pages, but before creation of the object.

`Standard:` This is the default; the page is presented after the preliminary and preparatory pages, but before creation of the object. It displays the name of the object if necessary.

`AfterCreation:` the page is presented after creation of the object.

`Conclusive:` the page is presented in last position.

It is possible to add several pages. If they are of the same type, they will be proposed in the order in which they are added.

If the option is not specified (or equals **Standard**), the page is proposed before object creation.

Example: creating a MetaAssociation

This is a case where InstanceCreator is essential: Creating a Metaassociation in API is impossible, since the two MetaClasses Source and Target must be known before Creation. It is however possible to create a MetaAssociation in a MetaModel diagram, using a link between two MetaClasses.

To create a MetaAssociation in API, we will simulate this creation mode by specifying source and target MetaClasses.

VB Script

```
Set myCreator = myRoot.GetCollection(« MetaAssociation »).InstanceCreator
myCreator.mode = 2
myCreator.SourceID = <identifiant de la métaclasse source>
myCreator.TargetID = < identifiant de la métaclasse cible>
myCreator.Name = « <MetaAssociationName> »
myAssocID = myCreator.Create
```

Java

```
MegaInstanceCreator myCreator = new
MegaInstanceCreator(myRoot.getCollection("MetaAssociation");
myCreator.mode(2);
myCreator.sourceID(<source metaclasse identifier >);
myCreator.targetID(<source metaclasse identifier>);
myCreator.name(<association name>);
Object myAssocID = myCreator.create() ;
```

10.7 Checking a script execution

From the execution context of a Macro, we can obtain a system enabling script execution check. This system makes available to the script a status area displayed in real time to the user, a gauge and the possibility of cancelling processing by means of a button.

This system can be implemented by the application launching the script; otherwise a specific window is displayed.

You get the execution context of a macro from the Root

VB Script

```
set mControl = getRoot.ContextObject("#Window")
mControl.Create
mControl.Text = "Hello"
mControl.Status = "This is a test"
mControl.SetRange 1, 20000
for i = 1 to 20000
mControl.Status = "We are in " & i
mControl.SetGauge i
if mControl.IsAborted then i = 20000
next
if mControl.IsAborted then print "The processing as been aborted"
```

Java

```
MegaProgressControl mgmpcControl = new MegaProgressControl(mgRoot);
```

```
 mgmpcControl.create();
 mgmpcControl.text("Hello");
 mgmpcControl.status("This is a test");
mgmpcControl.setRange(1, 20000);
for (int j = 1; j <= 20000; j++) {
mgmpcControl.status("On en est à " + j);
 mgmpcControl.setGauge(j);
 if (mgmpcControl.isAborted()) {
 j = 20000;
  }
  }
  if (mgmpcControl.isAborted()) {
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", "The processing has been
aborted");
 }
mgmpcControl.destroy();
```

Methods available with this component (MegaProgressControl) are the following:

`MegaProgressControl.Text:` title content

`MegaProgressControl.Status:` status content

`MegaProgressControl.Create:` window activation. If the calling context does not manage the StatusWindow, it is created at this occasion

`MegaProgressControl.Destroy:` window closing (if created by the script)

`MegaProgressControl.SetRange min, max:` scroll bar range definition (0 -> 32767)

`MegaProgressControl.SetGauge val:` scroll bar positioning

`MegaProgressControl.IncrementRange inc:` gauje incrementing

`MegaProgressControl.NextIcon:` animate the icon (when needed)

`MegaProgressControl.IsAborted:` indicates that the user wants to abort the processing (he can click Cancel button when present)

`MegaProgressControl.Abort:` activates 'IsAborted' flag

10.8 Setting up a progress bar in macro execution

This section details how to implement a progress bar (called also gauge) to show the execution progression of a script. The progress bar can be put inside an existing window or can create a new window if required. It is useful to give a visual feedback to the user while MEGA is executing a time consuming script.

When executing a time consuming script you can setup a Progress Bar to let the user know that something is happening, and also show an estimation of the progress if you know how much steps you will need to complete.

Mega has the **MegaProgressControl** object to implement progress bars.

If you are inside a wizard then the progress bar will be put in the bottom left corner of the wizard.
If you are not in a wizard then a new popup window will appears, just with the progress bar and a status line.

To create this use the following code (`root` is a MegaRoot):

VB Script

```
set pBar = root.ContextObject("#Window")
pBar.create
pBar.text = "I'm a progress bar"
pBar.setRange 1, 1000
for i=1 to 1000
pBar.status = "We are at " & i
pBar.setGauge i
next
pBar.destroy
```

Java

```
MegaProgressControl pBar = new MegaProgressControl(root);
pBar.create();
pBar.text("I'm a progress bar");
pBar.setRange(0, 1000);
for(int i=0; i<1000; i++) {
pBar.status("We are at " + i);
pBar.setGauge(i);
}
pBar.destroy();
```

If you want to know if the user clicked on the "Abort" button then check the value pBar.isAborted() (method in Java, value in VB), if the result is true then the user clicked on the abort button.

The list of available methods is (Java version, but VB is pretty the same, just use all methods as properties in VB):

- MegaProgressControl.text(String title) Add a title to the progress bar
- MegaProgressControl.status(String status) Add a status to the progress bar, useful to show what the script is doing
- MegaProgressControl.create() The progress bar is created, in this call if a container window exists (like a Wizard) then the progress bar is attached to it, if not a new popup window is created
- MegaProgressControl.destroy() Destruction of the object. Is mandatory to call it or you will get an error when your script will terminate

- MegaProgressControl.setRange(int min, int max) Range of values for the progress bar, the minimum min is 0, the maximum max is 32768
- MegaProgressControl.setGauge(int val) Set the progress bar at the specified value
- MegaProgressControl.incrementRange(int val) Increment the progress bar of the passed value
- MegaProgressControl.isAborted() Returns a boolean to understand if the user clicked on the Abort button.
- MegaProgressControl.abort() Has to be called to activate the Abort button

Let the user know what's happening!

10.9 Customizing an extraction using APIs

Availability: Mega 2007

This document supplements documentation on use of compound operators in APIs.

As a reminder, 'compound' operators do not have Macros and are designed to configure behavior of links related to particular concerns; each MetaAssociationEnd has an associated behavior, the most important of these being:

`'A' (Abort):` the MetaAssociationEnd is ignored.

`'L' (Link):` the MetaAssociationEnd is taken into account, but we do not continue search on the associated object, which can be extracted from the collection

`'S' (Standard):` the MetaAssociationEnd is taken into account, but we do not continue search on the associated object, which must nevertheless be included in the collection

`'D' (Deep):` the MetaAssociationEnd is taken into account and search continues on the associated object

Extraction by operator therefore consists of building a collection containing all objects associated with a start object (or start collection) by scanning associations according to their behavior.

Elements of the resultant collection have a 'Behavior' pseudo attribute containing the 'maximum' value that caused extraction. (L < S < D)

When extracted objects have been listed, this collection can be completed with the exhaustive list of existing associations between each of these extracted objects.

Call to such extraction is by:

```
Set myCollection = MegaItem.OperatorName(Optional options as String,Optional
filter As Object)
```

or

```
Set myCollection = MegaItem.CallFunction(operator As String, Optional options as
String,Optional filter As Object)
```

Examples:

VB Script

```
Set myCol = myObject.Extract
Set myCol = myObject.Extract
Set myCol = myObject.CallFunction("Extract")
```

Java
```
MegaCollection mgcolCollection = (MegaCollection)
mgobjMegaObject.invokeFunction("Extract");
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", mgcolCollection.size());
```

Note that extracted objects with an 'L' behavior are included in the collection.

10.9.1 Options

Options enable configuration of this extraction in greater or lesser detail. The character string can contain the following exclusive keywords:

NOMETACLASS: Only the extracted Associations are inserted in the collection

NOMETAASSOCIATION: Only the extracted Occurrences are inserted in the collection

DIRECTONLY

DIRECTASSOCIATIONS

DIRECTOBJECTS: 'Direct' extractions transform 'Deep'behaviors to 'Standard'

Extraction is therefore at only one level.

With DIRECTOBJECTS, associations are not listed;

With DIRECTASSOCIATIONS, objects are not listed

ALLDIRECTASSOCIATIONS: The operator is no longer taken into account, all behaviors are taken to 'Standard'

Only the associations are listed in the collection

10.9.2 Confidential object filtering

Confidential objects are not usually included in collections and are not scanned.

To change this behavior add the @TAKECONFIDENTIAL keyword to the options.

Example:

VB Script
```
set myCol = myObj.Extract("DIRECTOBJECTS,@TAKECONFIDENTIAL")
```

Java
```
MegaCollection mgcolCollection = (MegaCollection)
mgobjMegaObject.invokeFunction("Extract", "DIRECTOBJECTS,@TAKECONFIDENTIAL");
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", mgcolCollection.size());
```

10.9.3 Advanced filtering using a component

To execute a more precise filtering use an advanced filtering component.

This component enables definition of customized filtering, either at metamodel level, or at the level of occurrences themselves.

The filtering component is passed as parameter; it can be the current macro using keyword MySelf.

Example:

```
Set myCol = myObj.Extract(MySelf)
```

This component can implement the following functions:

```
Sub OnAssociationFilter(Context As MegaExtractContext,AssocEndID,Behavior)
```

This Method is then called once for each browsed MetaAssociationEnd and enables redefinition of its behavior.

```
Sub OnChildCollectionFilter(Context As MegaExtractContext,AssocEndID,Behavior)
```

This Method is then called for each extracted object and allows us to redefine specifically for this object the behavior of a browsed MetaAssociationEnd.

```
Sub OnChildObjectFilter(Context As MegaExtractContext,AssocEndID,Behavior)
```

This Method enables specific filtering of objects browsed by a given MetaAssociationEnd from a given object.

Extraction parameters are obtained from the MegaExtractContext context component.

`MegaContext.RootItem As MegaItem`: extraction start element (object or collection)

`MegaContext.CurrentSource As MegaObject`: element in course of extraction. Valid only for `OnChildCollectionFilter` and `OnChildObjectFilter` methods

`MegaContext.CurrentTarget As MegaObject`: current browsed element. Valid only for `OnChildObjectFilter` method

10.10 Using Administration APIs with callback objects

Administration APIs have the specificity to work in a Windows-based process separate from MEGA.

Any code execution (especially when complex) that can be implemented in a macro (and so which is not dependent on administration API specificities) must be implemented in a macro

The reasons are the following:

- **Performance**
  The macro will be executed in MEGA and thus does not need the interoperability to execute the functions. This can lead to a dramatic performance boost as an interoperability call cost is 10 to 100 times higher than an in-process call. This is not dramatic for standard administration functions (for example an "import" function interoperability of 1/100 sec instead of 1/10000 sec is not significant comparing to the import itself that lasts 15 min).

  On the other hand, the interoperability cost of a « for each» in GetCollection is proportionally higher and can even be higher than the cost of the function itself: if the function lasts 1/10000 sec, the interoperability will multiply by 100 the execution time.

- **Usual DCOM interoperability limitations**
  You cannot use a component, which has not been specifically created for this purpose, in another process. Especially, MEGA or Java components created in a specific process can only be used in this process.

10.10.1 Use case example: Customizing an extraction using APIs

See section 10.9 Customizing an extraction using APIs p. 142.

In principal, you can perform an extraction by calling a component that enables to dynamically filter MEGA objects:

Set myCollection = MegaItem.CallFunction(operator As String, Optional options as String,Optional filter As Object)

For example:

```
Class MyFilter
  Sub OnAssociationFilter(Context,AssocEndID,Behavior)
    ' recalculation code of the MetaAssociationEnd behavior …
  End Sub
End Class

Set myComponent = new MyFilter
Set myCollection = MegaItem.CallFunction("Extract","",myComponent)
```

This cannot be applied in a multiprocess context, thus in a .VBS outside MEGA. This is also true for a Java executable.

The CallFunction will be executed in MEGA process, but the MEGA process is not able to call myComponent in callback, as this component has not been created to be performed in the MEGA process, but to be performed in the VBS process. You would then get a message like « this component has been marshalled for another thread ».

➔ **To make all this work, you need to use a MEGA macro:**

```
Sub MyExecution(MegaItem)
Set myComponent = MegaItem.CurrentEnvironment.GetMacro("<filtering macro>"
Set myCollection = MegaItem.CallFunction("Extract","",myComponent)
End Sub
```

and call it unitarily in the external script:

```
MegaItem.CurrentEnvironment.GetMacro(<TheMacro>).MyExecution(MegaItem)
```

In that case, calls are one way VBS -> MEGA and there is no issue.

10.11 Implementing an Update Tool in script

Availability: 2007 SP1 and 2009

An UpdateTool:

- is a macro linked through the UpdateTool link to a MetaAttribute, a MetaAssociationEnd or a TaggedValue.
- Enables to modify this concept input behavior in the property pages and in-place areas (outside diagrams)

The Script implementation does not allow defining a data entry window; it only allows behavior modification of an existing control.

To be recognized as such, an UpdateTool script must implement the following function:

VB Script
```
Function AttCtl_GetDefaultKind() As String
```

Java
```
public String attCtl_GetKind(final MegaUpdateToolContext
objMegaUpdateToolContext)
```

This function enables to define the window type to use for input and must feed back a control type.

It is either an internal numerical value, or a character string with the following format:

```
<ControlName>{:<option>}{,<option>}
```

<ControlName>: see property pages documentation:

MegaEditCheck

ComboBox

DropDownList

ComboBitmaps

DatePicker

CheckBox

3StateCheckBox

ComboBoxMenu

DropDownListMenu

ComboLinks

EditButton

StaticMenu

EditMenu

Text

Static

Edit

options:

ReadOnly: read-only control

Numerical: the Edit area displays a number (right-justified)

WithDefault: the control displays the default value button

Mandatory: mandatory value

NoEdit: the Text area (of the combo box, of the Edit menu, of the Edit button) is in read-only

ResetOnRefresh: specific usage

ValidateInput: specific usage

External: display of the attribute external value

An update tool can simply redefine the control to be used. In that case the standard updatetool behavior is called to process the input.

If the control type to be used depends on the occurrence, you can redefine it specifically for this occurrence by implementing:

VB Script
```
Function AttCtl_GetKind(oContext As MegaUpdateToolContext) As String
```

Java
```
public String attCtl_GetKind(final MegaUpdateToolContext
objMegaUpdateToolContext)
```

This function returns the type of control to be used specifically for the current occurrence. The return value is similar to the AttCtl_GetDefaultKind one. You can get the current occurrence with the context provided in argument.

This function can also be used to parameterize the standard updateTool behavior.

The MegaUpdateToolContext component includes the following functions:

VB Script
```
Function MegaUpdateToolContext.MegaObject As MegaObject
```

Java
```
public MegaObject megaObject()
```

Occurrence to be modified

VB Script `Function MegaUpdateToolContext.AttributeID As Variant`

Java `public Object attributeID()`

Absolute identifier of the attribute (resp. TaggedValue, MetaAssociationEnd) managed by the updateTool.

VB Script `Sub MegaUpdateToolContext.Invalidate As Variant`

Java

Notifies that the element has been modified. This particularly allows to ungrey the Apply button of the property pages.

VB Script `Function MegaUpdateToolContext.GetRoot As MegaRoot`

Java `public MegaRoot getRoot()`

VB Script `Function MegaUpdateToolContext.AttributeControl As MegaObject`

Java

Sends the component that manages the updateTool piloted control. This component includes the interface of previously mentioned attributecontrols, we can however note availability of the property.

VB Script `AttributeControl.Page As MegaPropertyPageStandard`

Java `public MegaObject attributeControl()`

In wizard case, you can get the wizard context by Page.Cookie …

When the update tool implements a generic MetaAssociationEnd or an attribute or a taggedvalue of objet type, use the following property:

VB Script `MegaUpdateToolContext.ValueTypeID`

Java `public Object valueTypeID()`

To consult or update the MetaClass type of the target object. For example:

```
oContext.ValueTypeID = "Broker"
```

enables standard commands of the object (find/list) to do this on the "Broker" MetaClass.

When the control displays an editing area, you can catch the initialization of this area by implementing:

VB Script

```
Function AttCtl_SetText(oContext As MegaUpdateToolContext,sInitialValue as
String) As Boolean
```

Java

```
public void editText(final String setValue)
```

It is then possible to modify the displayed value by modifying the *sInitialValue parameter and sending back True value.*

To catch a command, for example a click on the button when the control has one, implement:

VB Script
```
Function AttCtl_OnCommand(oContext As MegaUpdateToolContext,Item As
Integer,Notification As Integer) As Boolean
```

Java

In the EditMenu case, you can add manual commands by implementing a MetaCommand in the update Tool. For this MetaCommand to be called, implement:

VB Script
```
Function AttCtl_ImplementsMetaCommand(oContext As MegaUpdateToolContext) As
Integer
```

Java
```
public String attCtl_ImplementsMetaCommand(final MegaUpdateToolContext
objMegaUpdateToolContext)
```

The returned value is 0 if you do not want standard menu commands to be displayed.

Otherwise it includes the capability to be used for this standard commands.

To access to the updatetool context in the CommandAccessor functions, you need to pass through the global MegaMacroData

```
Sub CmdInvoke(obj,num)
Dim AttCtlContext As MegaUpdateToolContext
Set AttCtlContext = MegaMacroData.GetBag.AttCtlContext
' …
End Sub
```

In that context, you can particularly exploit the following MegaUpdateToolContext functions:

VB Script
```
MegaUpdateToolContext.ValueTypeID
```

Java
```
public Object valueTypeID()
```

Current object IdAbs in Edit Menu

VB Script
```
MegaUpdateToolContext.EditText ID
```

Java
```
public String editText()
```

Value to be displayed in the area

Simple example (updatetool to link to an object type taggedvalue)

VB Script
```
'MegaContext(Fields,Types)

'Uses(Components)

Option Explicit
```

```
Function AttCtl_GetDefaultKind()
AttCtl_GetDefaultKind = "ComboBoxMenu:ValidateInput"
End Function
Function AttCtl_ImplementsMetaCommand(AttCtlContext As MegaUpdateToolContext)
AttCtlContext.ValueTypeID = "~BEy8SnY(yKk0[City Planning Area])"
AttCtl_ImplementsMetaCommand = 7
End Function
Sub CmdCount(obj,count)
count = 3
End Sub
Sub CmdInit(obj,num,name,category)
name = "Command " & num
category = 4
End Sub
Sub CmdInvoke(obj,num)
Dim AttCtlContext as MegaUpdateToolContext
Set AttCtlContext = MegaMacroData.GetBag.AttCtlContext
Dim oResult
Set oResult = AttCtlContext.MegaObject.GetRoot.GetCollection("~QrUiM9B5iCN0[Org-
Unit]").SelectQuery("Invoke Command #" & num & " on Attribute " &
AttCtlContext.AttributeControl.Page.GetID,True)
if oResult.Count = 1 Then
AttCtlContext.ValueID = oResult.Item(1).GetID
AttCtlContext.EditText = oResult.Item(1).ShortName
end if
End Sub
```

Java

```
public class UpdateToolExample extends MegaMacro {

  public String attCtl_GetDefaultKind() {
    return "ComboBoxMenu:ValidateInput";
  }

  public String attCtl_GetKind(final MegaUpdateToolContext
objMegaUpdateToolContext) {
    return "ComboBoxMenu:ValidateInput";
```

```
  }


  public String attCtl_ImplementsMetaCommand(final MegaUpdateToolContext
objMegaUpdateToolContext) {
    objMegaUpdateToolContext.valueTypeID("~BEy8SnY(yKk0[City Planning Area])");
    return "7";
  }


  public void CmdCount(final MegaObject mgobjValidationCandidateObject, final
Integer[] intCmdCount) {
    intCmdCount[0] = 3;
  }


  public void CmdInit(final MegaObject mgobjValidationCandidateObject, final
Integer iCommandNumber, final StringBuffer strNameReturned, final Integer[]
intCategoryReturned)    {
    strNameReturned.append("Command " + iCommandNumber);
    intCategoryReturned[0] = 4;
  }


  public void CmdInvoke(final MegaObject mgobjValidationCandidateObject, final
Integer iCommandNumber) throws MegaException {
    MegaPropertyBag mpbBag = this.getBag();
    ComObjectProxy copAttCtlContext = (ComObjectProxy)
mpbBag.basedObj.invokeFunction("AttCtlContext");
    MegaUpdateToolContext mgutcContext = new
MegaUpdateToolContext(copAttCtlContext);
    MegaCollection mgcolResult;
    String strText = "Invoke Command #" + iCommandNumber;
    MegaObjectProxy mgobjAttCtrl = (MegaObjectProxy)
mgutcContext.attributeControl();
    ComObjectProxy comopPage = (ComObjectProxy)
mgobjAttCtrl.invokeFunction("Page");
    String strID = (String) comopPage.invokeFunction("getID");
    strText = strText + " on Attribute " + strID;
    mgcolResult = (MegaCollection)
mgobjValidationCandidateObject.getRoot().getCollection("~QrUiM9B5iCN0[Org-
Unit]").invokeFunction("SelectQuery", strText, true);
    if (mgcolResult.size() == 1) {
      mgutcContext.valueID(mgcolResult.get(1).getID());
      mgutcContext.editText(mgcolResult.get(1).getProp("Short Name"));
    }
  }
}
```

When the specified control displays a combo box, you can feed it by implementing the following function:

VB Script
```
Function AttCtl_FillCombo(oContext As MegaUpdateToolContext,oFillCollection as MegaCollection,sInitialValue as String) As Integer
```

Java
```
public int attCtl_FillCombo(final MegaUpdateToolContext objMegaUpdateToolContext, final MegaCollection mgcolFillCollection, final StringBuffer strInitialValue)
```

oFillCollection can be used to supply the collection of objects to be displayed in the list. This collection is also available by

```
Function MegaUpdateToolContext.ComboListCollection
```

This collection is a collection of Values (see GetTypeObject.Properties.Item(x).Values)

So that elements will effectively be integrated in the list:

- either these should be effective occurrences of Value (obtained from a description). It is only in this case that we can display bitmaps (NB: There are no bitmaps in ComboEditMenus)
- or these should be MegaObjects created explicitly in the collection. These objects are virtual. To be taken into account, the attributes **InternalName** and **GUIName** must be specified. InternalName will be used for update (except for an object list) and GUIName for display
- for attributes of object type or legattributes, we must supply the idabs value of the corresponding object. This should be done when creating the value (see example below).

Possible return values are:

- 0: call default processing
- 1: display list from collection - case of simple tabular attribute
- 2: display list from collection, taking account of absolute identifier of the value - applicable particularly to attributes of object type and to legattributes
- -1: as 1, if we want to display (Default) rather than (Empty) to indicate empty value in the combo
- -2: as 2, if we want to display (Default) rather than (Empty) to indicate empty value in the combo

The following example enables management of an object type attribute or a legattribute of which target is compatible with the 'Org-Unit' MetaClass. The list is supplied with all org-units in the repository.

VB Script
```
Function AttCtl_FillCombo(oContext As MegaUpdateToolContext,oFillCollection As MegaCollection) As Integer

Dim oOrg-Unit

for each oOrg-Unit in oContext.MegaObject.GetRoot.GetCollection("~QrUiM9B5iCN0[Org-Unit]")

Dim oAdded

Set oAdded = oFillCollection.Create(oOrg-Unit.GetID) ' the Org-Unit Absolute identifier is set to the created value.
```

```
oAdded.GUIName = oOrg-Unit.ShortName
oAdded.InternalName = oOrg-Unit.MegaField() ' Be careful, the internal value
must not exceed 20 characters. In that case, you can add a simple counter as in
an object case the internal value is not used
Next
AttCtl_FillCombo = 2
End Function
```

Java

// CURRENTLY NOT AVAILABLE - TESTS in progress …

To catch the control update, implement the following function :

VB Script

```
Function AttCtl_Update(oContext As MegaUpdateToolContext,iStatus As
Integer,sErrorMessage As String) As Boolean
```

Java

This function sends back False value, the update defaut code is not called.

10.12 Managing MEGA undo/redo actions from a Script

Availability: from Mega2009 CP1 & SP1 version

The following Macro is available:

`CreateUndoCollection`

This Macro Enables the aggregation of several update commands into a single line in the Undo List. It is possible to explicitly undo the registered commands. This can be useful if you want to manage a 'Cancel' button after a complex update operation.

VB Script
```
It provides a MegaUndoRedoManager component
Set UndoCollection = Root.CallFunction("~oeTN2v5z8z10[CreateUndoCollection]")
```

Java
```
MegaUndoCollection class
```

This component implements the followings functions:

VB Script
```
MegaUndoRedoManager.Start "<Command Name>"
```

Java
```
public void start(final String name)
```

Starts the command aggregation. The given name will be shown in the undo list at the end of the collect.

VB Script
```
MegaUndoRedoManager.Active As Boolean
```

Java
```
public boolean isActive()
```

Indicates if the UndoCollection is started or not. This is the default function.

VB Script
```
MegaUndoRedoManager.Stop
```

Java
```
public void stop()
```

Stops the command aggregation. After this command the UndoCollection Name appears in the undo list. This method is automatically called on the component destruction by default.

VB Script
```
MegaUndoRedoManager.Abort
```

Java
```
public void abort()
```

Stops the command aggregation and undo all the commands. After this command the UndoCollection Name appears in the redo list. Stop and Abort methods are exclusive.

10.13 Converting VB Script APIs into Java

You need to know how to convert a VB Script API into java when you are unable to find a correspondence between a Script API and the Java API library. Although most of the usual MEGA specifics API components have been wrapped into Java Classes (see the com.mega.modeling.api.util package) the less used components have not been involved.

However it is possible to use such a component in Java.

The API anonymous components (typed by Object in VB) are accessible in Java through the MegaCOMObject class. In all the cases, you can replace the following VB Code by the following Java Code:

VB Script
```
Dim myObject
Set myObject = XXX
```

Java
```
MegaCOMObject myObject;
myObject = (MegaCOMObject)XXX;
```

Let's now invoke methods and functions on those components.

- The Java language is an early-binded language and all the methods and functions that are called on java classes must be declared explicitly.
- The script components are late-binded. In such a component you can invoke a method that is not declared on an interface.

This is why the MegaCOMObject Java class implements the following functions:

```
invokeFunction(String, Object...)
invokeMethod(String, Object...)
invokePropertyGet(String, Object...)
invokePropertyPut(String, Object, Object...)
```

With those functions you can invoke the component functions in all the cases.

- invoke a propertyGet or a function returning a value (for example a string or an object)

VB Script
```
Dim myString
myString = myObject.PropName
Dim myObj2
Set myObj2 = myObject.FunctionName(Parameter)
```

Java
```
String myString;
myString = (String)myObject.invokePropertyGet("PropName");
MegaCOMObject myObj2;
myObj2 = (MegaCOMObject)myObject.invokeFunction("FunctionName", Parameter);
```

- invoke a method (with no return value)

VB Script
```
myObject.MethodName Param
```

Java
```
myObject.invokeMethod("MethodName", Param);
```

- set a property

VB Script
```
myObject.PropName = "Value"
```

Java
```
myObject.invokePropertyPut("PropName", "Value");
```

All the classes of the com.mega.modeling.api.util package are based on the same principle:

- they involve a MegaCOMObject as a member
- they implement a constructor that set this MegaCOMObject
- they implement explicit function that embed the MegaCOMObject invocation

For example a sample on the MegaPropertyBag class.

```
public int count() {

return ((Integer) this.basedObj.invokeFunction("Count")).intValue();

}
```

The MegaCOMObject can handle ALL the vb script calls, unless they have too many parameters (the maximum is 6).

If a Java function seems not to behave as expected, note that all the MEGA API Java classes correspond directly or indirectly to a MegaCOMObject; then you can use the native call in all the cases.

10.14 Calling a URL construction function using APIs in HOPEX

To call the the URL construction function, you need a MegaRoot. The prototype is as follows:

```
Function DesktopURLBuild( sParameterization as string, sEnvironment as string,
sDatabase as string, sApplication as string, sDesktop as string,sRoleAndProfile)
as string
```

Only the first parameter is mandatory.

The following parameters are optional, but if one of them is specified, all of them must be present.

`sParameterization` is as follows:

`Tool:` tool idabs

`Param:` tool parameter idabs

`ParamValue:` parameter value – object idabs if it is a MEGA object

`ParamValueMetaclass:` MetaClass idabs if the parameter value is a MEGA object

`Affinity:` Affinity idabs

`Tool:` tool idabs

`sEnvironment` equals empty if you want the end user to select an environment at connection, 0 if you want to connect to the same environment as the one from which the link is created, the environment path if you want to access to a specific environment

`sDatabase` equals empty if you want the end user to select a repository at connection, 0 if you want to connect to the same repository as the one from which the link is created, the repository name if you want to access to a specific repository

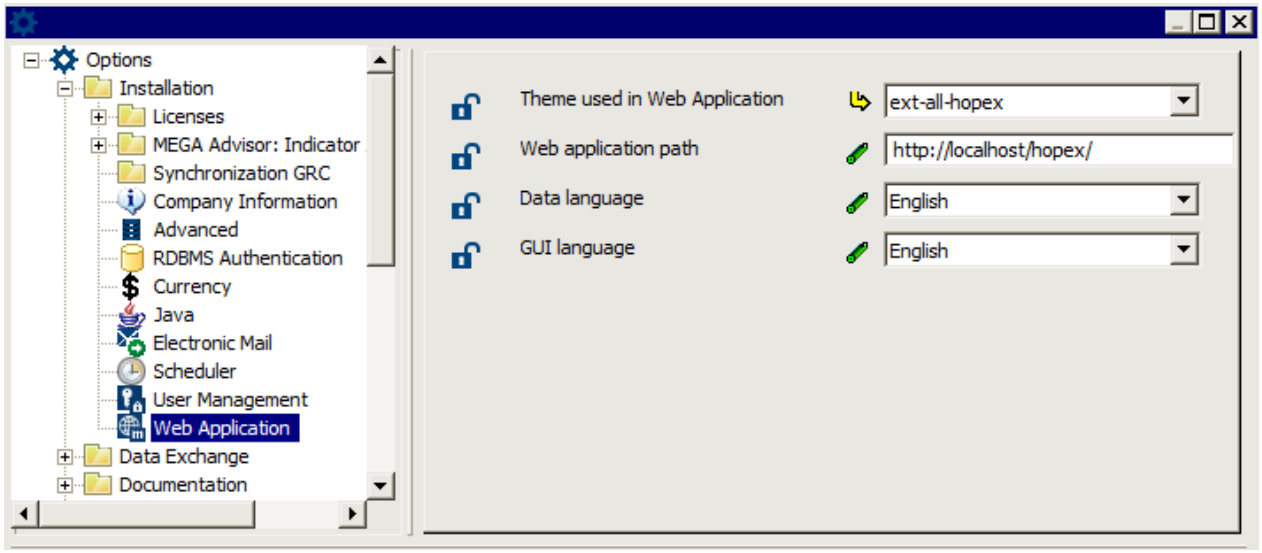
`sApplication` equals empty if you want the end user to select an application at connection, 0 if you want to connect to the same application as the one from which the link is created, application name if you want to access to a specific application

`sDesktop` equals empty if you want the end user to select a desktop at connection, 0 if you want to connect to the same desktop as the one from which the link is created, the desktop Hexaidabs if you want to access to a specific desktop

`sRoleAndProfile` equals empty if you want the end user to select a profile at connection, 0 if you want the user to use the same profile and Business role at connection from which the link is created, the Business Role absolute identifier followed by the profile absolute identifier or only the profile absolute identifier if you want to connect with a specific profile and Business Role.

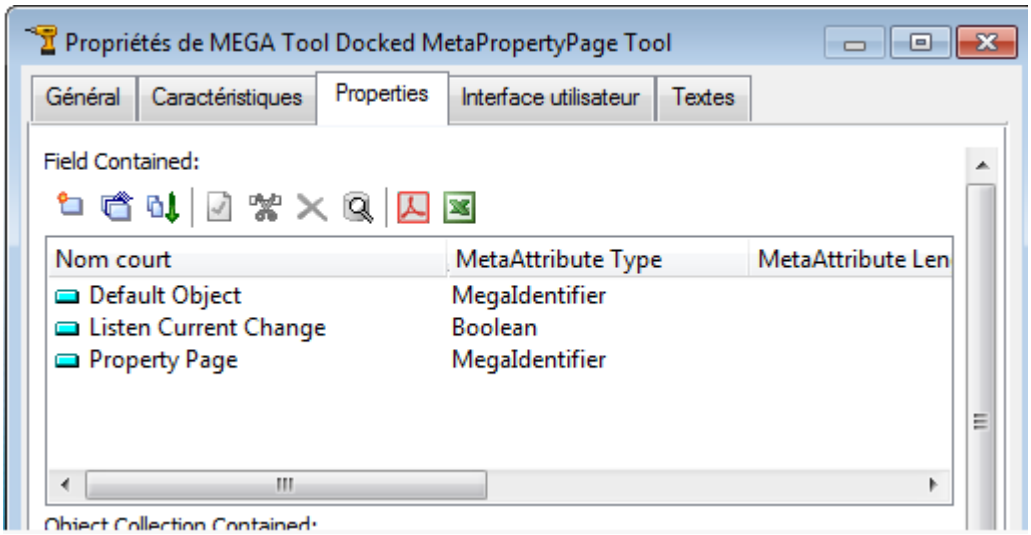
10.14.1 Code examples and results

Note: The **Web application path** has to be entered in **Options > Installation > Web Application**.



Example 1

➔ Display the property page tool and associate « Default object » parameter with the OIZYiqujArn0 object (Advisor Client) of type MrUiM9B5iyM0 (Application).



Code:

VB Script

```
print GetRoot.DesktopURLBuildEx("http://localhost/HOPEX/",
"Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclass=MrUiM9B5iyM0 ")
```

Java

```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgobjMegaObject.getRoot().invokePropertyGet("DesktopURLBuild",
"Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclass=MrUiM9B5iyM0 "));
```

Result:

http://localhost/HOPEX/default.aspx?userdata=Tool-Sj%283hVHpEXUN|VHRbsTkZFLvH-OIZYiqujArn0-MrUiM9B5i...

Exemple 2

1. Display the property page tool and associate « Default object » parameter with the OIZYiqujArn0 object (Advisor Client) of type MrUiM9B5iyM0 (Application).
2. Launch the diagram editor.
3. Launch macroLauncher tool with the macro that has the NiqwTtU5CPB0 identifier and has the « This is a test » additional parameter.

Code:

VB Script

```
print
GetRoot.DesktopURLBuildEx("http://localhost/HOPEX/","Tool=Sj(3hVHpEXUN,Param=VHR
bsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclass=MrUiM9B5iyM0,Tool=Fk(3zVHp
EXWN,Tool-
4i(35ZHpEHhN,Param=KXcGkNdEF1bR,ParamValue=NiqwTtU5CPB0,Param=6RQQCjsNFT)M,Param
Value=This is a test ")
```

Java

```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgobjMegaObject.getRoot().invokePropertyGet("DesktopURLBuild", "
Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclass
=MrUiM9B5iyM0,Tool=Fk(3zVHpEXWN,Tool-
4i(35ZHpEHhN,Param=KXcGkNdEF1bR,ParamValue=NiqwTtU5CPB0,Param=6RQQCjsNFT)M,Param
Value=This is a test "));
```

Result:

http://localhost/HOPEX/default.aspx?userdata=Tool-Sj%283hVHpEXUN|VHRbsTkZFLvH-OIZYiqujArn0-MrUiM9B5i...

Example 3

1. Display the same tools as in the previous example.
2. Reuse the same environment, the same repository, the same application and the same desktop.

Code:

VB Script

```
print GetRoot. DesktopURLBuildEx("http://localhost/HOPEX/",
"Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclas
s=MrUiM9B5iyM0,Tool=Fk(3zVHpEXWN,Tool-
4i(35ZHpEHhN,Param=KXcGkNdEF1bR,ParamValue=NiqwTtU5CPB0,Param=6RQQCjsNFT)M,Param
Value=This is a test ",0,0,0,0,0)
```

Java

```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgobjMegaObject.getRoot().invokePropertyGet("DesktopURLBuild", "
Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclass
=MrUiM9B5iyM0,Tool=Fk(3zVHpEXWN,Tool-
4i(35ZHpEHhN,Param=KXcGkNdEF1bR,ParamValue=NiqwTtU5CPB0,Param=6RQQCjsNFT)M,Param
Value=This is a test ", "0", "0", "0", "0", "0"));
```

Result:

http://localhost/HOPEX/default.aspx?Desktop=CCCCCC
CCCCCCCCCC&Db=Adventure&Env=C%3A%5cProgram%20Files...

Example 4

1. Display the same tools as in the previous examples.
2. Reuse the same environment and the same repository.
3. Change the application for «Enterprise Risk Management » application.
4. Use the "risk" desktop with "CF2FD0F14FC556D4" identifier.
5. Connect with « Other Participant in Audit » profile that has s28FH6qOG9qC identifier and the role « Other Participant in Audit » that has T28F56qOGLpC identifier.

Code:

VB Script

```
print GetRoot. DesktopURLBuildEx("http://localhost/HOPEX/",
"Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclas
s=MrUiM9B5iyM0,Tool=Fk(3zVHpEXWN,Tool-
4i(35ZHpEHhN,Param=KXcGkNdEF1bR,ParamValue=NiqwTtU5CPB0,Param=6RQQCjsNFT)M,Param
Value=This is a test ",0,0,"Enterprise Risk Managment", "CF2FD0F14FC556D4",
"T28F56qOGLpCs28FH6qOG9qC")
```

Java

```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgobjMegaObject.getRoot().invokePropertyGet("DesktopURLBuild", "
Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclass
=MrUiM9B5iyM0,Tool=Fk(3zVHpEXWN,Tool-
4i(35ZHpEHhN,Param=KXcGkNdEF1bR,ParamValue=NiqwTtU5CPB0,Param=6RQQCjsNFT)M,Param
Value=This is a test ", 0, 0, "Enterprise Risk Managment", "CF2FD0F14FC556D4",
"T28F56qOGLpCs28FH6qOG9qC"));
```

Result:

http://localhost/hopex/default.aspx/Enterprise Risk Managment?Desktop=z)opn3TnFHjL&from=RWP&Db=0GXuZ9UmHj4Q&Env=xG8xqoktHfgK&Profile=T28F56qOGLpCs28FH6qOG9qC&userdata=,%20Tool|VHRbsTkZFLvH-OIZYiqujArn0-MrUiM9B5iyM0Tool-Fk%283zVHpEXWN,Tool%2d4i%2835ZHpEHhN|KXcGkNdEF1bR-NiqwTtU5CPB0|6RQQCjsNFT%29M-This%20is%20a%20test%20

The new desktop is taken into account as well as the new application.

Example 5

1. Display the same tools as in the previous examples.
2. Specify "C:\Users\Public\Documents\MEGA 2012\Demonstration" environment and "DemoERM" repository.
3. Change the application for «Enterprise Risk Managment » application.
4. Use the desktop with "CF2FD0F14FC556D4" identifier.
5. Connect with « Other Participant in Audit » profile that has s28FH6qOG9qC identifier and the role « Other Participant in Audit » that has T28F56qOGLpC identifier.

Code:

VB Script

```
print GetRoot. DesktopURLBuildEx("http://localhost/HOPEX/",
"Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclas
s=MrUiM9B5iyM0,Tool=Fk(3zVHpEXWN,Tool-
4i(35ZHpEHhN,Param=KXcGkNdEF1bR,ParamValue=NiqwTtU5CPB0,Param=6RQQCjsNFT)M,Param
Value=This is a test ","C:\Users\Public\Documents\MEGA
2012\Demonstration","DemoERM","Enterprise Risk Managment", "CF2FD0F14FC556D4",
"T28F56qOGLpCs28FH6qOG9qC")
```

Java

```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgobjMegaObject.getRoot().invokePropertyGet("DesktopURLBuild", "
Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclass
=MrUiM9B5iyM0,Tool=Fk(3zVHpEXWN,Tool-
4i(35ZHpEHhN,Param=KXcGkNdEF1bR,ParamValue=NiqwTtU5CPB0,Param=6RQQCjsNFT)M,Param
Value=This is a test ", "C:\\Users\\Public\\Documents\\MEGA
2012\\Demonstration", "DemoERM", "Enterprise Risk Managment",
"CF2FD0F14FC556D4", "T28F56qOGLpCs28FH6qOG9qC"));
```

Result:

http://localhost/hopex/default.aspx/Enterprise Risk Managment?Desktop=z)opn3TnFHjL&from=RWP&Db=DemoERM&Env=C%3A%5cUsers%5cPublic%5cDocuments%5cMEGA%202012%5cDemonstration&Profile=T28F56qOGLpCs28FH6qOG9qC&userdata=,%20Tool|VHRbsTkZFLvH-OIZYiqujArn0-MrUiM9B5iyM0Tool-Fk%283zVHpEXWN,Tool%2d4i%2835ZHpEHhN|KXcGkNdEF1bR-NiqwTtU5CPB0|6RQQCjsNFT%29M-This%20is%20a%20test%20

All of the changes are taken into account in the url.

10.15 Calling a macro from HTML, code and RTF descriptors

10.15.1 HTML and code descriptors

The way to call a macro from an HTML or Code descriptor is to use the [ExternalCall/] Tag.

```
[ExternalCall Macro=MyMacroID]
MyUserData
[/ExternalCall]
```

This syntax automatically calls the macro named "MyMacroID" with the following prototypes:

```
Sub Generate(oObject , oContext, sUserData, sResult)
End Sub
```

`oObject` is the current object in the descriptor.

`oContext` is the generation context

`sUserData` is the string contained in the [ExternaCall/] tag, in this sample, it is "MyUserData".

`sResult` contains the string to be returned to the descriptor.

Or

```
Sub GenerateStream(oObject , oContext, sUserData, oStream)
End Sub
```

`oObject` is the current object in the descriptor

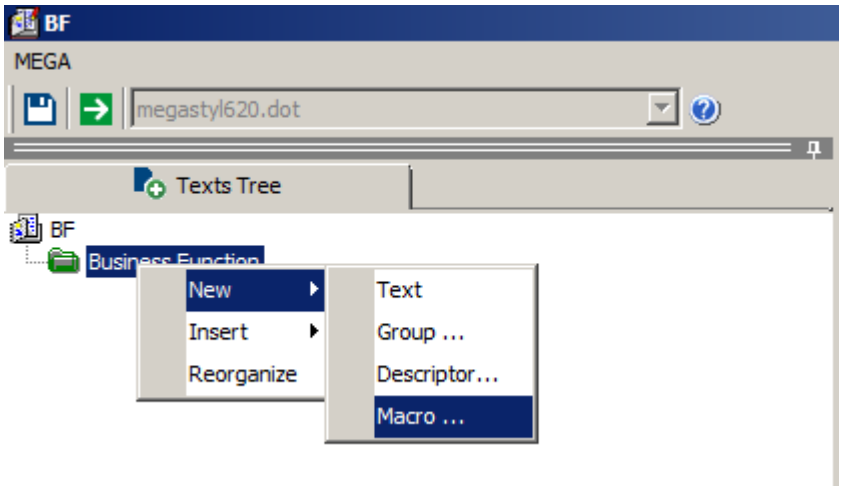
`oContext` is the generation context

`sUserData` is the string contained in the [ExternaCall/] tag, in this sample it is "MyUserData".

`oStream` is a TextStream containing the string to be returned in the descriptor.

10.15.2 RTF Decriptors

The RTF descriptor may include a new macro:



The macro is created and instanciated with the following prototype:

```
Sub Generate(oObject, oContext, sUserData, sResult)
End Sub
```

`oObject` is the current object in the descriptor.

`oContext` is the generation context

`sUserData` is empty at the moment

`sResult` contains the string to be returned to the descriptor.

10.16 Calling an operator

10.16.1 Calling a method (message box display)

You do not expect a return from the method.

Example: message box display

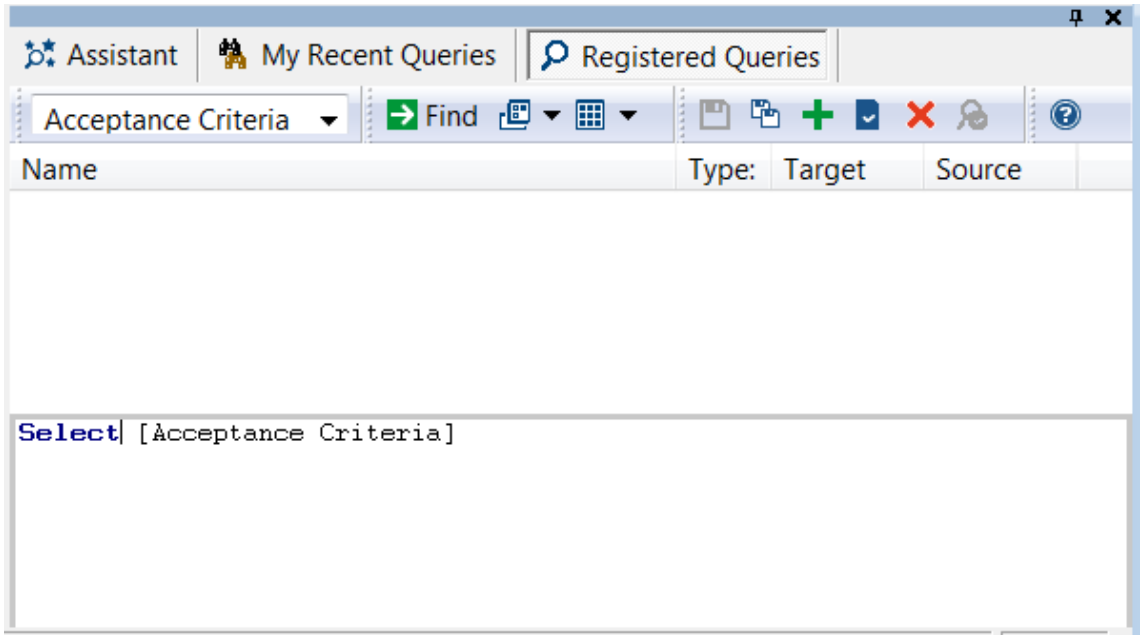
VB Script
```
oRoot.MessageBox("Error: Report Parameter is Misconfigured!")
```

Java
```
MegaRoot.CallMethod(“MessageBox” , "Error: Report Parameter is Misconfigured!")
```

10.16.2 Calling a function (RequestQuery)

The function returns some content).

Example: the function launches the Query window and returns the selected instances.



VB Script

```
Dim mc As MegaCollection
Set mc = Root. RequestQuery
```

Java

```
MegaCollection    pMegaCollection ;
pMegaCollection  = Root. CallFunction(“RequestQuery”)
```

10.16.3 Calling a function (RegulationApply)

With the **RegulationApply** API the **Check > Regulation with propagation** command is available on an object in MEGA (Web Front-End):

- The **RegulationApply** function returns an informal object including the result.
- You can customize the standard report displaying the results of this test.

To use the **RegulationApply** API, you need to apply ".RegulationApply" on the object you want to be checked and use the required **Modeling Regulation** parameter.

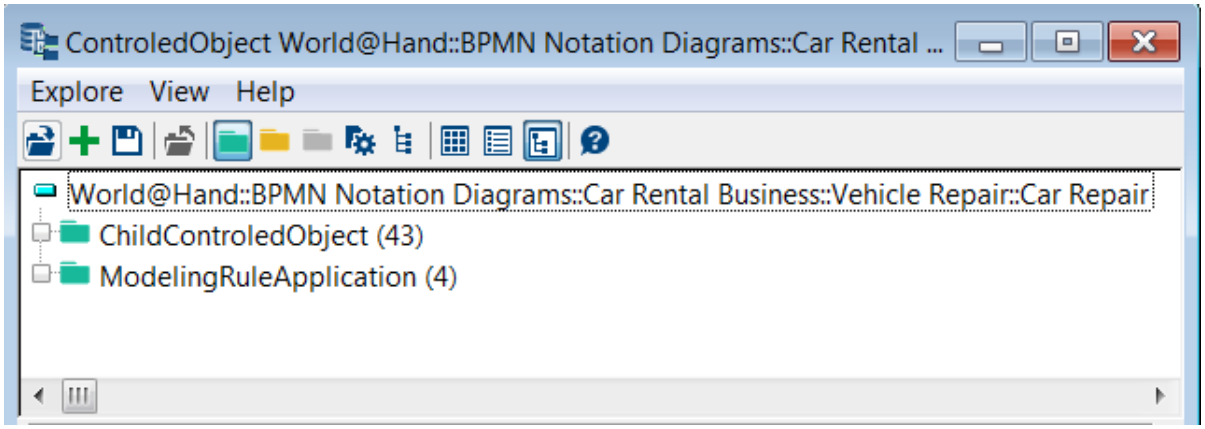
The processing consists in an extraction from the root object to be checked and in applying the Modeling regulation rules to each candidate item.

VB Script

```
Const cstrObjectToValidate = "~rpgNUz5T9570[Car Repair]"
Const cstrModelingRegulation = "~IdCWh3eh9T20[BPMN regulation]"

Dim mgobjObjectToValidate
Set mgobjObjectToValidate = GetObjectFromId(cstrObjectToValidate)

Dim mgobjModelingRegulation
Set mgobjModelingRegulation = GetObjectFromId(cstrModelingRegulation)

Dim mgobjResult
Set mgobjResult = mgobjObjectToValidate.RegulationApply(mgobjModelingRegulation)

mgobjResult.Explore
```

Java

```
String cstrObjectToValidate = "~rpgNUz5T9570[Car Repair]"
String cstrModelingRegulation = "~IdCWh3eh9T20[BPMN regulation]"
MegaObject mgobjObjectToValidate = megaRoot.getObjectFromID(
cstrObjectToValidate)
MegaObject mgobjModelingRegulation = megaRoot.getObjectFromID
(cstrModelingRegulation)
final MegaObject mgobjResult = (MegaObject)
mgobjObjectToValidate.callFunction(“RegulationApply”,mgobjModelingRegulation);
```

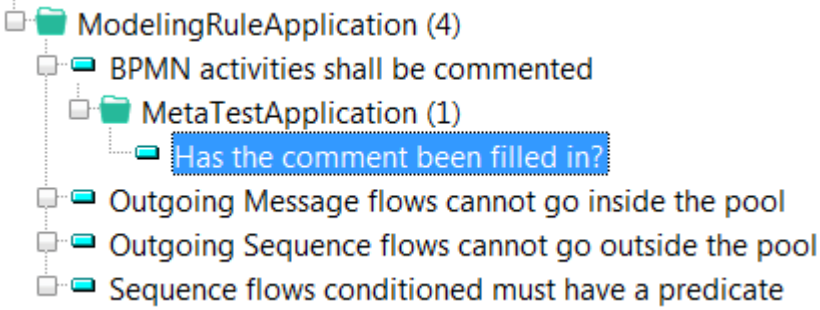


The **Root object** is the object on which the **RegulationApply** function is invoked.

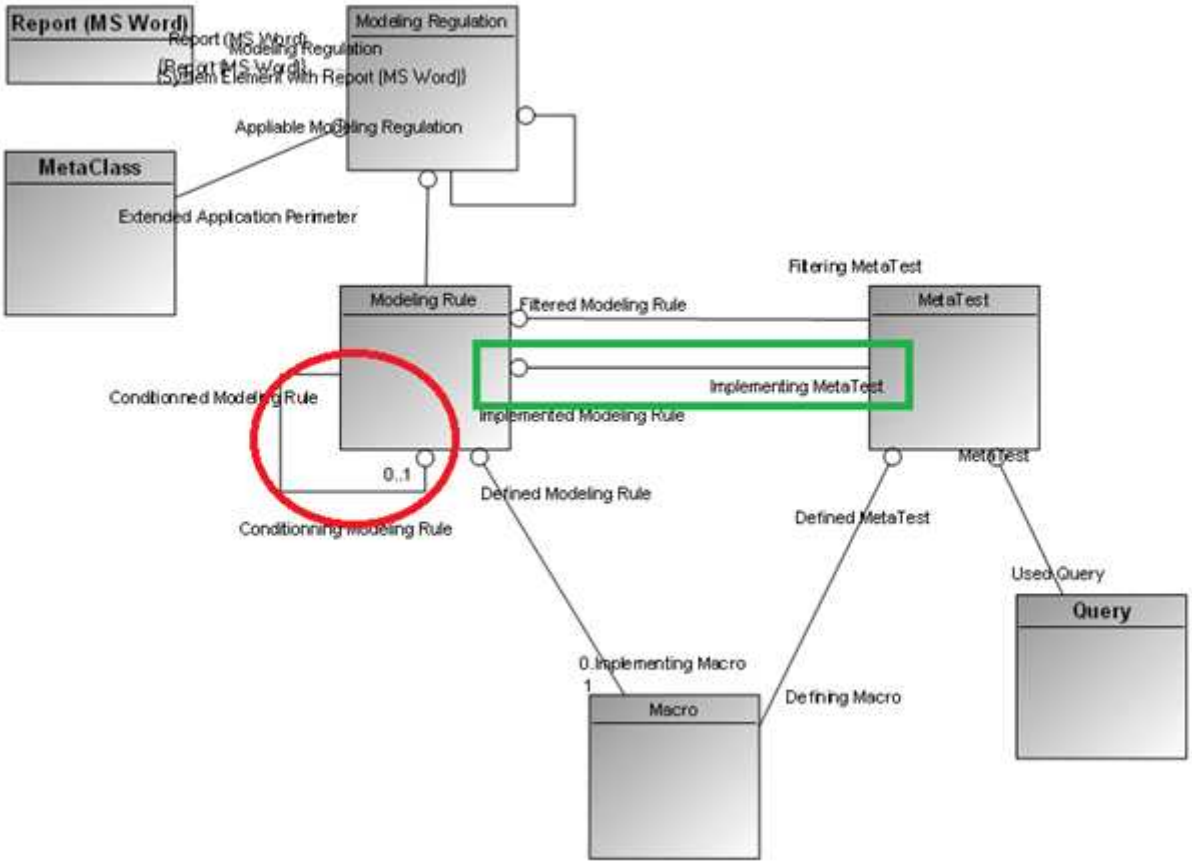
In the above example:

- The root object "ControledObject" holds a set of MEGA objects (43 ControledObjects), which are extracted from the root object to apply the regulations.
- Each "ControledObject" holds a **ModelingRuleApplication** for each ModelingRule applied to this MEGA object.
- Each **ModelingRuleApplication** can include:
  - one or several **MetaTestApplications**

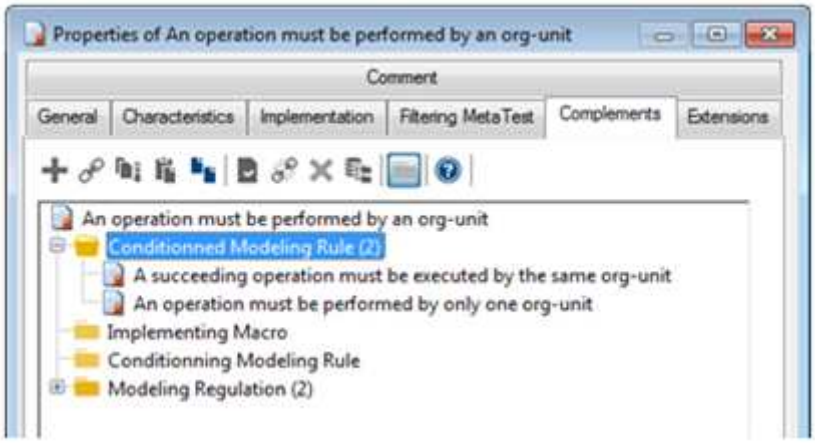
    There are as many MetaTestApplications as there are MetaTests needed to define the modeling rule test.

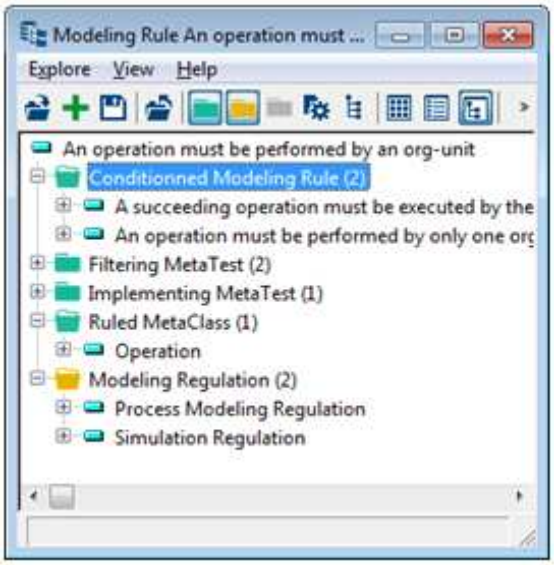
    

  - one or several **ChildModelingRuleApplications** (defined as "Conditioning Modeling Rule" in the following Regulation MetaModel diagram)



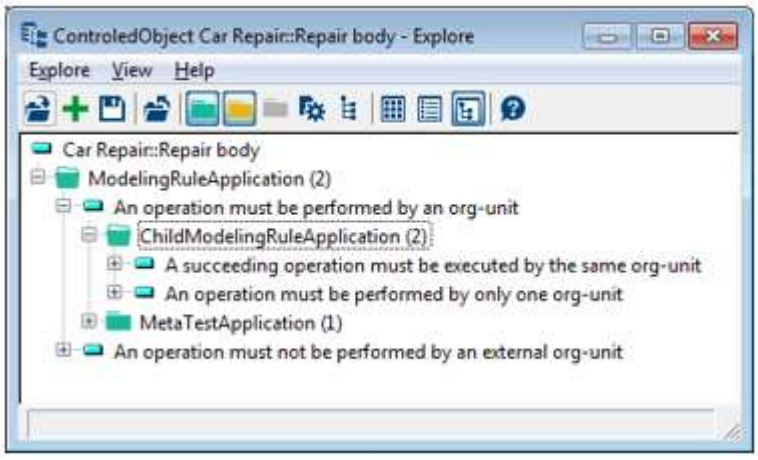
When the **ModelingRule** includes **Conditionned Modeling Rules**, the **ModelingRuleApplication** can hold other **ModelingRuleApplications**.





VB Script

```
Const cstrObjectToValidate = "~GhsmIDTJ)qU1[Repair body]"
Const cstrModelingRegulation = "~ga1YNZOP1H80[Process Modeling Regulation]"

Dim mgobjObjectToValidate
Set mgobjObjectToValidate = GetObjectFromId(cstrObjectToValidate)

Dim mgobjModelingRegulation
Set mgobjModelingRegulation = GetObjectFromId(cstrModelingRegulation)

Dim mgobjResult
Set mgobjResult = mgobjObjectToValidate.RegulationApply(mgobjModelingRegulation)

mgobjResult.Explore
```



10.17 Using MEGA identifiers in the code (Java, VBScript, others)

Here are some useful reminders regarding the use of MEGA identifiers via APIs.

10.17.1 "Physical" type of MEGA identifiers via APIs

You never:

- need to know the data « physical » type of a MEGA identifier.

  e.g.: int, long, float, this physical type that can change from a MEGA release to another.
- have to interpret directly a MEGA identifier that you got from a function of APIs.

In Java:

- all the functions that return an identifier return the Object type

  `e.g.: mgobjMegaObject.getID();`
- any function parameter used to receive a MEGA identifier is of Object type

  `e.g.: mgobjMegaObject.getRoot().currentEnvironment().toolkit().fieldID(Object ID);`

In VB Script :

- all the functions that return an identifier return the Variant type

  `e.g.: MegaObject.getId()`
- any function parameter used to receive a MEGA identifier is of Variant type

  `e.g.: MegaToolKit.fieldID(Vairiant ID)`

Naming convention of a variable that includes a MEGA identifier:

`mgid<etc>`

`e.g.: mgidAttribute, mgidMetaClass, mgidApplication`

10.17.2 Handling identifiers in their « physical » form

Any operation on a MEGA identifier in its “physical” form must be performed through APIs.

Examples:

- Comparing two identifiers:

  `MegaToolKit.sameID(...)`
- Comparing a MegaObject identifier:

  `MegaObject.sameID(...)`
- Identifier transformations:

  `- MegaToolKit.getString64FromID(...)`

```
- MegaToolKit.getStringFromID(...)
- MegaToolKit.getIDFromString(...)
```

10.17.3 MegaFields

Megafields are character strings that enable object identification while including identified object name.

- Do not parse megafield (unless absolute need)
- A megafield might not include the identified object name

```
e.g.: a megafield obtained via MegaObject.megaUnnamedField()
```

It is recommended to declare the constants that correspond to identifiers in megafield form.

This enables to carry out impact analysis of objects used in VB Script and Java macros (via the reference link for example).

As a megafield is a string, the variable that includes a megafield starts with "str".

```
e.g.: "strObjectMegaField".
```

10.17.4 MEGA identifier formats

You can handle MEGA identifiers with different format in your code:

Object / Variant (physical form)

This is what you obtained when you enter:

```
> MegaObject.getID()
> MegaObject.getProp("Att","INTERNAL")
```

Where Att is a "Mega Identifier" type attribute, for example Att = "~310000000D00[Absolute Identifier]"

string in numerical format in base 64

This is what you obtained when you enter:

```
> MegaObject.getProp("Att")
```

Where Att is a "Mega Identifier" type attribute, for example Att = "~310000000D00[Absolute Identifier]"

Identifiers in MGL files, MGR files, etc.

string in numerical format in base 16 (hexadecimal)

This is what you obtained when you enter:

```
> MegaObject.getProp("~H20000000550[_HexaIdAbs]")
> MegaToolKit.getStringFromID(...)
```

Identifiers in XMG files

string MegaField

This is what you obtained when you enter:

```
> MegaToolKit.fieldID(Object ID)
> MegaObject.megaField()
> MegaObject.megaUnnamedField()
```

Some function parameters can receive a value in any type amoung the following ones:

Examples:

```
- MegaRoot.getObjectFromID(Object)
- MegaObject.getProp(Object)
- MegaObject.sameID(Object)
- ...
```

10.17.5 Bad practice examples

Do not manage on your own:

- identifier comparisons
- conversions of types of variables including identifiers.
- name retrieval in megafields

In only a few particular cases you need to parse megafields.

Do not write:

```
  /**
   * Tells if two megaField having the same Id Abs.
   * @param megaField1
   * @return
   */
  @SuppressWarnings("javadoc")
  public static boolean sameIdAbs(final String megaField1, final String
megaField2) {
    if ((megaField1 == null) || (megaField2 == null)) {
      return false;
    }
    String id1 = Pouet.getIdAbsFromMegaField(megaField1);
    String id2 = Pouet.getIdAbsFromMegaField(megaField2);

    return id1.equals(id2);

  }
```

```
/**
 * Get ID Abs from Mega Field
 * @param megaField megaField
 * @return Id Abs
 */
public static String getIdAbsFromMegaField(final String megaField) {
  if ((megaField != null) && (megaField.length() > 13)) {
    return megaField.substring(1, 13);
  }
  return "";

}

/**
 * Get Name contained into megafield Example: for
 * "~a20000000I20[MetaAttributeValue]" results on "MetaAttributeValue"
 * @param megaField megafield input
 * @return Name extracted
 */
public static String getNameFromMegaField(final String megaField) {
  int index1 = megaField.indexOf("[");
  int index2 = megaField.lastIndexOf("]");
  String result = "";
  result = megaField.substring(index1 + 1, index2);
  return result;
```

10.18 Using macros to add calculated attributes

You can:

- define new attributes or parameters for a MEGA object.
- determine value read and save modes for this parameter using MEGA macros.

To calculate an attribute by creating a macro:

1. Open the **Explorer** on the new attribute or new parameter.
2. Right-click the attribute (or parameter) and select **New > Macro**.

   The macro creation wizard appears.
3. Select **Create Macro (VB)Script**.
4. Click **Next**.
5. (Optional) Modify the default Name ("AttributeName".Macro) of your macro.

   A macro is an object containing a VB Script code sequence interpreted at execution.
6. Click **Finish**.

Edit the "AttributeName".Macro macro and note that the VB Script contains in particular the following functions:

If they are not present, standard implementation is selected.

- **GetAttributeValue(ByVal Object, ByVal AttributeID, ByRef Value)**

  Defines attribute access mode. The parameters are:
  - Object: corresponds to the object of which the attribute value is requested.

- AttributeID: absolute identifier of the attribute (or taggedValue).
- Value: the attribute value returned by the function, concerning this object.

- **SetAttributeValue(ByVal Object, ByVal AttributeID, ByVal Value)**

  Defines attribute save mode. The parameters are:

  - Object: corresponds to the object of which the attribute value must be updated.
  - AttributeID: absolute identifier of the attribute (or taggedValue).
  - Value: the function saves the attribute value for this object.

The attribute nature (_AtNature) should be Virtual.

For both of these functions, attribute change mode is a character string. Conversion must be carried out to change text format to the internal format of the attribute.

Example:

VB Script

```
Sub GetAttributeValue (ByVal object, ByVal AttID, Value)
' internal value reading in integer format. numValue = CInt(objet.GetProp(AttID, "Physical"))
if numValue < 20 then
Value = "Young"
elseif numValue < 35 then
Value = "Youthful"
elseif numValue < 55 then
Value = "Mature"
else
Value = "Elderly"
end if
End   Sub
```

Java

```
public void getAttributeValue(final MegaObject mgobjObject, final Object AttID, final StringBuffer value) {
// internal value reading in integer format.
String strResult = "";
int iNumValue = (Integer) mgobjObject.getProp(AttID, "Physical");
if (iNumValue < 20) {
strResult = "Young";
} else if (iNumValue < 35) {
strResult = "Youthful";
} else if (iNumValue < 55) {
strResult = "Mature";
} else {
strResult = "Elderly";
}
value.append(strResult);
}
```

You can directly implement read-only and read/write access in the attribute format (without passing via standard conversion).

In this case, you must implement the following two functions, of which prototypes are similar to those above:

VB Script
```
GetExtendedAttributeValue(ByVal Format as LONG, ByVal Object, ByVal
AttributeID, ByRef Value)
```

Java
```
public void getExtendedAttributeValue(final int[] intFormat, final MegaObject
mgobjObject, final Double dAttributeID, final StringBuffer strValue) {}
```

VB Script
```
SetExtendedAttributeValue(ByVal Format as LONG, ByVal Object, ByVal
AttributeID, ByVal Value))
```

Java
```
public void setExtendedAttributeValue(final int[] intFormat, final MegaObject
mgobjObject, final Double dAttributeID, final StringBuffer strValue) {}
```

The difference is in the additional parameter: Format. The possible values are:

- 0 internal: value in internal format (binary, integer,...)
- 1 external: value in external format, but before display processing (certain objects have external form that is textual with the addition of index identifiers, as for class attributes or association roles).
- 3 Display: value in external format used in Web sites or Word documents (expurgated when identifiers in external format occur).

If one of the two extended functions is implemented, call by GetProp with "Physical" format on the same attribute is prohibited since it would lead to an infinite recursion.

10.19 Launching a tool in HOPEX using APIs

To allow a tool instantiation in a desktop, you need a MegaRoot. The prototype is as follows:

```
Function AddParameterizedTool( sParameterization as string) as string
```

sParameterization is as follows:

`Tool=` tool idabs

`Param=` tool parameter idabs

`ParamValue=` parameter value – object idabs if it is a MEGA object

`ParamValueMetaclass=` MetaClass idabs if the parameter value is a MEGA object

`Affinity=` Affinity idabs

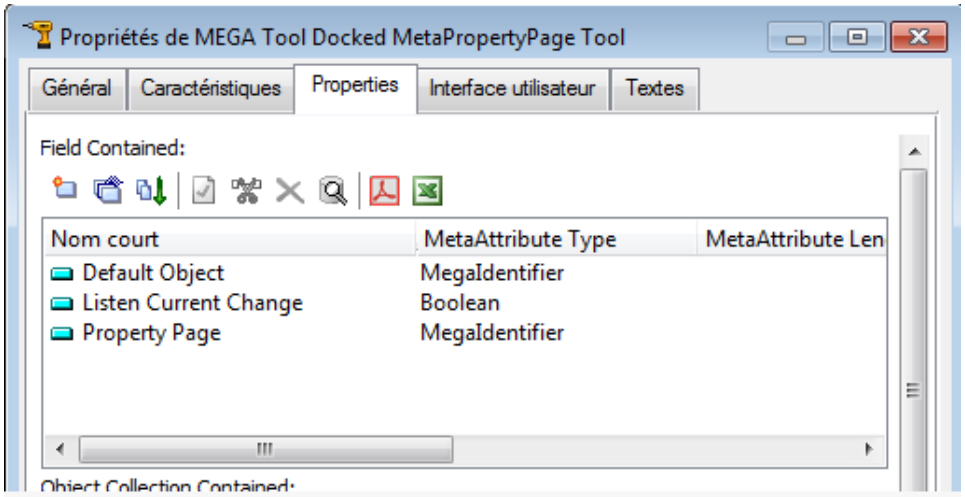
`Tool=` tool idabs

…

1. The function returns the JSON read by the client.
2. The client launches the tools and put them at the right places (according to defined candidates and affinities).
3. Tools can be instantiated with particular objects.

Example 1

Display in the desktop the property page tool positionned on OIZYiqujArn0 object (Advisor Client) of type MrUiM9B5iyM0 (Application)



Code:

VB Script

```
print GetRoot.AddParameterizedTool(
"Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclas
s=MrUiM9B5iyM0 ")
```

Java

```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgRoot.invokePropertyGet("AddParameterizedTool",
"Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclas
s=MrUiM9B5iyM0 "));
```

Result:

```
{"Tools":[{"toolId":"Sj%283hVHpEXUN","parameters":[{"parameterId":"VHRbsTkZFLvH","paramete
rName":"defaultObject","parameterValue":"OIZYiqujArn0","metaClassIdAbs":"MrUiM9B5iyM0"}]}]
}
```

Example 2

1. Display property page tool and associate « Default object » parameter with OIZYiqujArn0 object (Advisor Client) of type MrUiM9B5iyM0 (Application),
2. Launch the diagram editor,
3. Launch the macroLauncher tool with the macro that has the NiqwTtU5CPB0 identifier and the additional parameter « This is a test »

Code:

VB Script

```
print GetRoot. AddParameterizedTool( "
Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclass
=MrUiM9B5iyM0,Tool=Fk(3zVHpEXWN,Tool-
4i(35ZHpEHhN,Param=KXcGkNdEF1bR,ParamValue=NiqwTtU5CPB0,Param=6RQQCjsNFT)M,Param
Value=This is a test ")
```

Java

```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgRoot.invokePropertyGet("AddParameterizedTool",
"Tool=Sj(3hVHpEXUN,Param=VHRbsTkZFLvH,ParamValue=OIZYiqujArn0,ParamValueMetaclas
s=MrUiM9B5iyM0,Tool=Fk(3zVHpEXWN,Tool-
4i(35ZHpEHhN,Param=KXcGkNdEF1bR,ParamValue=NiqwTtU5CPB0,Param=6RQQCjsNFT)M,Param
Value=This is a test"));
```

Result:


```
{"Tools":[{"toolId":"Sj%283hVHpEXUN","parameters":[{"parameterId":"VHRbsTkZFLvH","parameterName":"defaultObject","parameterValue":"OIZYiqujArn0","metaClassIdAbs":"MrUiM9B5iyM0"}]},{"toolId":"Fk%283zVHpEXWN","parameters":[{"parameterId":"KXcGkNdEF1bR","parameterName":"macroId","parameterValue":"NiqwTtU5CPB0"},{"parameterId":"6RQQCjsNFT%29M","parameterValue":"This%20is%20a%20test"}]}]}
```


10.20 API access

Diagrams API can be useful to programmatically extract information from diagrams, modifying existing diagrams or creating new diagrams from scratch.

10.20.1 Opening an existing diagram

The first step to handle a diagram is to obtain a MegaDrawing API object. This is achieved by first obtaining a reference on the diagram repository object using the repository API.

The following code sample retrieves a diagram in the repository using its unique identifier:

```
Set oRepositoryDiagram = GetObjectFromId("hk5HbIaO8b70")
```

The diagram can then be "opened", either for read-only or read/write access. The following function returns a MegaDrawing which will allow further manipulation:

```
Set oDrawing = oRepositoryDiagram.Drawing("RO") 'Read-Only
Set oDrawing = oRepositoryDiagram.Drawing("RW") 'Read-Write
```

Note that invoking the Drawing function is like opening the diagram in the desktop: it is a costly call, and for improved performance scripts should avoid opening multiple times the same diagram.

Also note that if the diagram is already opened in the desktop, this function will open a separate instance of the diagram in the state of its last save. Most API modifications on this diagram will not therefore be reflected live in the graphical user interface: it will be necessary to close it without saving, then reopen it. To handle diagrams live see section Setting up interactive plug-ins in a diagram p. 185.

10.20.2 Creating a new diagram

Like any repository object, a diagram must first be created in the repository. However, there are two additional requirements to successfully create a diagram:

- A diagram must have a "nature", which indicates the type of diagram (a flowchart, an organizational chart …). Available natures are listed as Meta MetaAttributeValue of the MetaAttribute "Nature".
- A diagram must describe a repository object (a Business Process, an Application …) and be linked to this object via the adequate MetaAssociationEnd. To find out which MetaAssociationEnd is used as well as the list of allowed natures for a given described object, it is necessary to browse the DiagramTypeZoom linked to the MetaClass of the described object.

```
' Retrieving a Business Process
Set oBusinessProcess = GetObjectFromId("0ynRE)tLxy81")
```

```
Set colDescription = oBusinessProcess.GetCollection("Description")
Set oNewDiagram = colDescription.Add("My new diagram")
oNewDiagram.GetProp("Nature") = "BPDD" ' Business Process Component Diagram
Set oDrawing = oNewDiagram.Drawing("RW")
```

Note that initialization macros are not called when a diagram is created programmatically.

10.20.3 Saving

Unlike repository modification, diagram manipulations are not committed automatically. It is necessary to explicitly save the diagram:

```
oDrawing.Flush
```

10.21 Report DataSets and APIs

10.21.1 Getting a Report DataSat content using an API

To get a Report DataSet content use GetCollection("~Yvazr2mvKf21[DataSet Create]") on the Report DataSet object.

Example: Opening the selected Report DataSets in MEGA explorer

VB Script

```
Dim mgcolDataSubSet
Set mgcolDataSubSet = GetRoot.RequestQuery("~)ilXTIGrKPiB[Report DataSet]")
If Not mgcolDataSubSet Is Nothing Then
  Dim mgobjDataSubSet
  For Each mgobjDataSubSet In mgcolDataSubSet
    Dim mgcolDataSubSetValues
    Set mgcolDataSubSetValues =
mgobjDataSubSet.GetCollection("~Yvazr2mvKf21[DataSet Create]")
    mgcolDataSubSetValues.explore
  Next
End If
```

Java

```
String datasetId="";
MegaObject mgobjDataSubSet = megaRoot.getCollection("~)ilXTIGrKPiB[Report
DataSet]").get(datasetId);
MegaCollection mgcolDataSubSetValues =
mgobjDataSubSet.getCollection("~Yvazr2mvKf21[DataSet Create]");
```

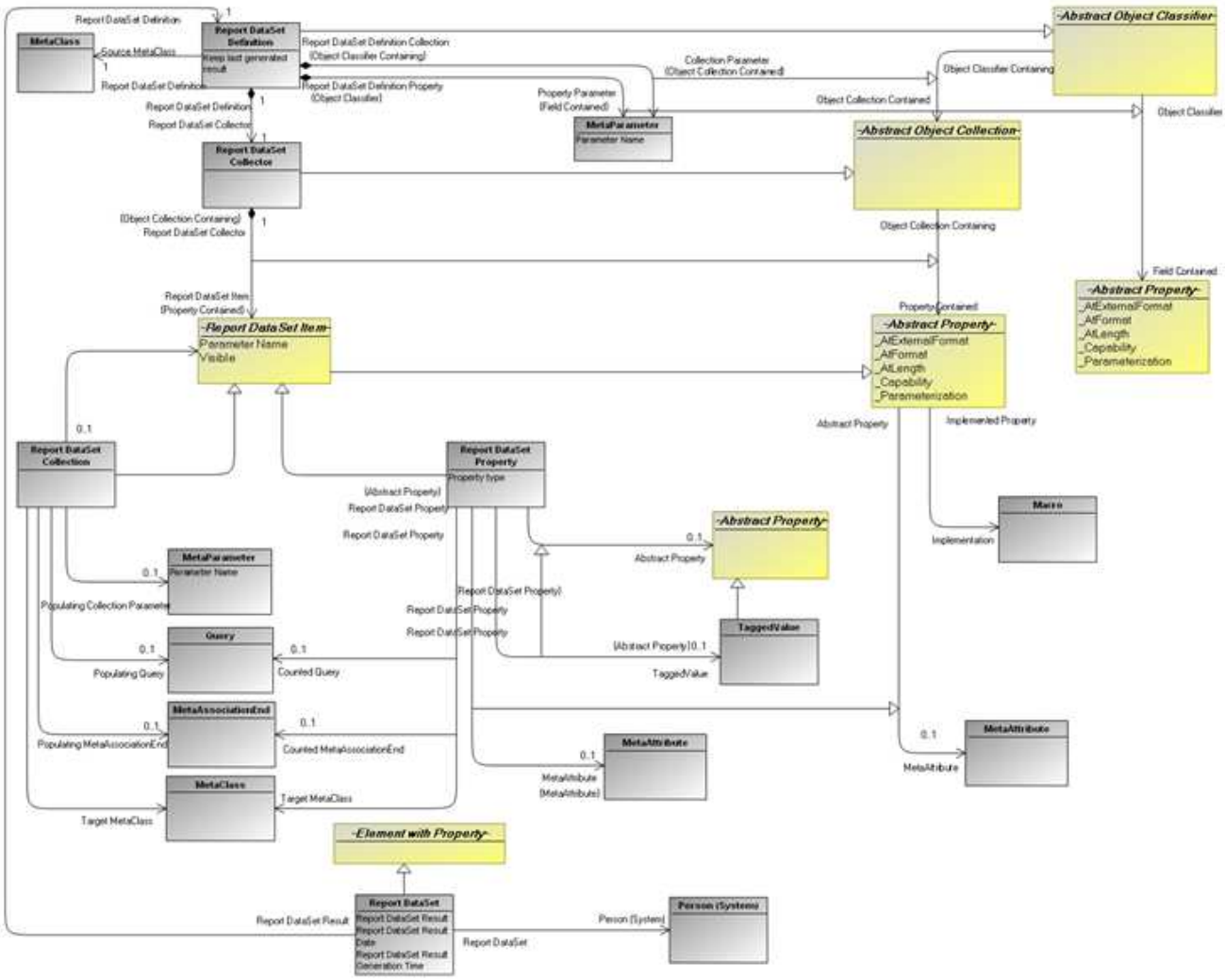
10.21.2 Regenerating a report DataSat content using an API

Once a Report DataSet has been generated in a MEGA session, each successive consultation of this Report DataSet in the same session returns the same previous Report DataSet (the Report DataSet is not calculated at each consultation, it is stored in the cache).

To refresh the Report DataSet content, you need to invoke the CollectionCacheReset operator to delete this cache. In this way, the next consultation of the Report DataSet triggers a new Report DataSet calculation.

To do so, call the CollectionCacheReset operator on the MEGA root, with the Report DataSet identifier and the Report DataSet Collector identifier as argument.

The Report DataSet MetaModel is as follows:



Example: Deleting the cache of all selected report Dataset

VB Script

```
Dim mgcolDataSet
Set mgcolDataSet = GetRoot.RequestQuery("~)ilXTIGrKPiB[Report DataSet]")
If Not mgcolDataSet Is Nothing Then
  Dim mgobjDataSet
  For Each mgobjDataSet In mgcolDataSet
```

```
    GetRoot.CollectionCacheReset mgobjDataSet.GetId,
mgobjDataSet.GetCollection("~rLU9sCZtKLx6[Report DataSet
Definition]").Item(1).GetCollection("~NMU9ZfYtKHh6[Report DataSet
Collector]").Item(1).GetId()
  Next
End If
```

Java

```
megaRoot.callMethod("CollectionCacheReset",
mgobjDataSubSet.getID(),mgobjDataSubSet.getCollection("~rLU9sCZtKLx6[Report
DataSet Definition]").get(1).getCollection("~NMU9ZfYtKHh6[Report DataSet
Collector]").get(1).getID());
```

10.22 Accessing graphical objects in a diagram

The all-around way to access graphical objects in a diagram is via the DrawingItems collection available on the MegaDrawing. Some items can also have child items. The structure of the diagram is therefore a tree of DrawingItems. Items are sorted by depth, from the background to the foreground.

An important property of all DrawingItems is the DrawClassName which indicates its object type. The most important types will be detailed in the sections below.

Sample code illustrating how to print the object tree of a diagram

```
Sub PrintDrawingItems( colDrawingItems, identLevel )
  sPrefix = "-"
  For i = 1 to identLevel
    sPrefix = " " & sPrefix
  Next
  For Each oItem in colDrawingItems
    Print sPrefix & oItem.DrawClassName
    PrintDrawingItems oItem.SubDrawingItems, identLevel+1
  Next
End Sub

PrintDrawingItems oDrawing.DrawingItems, 0
```

10.22.1 Accessing repository objects

Repository objects present in the diagram can be accessed in two ways:

- Via the generic DrawingItems tree. Their class name is “ModeOcc”. Objects returned by this collection support the DrawingItem interface.
- Via the legacy DrawingObjects collection which returns only repository objects in a flat list. Objects returned by this collection support the DrawingObject interface and are NOT sorted by depth.

The newer DrawingItem interface has more features than the legacy DrawingObject interface. It is possible to switch between interfaces, as shown in the example below.

```
For Each oDrawingObject in oDrawing.DrawingObjects
```

```
  Print oDrawingObject.Name
  Set oDrawingItem = oDrawingObject.DrawingItem
  Print oDrawingItem.ItemProperty("ModeOcc_Name")
  Set oDrawingObjectBis = oDrawingItem.ItemProperty("ModeOcc_DrawingObject")
  Print oDrawingObjectBis.Name
Next
```

To add a repository object in a diagram, the MegaObjectInsert function must be used. It is possible to add several graphical representations of the same repository object. If the object was not already present in the diagram, the repository link between the repository object and the diagram will be created automatically. Note that even if the modifications of the diagram are not saved, this link will still be present in the repository.

```
Set oRepositoryObjectToAdd = GetObjectFromId("C(RtlVUV9n42")
Set oNewDrawingItem = oDrawing.MegaObjectInsert(oRepositoryObjectToAdd)
oNewDrawingItem.SetPos 200, 200, 4200, 2200
Set oNewDrawingItem2 = oDrawing.MegaObjectInsert(oRepositoryObjectToAdd)
oNewDrawingItem2.SetPos 5200, 200, 9200, 2200
```

Removing a repository object from the diagram is done using the Erase function, as for any DrawingItem. If the DrawingItem removed is the last representation of a repository object, the repository link between the object and the diagram will be automatically deleted.

10.22.2 Accessing repository links

Repository links present in the diagram can be accessed in two ways:

- Via the generic DrawingItems tree. Their class name is "ModeLink". Objects returned by this collection support the DrawingItem interface.
- Via the legacy DrawingLinks collection which returns only repository links in a flat list. Objects returned by this collection support the DrawingLink interface and are NOT sorted by depth.

The newer DrawingItem interface has more features than the legacy DrawingLink interface. It is possible to switch between interfaces, as shown in the example below.

```
For Each oDrawingLink in oDrawing.DrawingLinks
  Print oDrawingLink.DirectLegName
  Set oDrawingItem = oDrawingLink.DrawingItem
  Set oDrawingLinkAgain = oDrawingItem.ItemProperty("ModeLink_DrawingLink")
  Print oDrawingLinkAgain.DirectLegName
Next
```

Repository links are automatically added or removed in a diagram when a link is added or removed in the repository and connected objects are present in the diagram.

```
' Creation of a link in repository
Set oOperation1 = GetObjectFromId("lktopHlNBL51")
Set oOperation2 = GetObjectFromId("JltopHlNBb71")
Set oNextLeg = GetObjectFromId("KsUirDB5iCu2")
oOperation1.GetCollection(oNextLeg.GetId()).Add(oOperation2)
```

```
' Retrieving the graphical representation of the new link
Function FindDrawingLink( oDrawing, idRpMaster, idRpSlave, idLeg )
  ReDim arDrawingLink(0)
  For Each oDrawingLink In oDrawing.DrawingLinks
    If ( ( idLeg  = oDrawingLink.OppositeLegId       _
           And idRpSlave  = oDrawingLink.OppositeDrawingObject.Id _
           And idRpMaster = oDrawingLink.DirectDrawingObject.Id ) _
      Or ( idLeg  = oDrawingLink.DirectLegId         _
           And idRpSlave  = oDrawingLink.DirectDrawingObject.Id   _
           And idRpMaster = oDrawingLink.OppositeDrawingObject.Id ) ) Then
      ReDim Preserve arDrawingLink(UBound(arDrawingLink)+1)
      Set arDrawingLink(UBound(arDrawingLink)-1) = oDrawingLink
    End If
  Next
  FindDrawingLink = arDrawingLink
End Function

arDrawingLink = FindDrawingLink( oDrawing, oOperation1.GetId(), _
                                 oOperation2.GetId(), oNextLeg.GetId() )
Set oDrawingLink = arDrawingLink(0)
```

It is then possible to modify graphical aspects of the link. Many visual properties of texts and lines are available in structures which must be manipulated by value, and not by reference: API functions usually return copies of these structures and not direct handles to the original structures. This allows preparing a structure and then being able to assign it to different objects. The following example illustrates this distinction when changing the color of a line.

```
Set oDrawingItem = oDrawingLink.DrawingItem
Print Hex(oDrawingItem.Pen.Color) ' For example, returns FF0000FF - blue

' The Pen property returns a copy of the MegaDrawingPen structure so
' the following statement will NOT change the color of the link
oDrawingItem.Pen.Color = &hFFFF0000 ' red
Print Hex(oDrawingItem.Pen.Color) ' still blue

Set oLinePen = oDrawingItem.Pen
oLinePen.Color = &hFFFF0000
' Only the structure copy in oLinePen has been modified
Print Hex(oDrawingItem.Pen.Color) ' still blue,

' This statement will commit the change in the original structure
oDrawingItem.Pen = oLinePen
Print Hex(oDrawingItem.Pen.Color) ' red at last

' Short Version
```

```
Set oLinePen = oDrawingItem.Pen
oLinePen.Color = &hFF00FF00
oDrawingItem.Pen = oLinePen

oDrawing.Flush ‘ Do not forget to save the diagram
```

Modifying the points of a line is done via a special interface. To be successful, the new points collection must keep the original extremities coordinates (the first and the last points). If the line is orthogonal, extra care must be taken to ensure that the new point collection is orthogonal; otherwise the line points will be reset.

```
' Setting orthogonal style
Set oLineStyle = oDrawingItem.LineStyle
oLineStyle.Style = 7 ' orthogonal
oDrawingItem.LineStyle = oLineStyle

‘ Utility function for debug which prints the points of a collection
Sub DumpPoints( colPoints )
  Print colPoints.Count & " points"
  For i = 0 to colPoints.Count - 1
    Print "[ " & colPoints.Abscissa(i) & ", " & colPoints.Ordinate(i) & " ]"
  Next
End Sub

Set colPoints = oDrawingItem.ItemProperty("DrwLine_Points")

' Removing old points, keeping both extremities
For i = 1 to colPoints.Count - 2
  colPoints.RemovePoint(1)
Next

' Creating a "stair"
nbSteps = 7
dw = (colPoints.Abscissa(1)-colPoints.Abscissa(0)) / nbSteps
dh = (colPoints.Ordinate(1)-colPoints.Ordinate(0)) / nbSteps
For i = 1 to nbSteps
  colPoints.InsertPointBefore colPoints.Count-1, _
                              colPoints.Abscissa(colPoints.Count-2), _
                              colPoints.Ordinate(colPoints.Count-2)+dh
  colPoints.InsertPointBefore colPoints.Count-1, _
                              colPoints.Abscissa(colPoints.Count-2)+dw, _
                              colPoints.Ordinate(colPoints.Count-2)
Next
colPoints.Abscissa(colPoints.Count-2) = _
                                 colPoints.Abscissa(colPoints.Count-1)
```

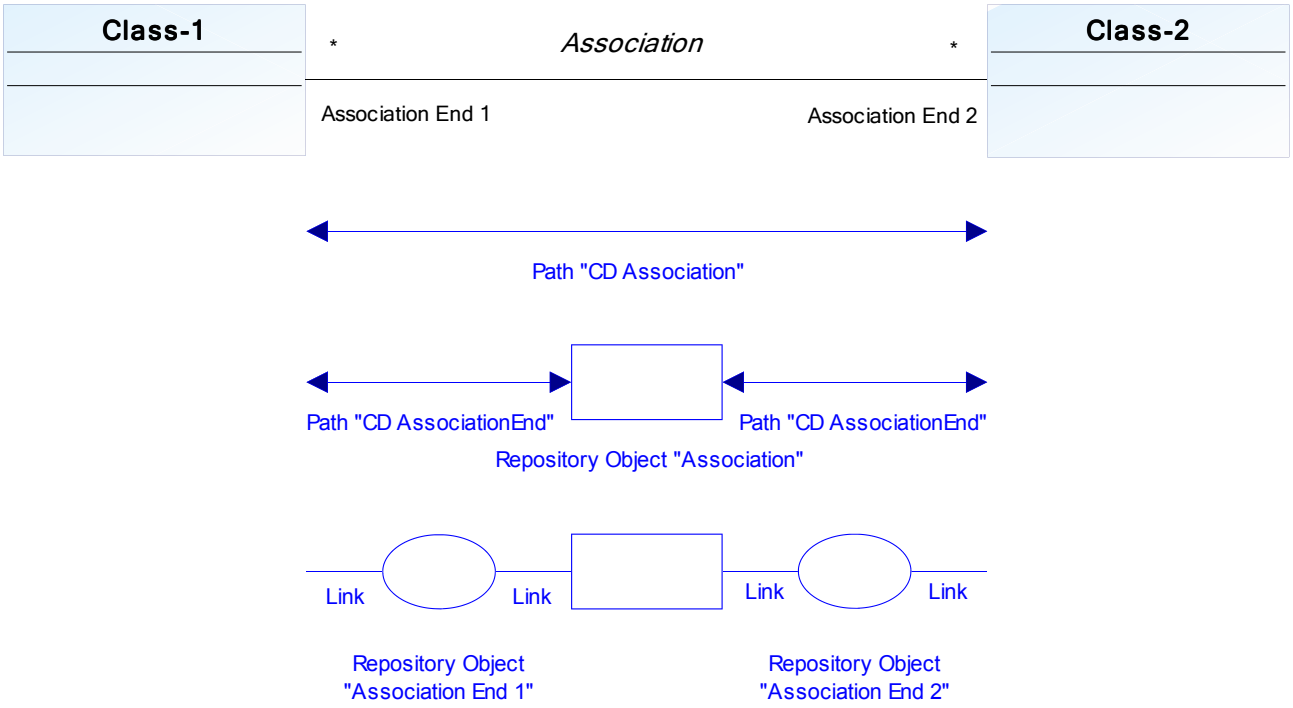
```
oDrawingItem.ItemProperty("DrwLine_Points") = colPoints
```

10.22.3 Paths

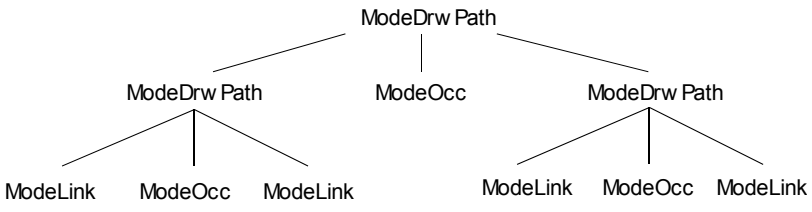
A path is a special kind of graphical object: it is displayed as a link, but actually hides a sequence of repository links and objects. Paths are declared as DiagramTypePath in the diagram parameterization. It is declared as a sequence of:

1. A link or a path
2. An object
3. A link or a path

An example of path is an association in a UML class diagram. It is a multi-level path which chains 3 repository objects.



There is no specific API for paths. They must be accessed via the DrawingItem collection, their DrawClassName is "ModeDrwPath". The DrawingObjects and DrawingLinks hidden by the path can still be accessed via the legacy collections, as they are actually children of the path.



However, as these graphical objects are not displayed, there is no real interest in manipulating them. To modify the aspect of the path (like the color or the points, it is necessary to modify the root ModeDrwPath DrawingItem. However, access to the underlying DrawingObjects and DrawingLinks is necessary to identify the path.

```
Function FindPathItem( oDrawing, idRpObjectCenter )
  ReDim arDrawingItem(0)
  For Each oDrawingObject in oDrawing.DrawingObjects
    If oDrawingObject.Id = idRpObjectCenter Then
      Set oParent = _
          oDrawingObject.DrawingItem.ItemProperty("ParentDrawingItem")
      If oParent.DrawClassName = "ModeDrwPath" Then
        ReDim Preserve arDrawingItem(UBound(arDrawingItem)+1)
        Set arDrawingItem(UBound(arDrawingItem)-1) = oParent
      End If
    End If
  Next
  FindPathItem = arDrawingItem
End Function

Set oAssociation = GetObjectFromId("PhnuDD(NB1W0")
arPathDrawingItem = FindPathItem( oDrawing, oAssociation.GetId() )
Set oPathDrawingItem = arPathDrawingItem(0)
```

To add a path to diagram, all necessary objects and links must first be created in the repository and the extremity objects must appear in the diagram. The center object of the top level path must then be added. The remaining objects and links will be added automatically and a path will be generated. It can then be accessed as previously.

```
oDrawing.MegaObjectInsert oAssociation
arPathDrawingItem = FindPathItem( oDrawing, oAssociation.GetId() )
Set oPathDrawingItem = arPathDrawingItem(0)

Set oPathPen = oPathDrawingItem.Pen
oPathPen.Color = &hFF0000FF
oPathDrawingItem.Pen = oPathPen
```

10.23 Setting up interactive plug-ins in a diagram

Availability: MEGA 2007 Release

You can set macros as diagram plug-ins for a given DiagramType. These macros are called when:

- diagrams are opened or saved,
- graphical representations of repository objects are moved or resized
- represented objects are modified.

In response to a user input, they can then further modify the diagram or the repository, for example to perform an automatic layout or to create complex repository relationships which cannot be achieved with standard parameterization.

In addition, diagram plug-ins help declare and implement macro commands; each macro command is associated with a button in a specific "macro" toolbar.

10.23.1 Writing a diagram plug-in

A diagram plug-in must be implemented in a Mega "Macro" repository object. It defines a number of functions which are called in response to events occurring in the diagram.

Write the following code in the macro editor and save it as a macro.

```
' ********************************************************************
' Event handlers - each handler implementation is optional
Sub OnLoadDrawing( oDrawing As MegaDrawing )
End Sub

Sub OnMoveDrawingObject( oDrawing       As MegaDrawing, _
                         oDrawingObject As MegaDrawingObject )
End Sub

Sub OnResizeDrawingObject( oDrawing       As MegaDrawing, _
                           oDrawingObject As MegaDrawingObject )
End Sub

Sub OnSaveDrawing( oDrawing As MegaDrawing )
End Sub

Sub OnObjectChanged( oDrawing       As MegaDrawing, _
                     oDrawingObject As MegaDrawingObject )
End Sub

' New events in Mega 2009 SP2
Sub OnInsertDrawingObject( oDrawing       As MegaDrawing, _
                           oDrawingObject As MegaDrawingObject)
End Sub

Sub OnEraseDrawingObject( oDrawing       As MegaDrawing, _
                          oDrawingObject As MegaDrawingObject)
End Sub

Sub OnUndo( oDrawing As MegaDrawing)
  ' WARNING: no modification should be made, this is only for updating
  ' internal structures
end Sub


' ********************************************************************
```

```
' Plug-in custom toolbar button declaration
' To add macro commands, the three following methods must implemented.
' Otherwise none is required to be implemented.

' Returns the number of macro commands implemented by the plug-in.
Sub GetCommandCount( oDrawing, nCmd )
  nCmd = 1
End Sub

' For each command from 1 to count (returned by GetCommandCount),
' this method should return the text description of the command in sCmdName
' and optionally the ID of an existing MetaPicture which will be used for the
' button in the toolbar instead of the default picture.
Sub GetCommandDescription( oDrawing, nCmd, idPict, sCmdName)
  If nCmd = 1 Then
    sCmdName = "ZOrderCompare test"
    idPict = oDrawing.GetRoot.CurrentEnvironment.Toolkit _
                     .GetIDFromString("~uxuU1odd5z00[Book]")
  End If
End Sub

' This method will be called when the user clicks the corresponding button
Sub CommandCall( nCmd, oDrawing )
  If nCmd = 1 Then
    Set mySelCol = drawing.SelDrawingItems
    Set myItem   = mySelCol.Item(1)
    Set myObjCol = drawing.DrawingObjects
    For Each obj In myObjCol
      If myItem.ZOrderCompare(obj.DrawingItem) = 1 then
        MsgbBx "Selected object is BEHIND " & obj.Name
      Else
        MsgBox "Selected object is IN FRONT of " & obj.Name
      End If
    Next
  End If
End Sub
```

10.23.2 Writing a drag'n drop plugin

Just like the diagram plug-in, you can handle your own behavior for the drag'n drop in the diagram. The drag'n drop plug-in must be implemented in a MEGA "Macro" repository object. It defines a number of functions which are called in response to events occurring in the drag'n drop.

Even if technically a diagram plug-in macro can be used to implement the custom drag'n drop behavior it is strongly recommended to use distinct macro for diagram plug-in and drag'n drop plug-in.

```
Function OnDragEnter(oDragnDropContext as Object)
End Function
```

The On DragEnter function is called each time a drag'n drop enters the diagram area.

The return value must be one of the following options:

- 0: This is the default behavior. The Drag'n drop plug-in does not implement a specific behavior for this drag'n drop
- 1: The behavior of the drag'n drop is handled by the plug-in. The data of the drag'n drop can be dropped in the current diagram.

- 2: The behavior of the drag'n drop is handled by the plug-in. The data is not accepted for the drop action in the current diagram.

The oDragnDropContext is a context object available for each methods of this plug-in. It is created when a given drag'n drop action enters the given diagram area for the first time. This object is available until the drag'n drop is dropped in the given diagram or when another drag'n drop enters the diagram's area.

This object supports the following methods:

- GetRoot: Returns the MEGARoot
- GetDrawing: Returns the drawing of the current diagram.
- GetSourceObjClassId: Returns the MetaClass Id of the data attached to the current drag n drop.
- GetSourceObjCollection: Returns the collection of the data attached to the drag n drop.
- GetDragPosX: Returns the position in the diagram of the cursor.
- GetDragPosY: Returns the position in the diagram of the cursor.
- GetBag: Returns a bag attached to the context object.

```
Function OnDragOver(oDragnDropContext as Object)
End Function
```

The OnDragOver is called each time the drag'n drop moves over the diagram's area. Only very light computation should be implemented in this function.

To lighten the computation of this function you can for example do your computation in the OnDragEnter function and store the result on the context object thanks to its bag.

```
Sub OnDragLeave(oDragnDropContext as Object)
End Sub
```

This procedure is called each time the drag'n drop leaves the diagram's area. It does not have any return value.

```
Function OnDragDrop(oDragnDropContext as Object)
End Function
```

This function is called when the drag'n drop data is dropped on the diagram's area. If you allowed the drag'n drop action with function OnDragEnter or OnDragOver, you must handle the drag'n drop here.

The available return values are:

- 0: The drag'n drop was not handled by the plug-in.
- 1: The drag'n drop was handled by the plug-in.

10.23.3 Registering the macro on a DiagramType:

In the properties box of the DiagramTypeParam of the DiagramType of a diagram, in the _settings text, in the [Macros] section, just add an entry like <n>=<macroIdAbs> where n is any integer not already used and macroIdAbs is the macro idAbs which can be easily obtained via the "Field insert" button of the Text page toolbar.

Note:

When the macro is modified, the diagram needs to be reloaded to take changes into account.

The drawing given to each method of the plugin is a Read/Write MegaDrawing.

Several plugins can be set up on one MetaDiagramType. Each plugin can provide macro commands, however the overall maximum number of macro commands is 16.

10.24 Writing a dynamic query

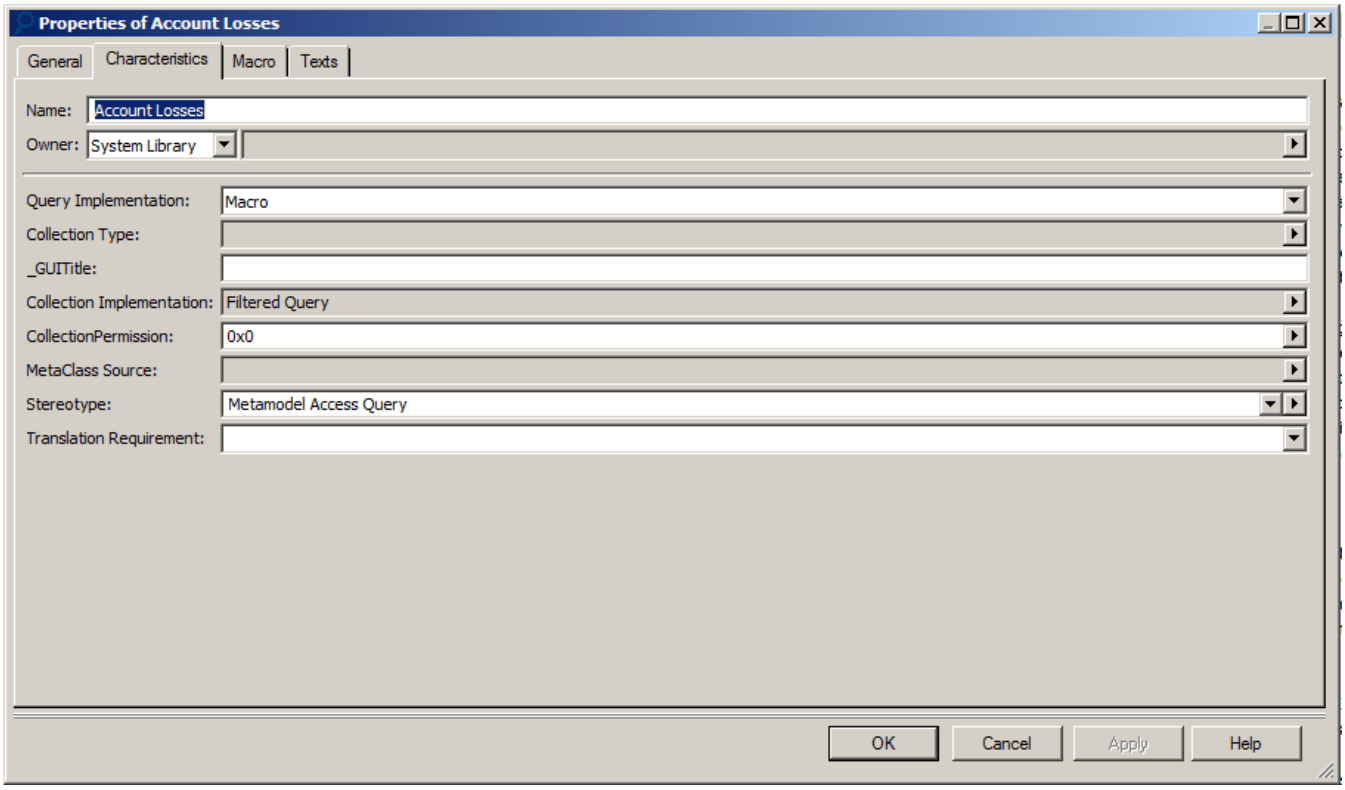
A query retrieves a set of objects.

A dynamic query is a query that cannot be written with an ERQL syntax; Algorithm is necessary to retrieve the collection content.

A dynamic query is based on the Query concept.

To write a dynamic query:

1. Open the query **Properties** window.
2. Set its **Query Implementation** attribute to "Macro".
3. Set its **collection Implementation** attribute to the macro that will compute the collection:



4. It is recommended to implement the query code in VB Script except if you have very complex code. In that case, you can use Java

   Two prototypes can be defined when computing the collection:

VB Script

```
Function getSelectionCollection (oSourceObject as MegaObject, SelectorID as Variant)
End Function


Sub FillSelectionCollection(oSourceObject as MegaObject, SelectorID as Variant, mgcoll as MegaCollection)
End Sub
```

Java

```
public void InvokeOnCollection(final MegaCollection lst )
{
}
Or if you want your method to return a value, use
//return an integer
public int InvokeOnCollection(final MegaCollection lst )
{
}
```

The **getSelectionCollection** creates and sends back the collection to be retrieved. This function can be used particularly when the collection contains different kind of objects.

The **FillSelectionCollection** has the collection already initialized. The function role is just to fill it in.

The arguments are:

- the source object from which the query may be requested or the MegaRoot
- the query identifier.

10.25 Accessing rules and regulations using APIs

10.25.1 Accessing regulations using APIs

Implementing a regulation

```
myObject.ApplyRegulation(myRegulation as MegaObject) as Boolean
```

Applies the regulation to the object.

Returns **True** or **False** according to the regulation compliancy.

Defining the default active regulation

```
myRegulation.RegulationActivate = True
```

myRegulation becomes the active regulation.

To get the result of implementing this regulation to a specific object, read the calculated text "object control report".

Example:

VB Script

```
set myRule = GetObjectFromID("~W6Or(qjB2zB0[Architecture Modeling Regulation]")
myRule.RegulationActivate = True
print Application.item(1).getprop("object control report","Display")
myRule.RegulationActivate = False
```

Java

```
MegaObject mgobjMyRule = mgobjMegaObject.getRoot().getObjectFromID("~W6Or(qjB2zB0[Architecture Modeling Regulation]");
                mgobjMyRule.invokePropertyPut("RegulationActivate", true);
                mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", mgobjMegaObject.getRoot().getCollection("Application").get(1).getProp("object control report ", "Display"));
                mgobjMyRule.invokePropertyPut("RegulationActivate", false);
```

By default «object control report» text is in HTML format. « Display » keyword enables to display it in text mode.

```
myRegulation.RegulationActivate = False restaure le règlement actif défini dans les options utilisateurs
```

10.25.2 Accessing rules using APIs

To apply "myRule" rule to "myObject" objet, you can use the following symmetrical methods:

```
myRule.RuleApply(myObject as MegaObject) as Boolean
myObject.ApplyRule(myRule as MegaObject) as Boolean
```

Both functions send **False** when the rule is not complied with.

Reminder: the rule description is included in the text "Description of the rule". When the rule is not complied with, the message to be displayed is "When the rule is not complied with"

Example:

VB Script

```
set myRule = GetObjectFromID("~o6OrCgnB2fp2[An application cannot be its own component]")
print myRule.RuleApply(Application.item(1))
print Application.item(1).ApplyRule(myRule)
print myRule.GetProp("Description of the rule")
print myRule.GetProp("When the rule is not complied with")
```

Java

```
MegaObject mgobjMyRule = mgobjMegaObject.getRoot().getObjectFromID("~o6OrCgnB2fp2[An application cannot be its own component]");
                mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", mgobjMyRule.invokeFunction("RuleApply", mgobjMegaObject.getRoot().getCollection("Application").get(1)));
                mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", mgobjMegaObject.getRoot().getCollection("Application").get(1).invokeFunction("ApplyRule", mgobjMyRule));
                mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", mgobjMyRule.getProp("Description of the rule"));
                mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", mgobjMyRule.getProp("When the rule is not complied with"));
```

10.26 Business Documents and APIs

The following APIs are available in the frame of the Document Management module regarding static documents:

- StaticDocumentFilePathGet
- DesktopUrlBuild with DocumentLauncher
- SaveAsStatic

10.26.1 StaticDocumentFilePathGet

Retrieving the address of a physical file from a static document (static document version, system static document or system static document version) that can be used to send a document as an attachment to a mail for example.

Function prototype:

```
Function StaticDocumentFilePathGet() As String
```

Use example:

mgoDoc a static document (static document version, system static document or system static document version) retrieves the path to the file containing the static document.

```
strFilePath = mgoDoc.CallFunction("~FFopcJjTGnIC[StaticDocumentFilePathGet]")
```
or
```
strFilePath = mgoDoc.StaticDocumentFilePathGet()
```

Result examples:

C:\PROGRAM FILES (X86)\MEGA\ENVS\DEMONSTRATION\SYSDB\USER\User\mg_webtmp\68412896\129955724059390000\Exemple de DOCX v1.docx

Warning: the file life cycle is the responsibility of the StaticDocumentFilePathGet function caller. It should be deleted after use.

However the file will be deleted when the transaction is closed

10.26.2 DesktopUrlBuild with DocumentLauncher tool

Generation of a link enabling static document opening from the MEGA Web Front-end.

Use the API DesktopUrlBuild (see Calling a URL construction function using APIs in HOPEX p. 169).

The Mega Tool used here is "Document Launcher": ~LA)RAINPGnHP[Document Launcher]

The Mega Tool parameter is "documentId": ~LB)R72OPGDLP[documentID]

Use as parameter the identifier of the document we want to open and the identifier of the MetaClass of the document we want to open (for example MetaClass Document (static))

Use example:

Let us take a static document with the following absolute identifier: WEsYsj8PGnU4print

VB Script
```
Print mgobjMegaObject.getRoot.DesktopURLBuild
("Tool=LA)RAINPGnHP,Param=LB)R72OPGDLP,ParamValue=WEsYsj8PGnU4,ParamValueMetacla
ss=UkPT)TNyFDK5")
```

Java
```
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]",
mgobjMegaObject.getRoot().invokePropertyGet("DesktopURLBuild",
"Tool=LA)RAINPGnHP, Param=LB)R72OPGDLP, ParamValue=WEsYsj8PGnU4,
ParamValueMetaclass=UkPT)TNyFDK5"));
```

Result example:

http://localhost/HOPEX/default.aspx?userdata=Tool-LA%29RAINPGnHP|LB%29R72OPGDLP-WEsYsj8PGnU4-UkPT%29TNyFDK5

Note:

This MegaTool also functions for static document versions, system static documents, system static document versions and also for external references and system external references. To do this, just modify the ParamValueMetaClass.

10.26.3 SaveAsStatic

Saving a dynamic document (Word document, book, report) as a static document.

Function prototype:

```
Function SaveAsStatic(mgoDocPattern As MegaObject, bBatch As Boolean, strFormat As String)
As String
```

- `mgoDocPattern` static document template (can be Nothing)
- `bBatch` a boolean indicating if we want to create the static document in batch or not (display of static document wizard or not)
- `strFormat`, the generation format. Can be empty "".

  For books, available formats are: "PDF" or "RTF" (RTF by default)

  For reports, available formats are: "PDF", "RTF", "XLS" (RTF by default)

  For documents, format is not taken into account. It depends on the option defined in MEGA (DOC or RTF) and on the generation context (MEGA Windows Front-End or MEGA Web Front-End)
- It returns the identifier of the object (static document) created.

Use example:

Let us take mgoDynamicDoc a book, a report or a document (Word dynamic)

```
sDocId = mgoDynamicDoc.CallFunction("~9Zy9faIVGbdG[SaveAsStatic]",
mgoDocPattern, bBatch, strFormat)
ou
sDocId = mgoDynamicDoc.SaveAsStatic(mgoDocPattern, bBatch, strFormat)
```

10.26.4 Macro Script global properties (MegaPropertyBag)

Script macro global data

At creation of a MEGA macro, a global component is made accessible to macro Script code. This component is named **MegaMacroData**. It enables indication of the script instancing context, and also handling of macro global variables.

- Macro context data: this data comprises read-only variables.
  - **MegaMacroData.GetID**: absolute identifier of the MEGA macro instancing the current script.
  - **MegaMacroData.ServiceID**: absolute identifier of the service implemented by the macro. For example, when a macro implements a *MetaCommand*, this variable contains the absolute identifier of the *MetaCommand*. This function offers the possibility of implementing a reusable macro, since it can access a configuration available on the implemented service.
  - **MegaMacroData.InitString**: macro initialization string. In certain use cases, a macro is initialized with a configuration string; this enables macro reuse without creating a specific service. This variable enables access to this initialization string.

- *PropertyBag* associated with a macro: this function enables definition of macro global properties, which is independent of script instances executing macro methods.

  The Microsoft script interpreter used to execute macro code is strictly monothread: it can be accessed (or destroyed) only in the thread in which it was created. A macro can be globally instanced and called in a multithread context: in particular this is the case for implementations of *MetaAttribute* and *_Operator*. To operate in these contexts, the macro execution engine instances as many scripts as there are calling threads. Each script instance is strictly independent of the others: In particular this is true for globals declared in a script – including the global code, which is then executed for each new script instance. If, for optimization reasons, it is desirable to maintain globals valid for all macro script instances, we must have available a mechanism independent of the script. To do this, a global *PropertyBag* is made available to the macro.

  MegaMacroData.GetBag enables access to the global *PropertyBag* of the macro. This *PropertyBag* is a component implementing the **MegaPropertyBag** interface described in the paragraph below.

MegaPropertyBag component

Such a component enables management of a list of named properties, in a similar way as for a javascript class. This data can be managed and maintained independently of the instances of scripts that handle them.

You can access a **MegaPropertyBag**, either:

- by using MegaMacroData.GetBag property, available on each macro script, or
- by instantiating specifically a MegaPropertyBag as follows:

VB Script
```
MegaToolkit.CreateMegaObject(« MegaPropBag »)
```

Java
```
MegaPropertyBag mgpbBag = new
MegaPropertyBag(mgRoot.currentEnvironment().toolkit());
```

Default mode of MegaPropertyBag operation does not require prior declaration of stored variables. Its operation is therefore very similar to that of a javascript class, as the following example shows:

```
Set Bag = CurrentEnvironment.Site.Toolkit.CreateMegaObject("MegaPropBag")
Bag.Prop1 = "Value 1"
Bag.Prop2 = 17
print Bag.Prop1
print Bag.Prop2
```

Unlike VBScript assignment, object reference assignment is automatic, and keyword **Set** should not be included in assigning an object to a property:

```
Bag.Prop3 = GetCollection("Package").Item(1)
print Bag.Prop3.GetProp("Name")
Keyword Item enables generic access to a property:
print Bag.Item("Prop1")
```

You can configure a propertyBag to impose declaration of properties by means of the Explicit method; this system limits risk of error when you want to share content of a propertyBag, or simply to avoid programming bugs (assured by the Explicit Option in VBScript).

```
Bag.Explicit
```

Used as a function, Explicit indicates the PropertyBag operating mode.

In Explicit mode, you need to declare a property – using Dim method – before using it, either for reading or update.

```
If Bag.Explicit Then
Bag.Dim "Prop1"
Bag.Dim "Prop2"
End If
Bag.Prop1 = "Value 1"
Bag.Prop2 = 17
```

in VBScript, the propertyBag is also an iterator enabling listing of properties used (either in update or consultation).

```
For Each propName In Bag
print "Prop " & propName & " = " & Bag.Item(propName)
Next
```

In languages in which the iterator cannot be used (such as java or javascript), we can access properties using an index and the Count function. In this case, we cannot access the property name (future function ItemName).

VB Script

```
For propIndex = 1 To Bag.Count
print "Prop #" & propIndex & " = " & Bag.Item(propIndex)
Next
```

Java

```
MegaPropertyBag mgpbBag = new
MegaPropertyBag(mgRoot.currentEnvironment().toolkit());
mgpbBag.explicit(true);
mgpbBag.dim("Prop1");
  mgpbBag.dim("Prop2");
  mgpbBag.basedObj.invokePropertyPut("Prop1", "Value1");
  mgpbBag.basedObj.invokePropertyPut("Prop2", 17);
  for (int j = 1; j <= mgpbBag.count(); j++) {
  mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", "Prop #" + j + " = " +
mgpbBag.item(j));
  }
```

You can test the nature of a *propertyBag* variable, so as to determine if it is an object or a data item, by means of the **IsObject** function.

If the list of properties indicated in the above example includes objects, these should be treated differently:

```
For Each propName In Bag
If Bag.IsObject(propName) Then
Set propItem = Bag.Item(propName)
```

```
print "Prop " & propName & " is an Object"
Else
propItem = Bag.Item(propName)
print "Prop " & propName & " = " & propItem
End If
Next
```

propertyBag expression evaluator.

The *propertyBag* has an expression interpretation function; Evaluated expressions are based on a VBScript syntax and allow only elementary actions:

- numerical or alphanumerical expression calculation

- assignment of properties with these expressions.

In particular, this evaluator is used by indicators in MEGA; it enables call on dynamic code without calling **ExecuteGlobal**. This function offered by Microsoft scripting has the significant drawback of enabling implementation of a trojan horse, since the Script code called has MEGA macro execution rights, and more specifically of the connected user - this therefore representing a potential security risk. However the expressions interpreted by a *propertyBag* do not have all VBScript functions (the **CreateObject** function cannot be called) thus limiting security fault risks.

In these expressions, variables used correspond to *propertyBag* properties. It is nevertheless possible to cite external variables managed by a component given as a parameter to the evaluator; these variables are presented in the expression in the form of fields. The evaluator calls the function whose name is passed as parameter when a field is found in the expression.

A field can comprise a series of fields separated by dots and possibly terminated by an option:

`Field1.Field2.option`

The function implemented allows as parameter the propertyBag itself, which can supply context information required for evaluation of the field using the following contextual functions:

- Number of consecutive fields:

VB Script `Bag.FieldCount`

Java `mgpbBag.fieldCount()`

- Identifier corresponding to nuField field:

VB Script `Bag.FieldValue(nuField)`

Java `mgpbBag.fieldValue(nuField)`

- Option value:

VB Script `Bag.FieldOption`

Java `mgpbBag.fieldOption()`

The evaluator exists in two forms:

- an expression evaluator, which returns a value.

VB Script

```
MegaPropertyBag.Evaluate(
EvaluableString As String,
FieldResolver As Object,
ResolvingFunction As String) As Variant
```

Java

```
public Object evaluate(final String formula, final Object fieldProcessor, final String callbackName)
```

- a code evaluator, which enables assignment of *propertyBag* properties with expressions. This evaluator can include several VBScript instructions separated by ':' or on different lines.

VB Script

```
MegaPropertyBag.Execute(
EvaluableString As String,
FieldResolver As Object,
ResolvingFunction As String)
```

Java

```
public String execute(final String script, final Object fieldProcessor, final String callbackName)
```

Example:

VB Script

```
Class FieldResolver
Function Field(Bag)
' nothing to evaluate
End Function
End Class

Set Bag = CurrentEnvironment.Site.Toolkit.CreateMegaObject("MegaPropBag")
Bag.Prop1 = "Value 1"
Bag.Prop2 = 17
Bag.Sel = GetCollection("Package")
print Bag.Evaluate("Prop1 & Prop2 & Sel.Count", New FieldResolver, "Field")
Bag.Execute("Prop1 = Prop2 & Sel.Count", New FieldResolver, "Field")
print Bag.Prop1
```

Java

```
class FieldResolver {
    public FieldResolver() {}
    public String Field(final MegaPropertyBag mgpbBag) {
      //nothing to evaluate
      return "";
    }
  }
MegaPropertyBag mgpbBag = new MegaPropertyBag(mgRoot.currentEnvironment().toolkit());
mgpbBag.basedObj.invokePropertyPut("Prop1", "Value1");
mgpbBag.basedObj.invokePropertyPut("Prop2", 17);
```

```
mgpbBag.basedObj.invokePropertyPut("Sel", mgRoot.getCollection("Package"));
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", mgpbBag.evaluate("Prop1 & Prop2
& Sel.Count", new FieldResolver(), "Field"));
mgpbBag.execute("Prop1 = Prop2 & Sel.Count", new FieldResolver(), "Field");
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", (String) mgpbBag.item(3));
```

11 CODING RECOMMENDATIONS & PERFORMANCES

11.1 Coding recommendations

11.1.1 Handling Identifiers

You must make reference to an object using its absolute identifier rather than its name: in this case the code is most highly optimized and resists renaming of the instance as well as change of language.

*An absolute identifier is a unique identifier that can be assigned to any instance, characteristic, or link in **MEGA**.*

However, an absolute identifier is less readable than a name, which is why the MEGA Script Editor offers the possibility of using fields rather than the name of the MEGA object.

Each MEGA instance is identified uniquely by its absolute identifier. This identifier is a 64-bit key, represented by MEGA in the form of a 12 character string.

Objects that define the MEGA metamodel (MetaClasses, MetaAssociations and MetaAttributes amongst others) are also MEGA objects; and they therefore have absolute identifiers.

Obtaining identifiers

A data item representing a MEGA object is implicitly considered as an identifier. In particular, an identifier enables positioning in a collection. For example, if "OperationID" variable contains an operation identifier, it can be used to execute a search in a collection of operations as follows:

VB Script

```
Set myOperations = ...
' myOperations is a collection of operations. OperationID = ...
' OperationID is the identifier of an operation.
Set anOperation = myOperations(operationID)
If anOperation.Exists Then
OperationID = anOperation.GetID ' the variable keeps the same value.
End If
```

Java

```
MegaCollection mgcolOperations = mgobjProject.getCollection("Operation");
              // mgcolOperations is a collection of operations.
              Object objOperationID =
mgobjProject.getCollection("Operation").get(1).getID();
              // objOperationID is the identifier of an operation.
              MegaObject mgobjOperation = mgcolOperations.get(objOperationID);
              if (mgobjOperation.exists()) {
                objOperationID = mgobjOperation.getID();
                // the variable keeps the same value.
              }
```

The **GetID** function obtains the identifier of an object in its internal format.

The value returned by the GetID function is only meaningful within the same executable. This value cannot be generalized and must not be used globally.

The **GetPropID** and **GetCollectionID** functions are used to obtain the identifier (not the value) of an object or a MetaAssociationEnd accessible from an instance. These values are persistent and independent of language (in the context of a multilingual repository for example).

VB Script

```
Ope.GetProp("Name")
Ope.GetProp(Ope.GetPropID("Name"))
' These last two lines are synonymous.
```

Java

```
String strName = mgobjOperation.getProp("Name");
                strName =
mgobjOperation.getProp(mgobjOperation.getPropID("Name"));
                strName = (String) mgobjOperation.invokePropertyGet("Name");
```

Note that **GetProp** (like **GetCollection**) accepts a name or identifier as parameter. The **Item** function also accepts name or identifier as parameter:

VB Script

```
Set myOperations = oProject.GetCollection ("Operation")
' the code finds a collection of operations.
Set anOperation = myOperations.Item("operation-1")
```

Java

```
MegaCollection mgcolOperations = mgobjProject.getCollection("Operation");
              //the code finds a collection of operations.
              MegaObject mgobjAnOperation = mgcolOperations.get("operation-1");
```

The object type for an instance is obtained with the **GetClassID** function.

Using fields

This consists of replacing the character string containing the name by a character string starting with escape character '~', followed by the absolute identifier, then by the object name between square brackets.

Field display with standard editor

```
Set myproject=
oRoot.getCollection("~qekPESs3iG30[Project]").Item("~7qv3W01mCz10[MyProject]")
```

Field display with scriptSet editor

The script editor masks the field code and displays the name only, underlined.

```
Set oproject = oRoot.getCollection("Project").Item("MyProject")
```

In the properties dialog box of a macro, the VB Script tab enables edit of VB Script of the macro. The {} ***Hide/Show Fields*** *button enables display of fields used in the macro, according to"ScriptSet" mode or to "standard" mode.*

Fields can be used in different functions such as: ***GetCollection***, ***GetProp***, ***SetProp***, ***Item***, ***GetMacro***, ***GetObjectFromID***.

Using standard functions available to convert and compare identifiers

Using the MegaField function

To represent a MEGA field identically from one editor to another, the **MegaField** function builds the field corresponding to the identifier of an object.

VB Script

```
myprojectField = myproject.MegaField
Set operations = Project.Item(myprojectField).getCollection("Operation")
```

Java

```
String strProjectField = mgRoot.getCollection("~qekPESs3iG30[Project]").item("My Project").megaField();
                MegaCollection mgcolOperations = mgRoot.getCollection("~qekPESs3iG30[Project]").item(strProjectField).getCollection("Operation");
```

Using directive fields

Fields can also be used to invoke methods of a ***MegaObject***. This type of MEGA script cannot be used directly and must be transformed in order to be executed. This transformation is controlled by the ***Fields*** directive in the **MegaContext** command invoked below.

```
'MegaContext(Fields)   - field interpretation activation
'MegaContext(Fields,Batch) - field interpretation and global DBRoot deactivation
```

When the "Fields" context is activated, the VB script is pre-interpreted so that its fields can be replaced by expressions compatible with VB Script syntax. MetaClasses, MetaAttributes and MetaAssociationEnds can then be pasted in the script in the form of fields rather than the corresponding object names.

Example:

The following code that does not use fields:

```
for each ope in myOperations
  print ope.Nom
next
```

can be replaced by the following which uses them:

```
'MegaContext(Fields)
for each ope in myOperation
  print ope.Name
next
```

When this script is opened with an editor that does not process MEGA fields (for example Windows Notepad, the fields appear in their storage format.

```
'MegaContext(Fields)
for each ope in myOperation
  print ope.~210000000900[Name]
next
```

This file cannot be directly executed by the script interpreter. The **MegaContext(Fields)** option enables transformation of this code to a script acceptable by the script interpreter. For information, this transformation is limited to moving an opening square bracket: the code below can therefore be executed.

```
'MegaContext(Fields)
for each ope in [~gsUiU9B5iiR0myOperation]
  print ope. [~210000000900Name]
next
```

11.1.2 How to speed up queries in API code by using Absolute Identifiers

The way you use to write queries in API code may affect the performance of the query.

The following information shows how to raise performance in queries used inside the code (Java or VB) by simply changing names to absolute identifiers.

Example:

VB Script

```
root=getRoot
s=Timer
for i=1 to 10000
set res = root.getSelection("Select [Application] Where [Defined-
Service].[Operation].[Organizational Process]='World@Hand::BPMN Notation
Diagrams:: Purchasing:: Purchase Goods & Services:: Contract Negotiation'")
next
e=Timer
print "Query without IdAbs: " & (e-s)*1000 & "ms"
s=Timer
for i=1 to 10000
set res = root.getSelection("Select ~MrUiM9B5iyM0[Application] Where
~ltSTdNNHjqj0[Defined-Service] in (~TsUiT9B5iyQ0[IT Service] WHERE
~hqUiTCB5iK72[Operation].~mrUiaCB5iCB2[Organizational
Process]='~W3qoNsjV91e6[Contract Negotiation]')")
next
e=Timer
print "Query with IdAbs: " & (e-s)*1000 & "ms"
```

Java

```
Date dCurrDateS1 = new Date();
for (int j = 1; j <= 10000; j++) {
MegaCollection mgcolTest = mgRoot.getSelection("Select [Application] Where
[Defined-Service].[Operation].[Organizational Process]='World@Hand::BPMN
Notation Diagrams:: Purchasing:: Purchase Goods & Services:: Contract
Negotiation'");
}
Date dCurrDateS2 = new Date();
mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", "Query without IdAbs: " +
(dCurrDateS2.getTime() - dCurrDateS1.getTime()) + "ms");
```

```
dCurrDateS1 = new Date();

for (int j = 1; j <= 10000; j++) {

MegaCollection mgcolTest = mgRoot.getSelection("Select
~MrUiM9B5iyM0[Application] Where ~ltSTdNNHjqj0[Defined-Service] in
(~TsUiT9B5iyQ0[IT Service] WHERE
~hqUiTCB5iK72[Operation].~mrUiaCB5iCB2[Organizational
Process]='~W3qoNsjV91e6[Contract Negotiation]')");

}

dCurrDateS2 = new Date();

mgRoot.callFunction("~U7afnoxbAPw0[MessageBox]", "Query with IdAbs: " +
(dCurrDateS2.getTime() - dCurrDateS1.getTime()) + "ms");
```

The environment is Demonstration with MEGA 2009 SP5 CP6 R6.

We execute (a lot of times, 10000 times) the same query and then we will print the elapsed time.

The first query uses the names for MetaClass, MetaAssociationEnds, and the name for the source object.

The second query replace names with Absolute Identifiers (in the MegaField format).

The first cycle of queries took 20312,51 ms to be executed (about 20 seconds), the second cycles took about 16796,885 ms to be executed (a little bit more than 1 second and half).

By using idabs for the query we gain about the 25% in performances.

Remember (for the entry point) that MEGA accept a megaField when you can put a name.

To get the megaField of an object (its Id) just write `myObject.megaField` (you can use also `.megaUnnamedField`, in this case the name of the object will be replaced by a X, for the system it is the same).

In complex reports where you do a lot of queries the execution time of the report can benefit from queries with Identifiers.

This approach can also be implemented for imbedded queries in HTML descriptions for websites.

Not only it is interesting as far as peformance is concerned, it is also more maintainable (as written text for names of MetaClasses, MetaAssociationEnds and MetaAttributes is not stable in the long term).

11.1.3 Browsing repository (collection use)

There is no other way than browsing through the repository the information you want to handle in your code.

However, be aware that browsing the repository is time consuming (e.g.: finding objects, listing related objects).

You must pay particular attention to write optimized code.

Do not ask several times the same information to the repository.

Do not ask an information you do not need to the repository.

Take advantage of indexes in your browsings.

Here are examples of good vs bad :

Example 1:

- Bad: count is called as many time as there are elements

```
for (int i = 1; i <= mgobj.getCollection(…).count(); i++) {
  ...
}
```

- Good:

```
int iCount = mgobj.getCollection(…).count();
for (int i = 1; i <= iCount; i++) {
  ...
}
```

Example 2:

mgidSearchedId is the object identifier that you find via a MetaAssociationEnd from another object

Bad: browsing all the items of a list when you are interested in only one item and you know how to identify it by an indexed attribute

```
int iCount = mgobj.getCollection(…).count();
for (int i = 1; i <= iCount; i++) {
  MegaObject mgobjChild = mgobj.getCollection(…).get(i);
  if (mgobjChild.sameId(mgidSearchedId)) {
    mgobjSearched = mgobjChild;
    // my code
  }
}
```

Good:

```
MegaObject mgobjSearched = mgobj.getCollection(…).get(mgidSearchedId);
if (mgobjSearched.exists()) {
  // my code
}
```

Example 3:

You have an identifier in Hexadecimal format (Base 16) and you want it in Base 64 (or any other variant enabling to retrieve the identifier in any other format)

Bad: searching for an object to retrieve information you already have in another format.

```
strId = mgRoot.getObjectFromId(strHexaIdentifier).getProp(
"~310000000D00[Absolute Identifier]");
```

- Good:

```
strId = mgToolKit.getString64FromId(strHexaIdentifier);
```

11.1.4 Writing code rules

Writing code rules regarding GUIs

For web compatibility:

- do not use:
  - Java GUIs in MEGA code
  - VB (Visual Basic) GUIs in MEGA code
- use HOPEX forms to execute your own GUIs, with:
  - Property pages
  - Wizards

For detailed information on Property Pages and Wizards, see "HOPEX Forms" Technical Article.

Web compatibility: using Java or VB (Visual Basic) GUIs in MEGA code would launch the GUIs on MEGA server instead of the Web client and block MEGA server.

Writing code rules regarding Performance

Calculated MetaAttributes:

- browsing an excessive number of objects is forbidden
- do not over consume resources

Release objects:

See Processes going slower (Tracking down non released instances) p. 219.

- Java: release objects instead of using the default garbage collector, enter:

```
yObj.release()
```

- VB Script:

`Set myObj = Nothing` only if the process may take a long time.

Do not use GetObjectFromID, instead use:

```
myRoot.GetCollection(«  ~MrUiM9B5iyM0[Application]  »).Item(AppID)
```

Use of field with IdAbs is mandatory:

See MegaFields p. 53, MegaFields p. 170, and Handling Identifiers 199)

- the field format is: absolute identifier followed by the object name:

```
~MEsJ0p5rATw0[AllStoppedWorkflows]
```

- to retrieve the field form an object, enter:

VB Script

```
sID = myObject.megafield
```

Java `sID = myObject. megaField();`

When the object name is not required, to improve performance, use megaUnnamedField:

VB Script `sID = myObject. megaUnnamedField`

Java `sID = myObject.megaUnnamedField();`

Use variables to store objects

11.1.5 Confidentiality

Objects in MEGA may be confidential.

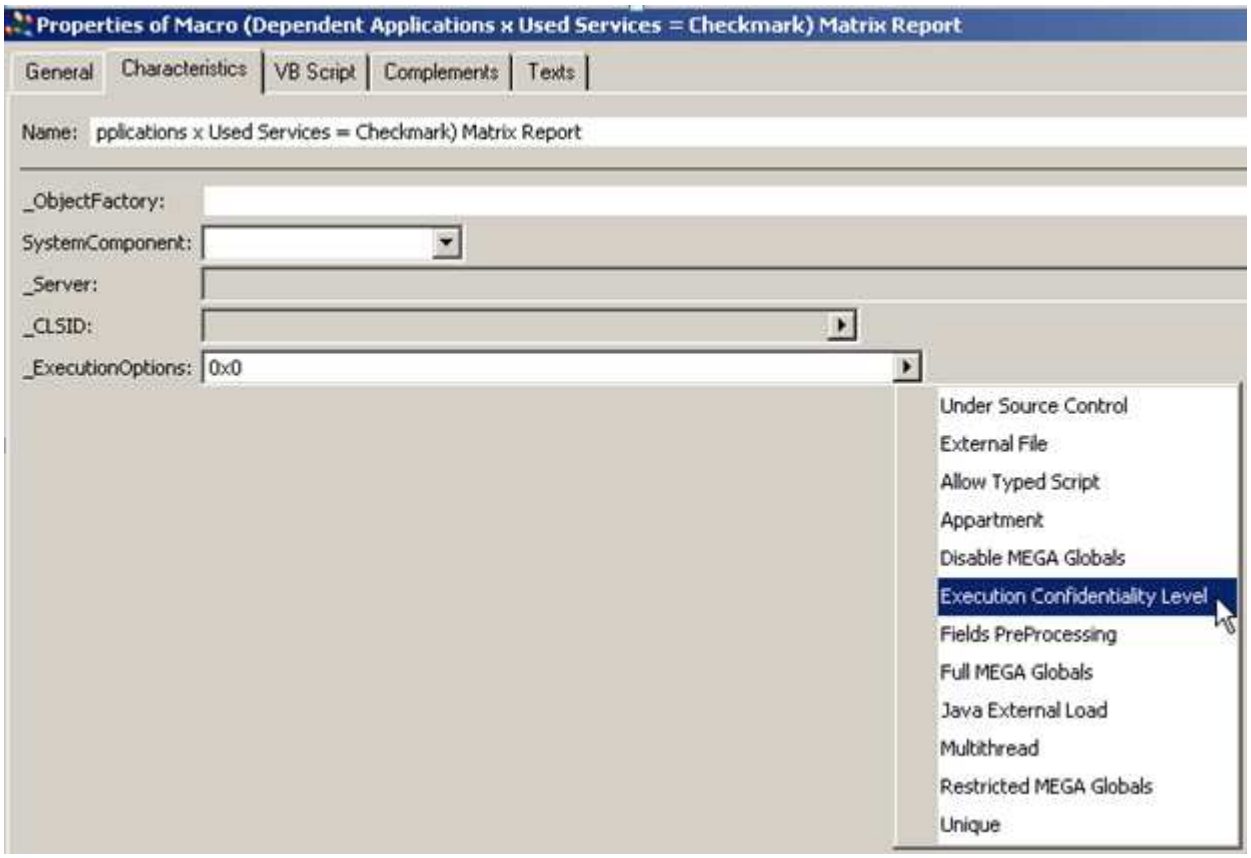
A macro does not give you acces to confidential objetcs:

`GetProp(MyProp , « display »)` sends back ****

`GetCollection` sends back a collection with the confidential objects but you cannot access to any properties

According to the macro parameterization, scripts are executed at:

- the macro confidentiality level : algorithm
- the user confidentiality level: displayed information



When creating a macro, always keep in mind the confidentiality issue.

The macro must give a valid result.

Examples:

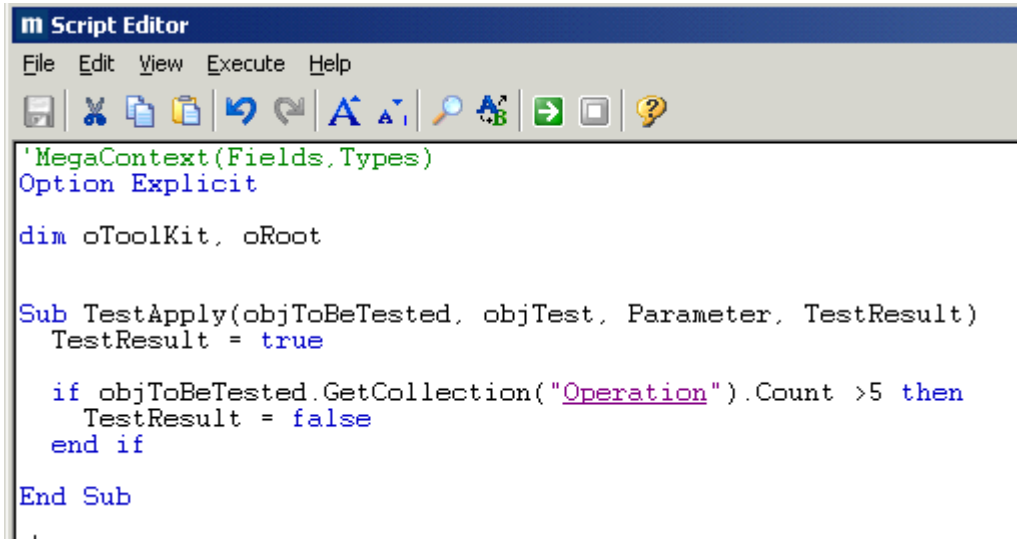
- **Regulation rule**: the organizational process cannot include more than five operations
- **Properties**: display the name and comment of an organizational process operations
- **Matrix**: displays the relations between the organizational processes and IT Services through operations

Example1: Regulation rule

Regulation rule: the organizational process cannot include more than five operations

For example:

- The organizational process P1 includes six operations: Op1, … , Op6.
- User U1 can see P1 but Op3 is confidential
- User U2 can see P1 and all of its operations
- Script:



When the above script is executed with:

- U1 view, the procedure execution result gives:

  `5 ≤ 5 -> TestResult = true`

  The regulation rule is valid and displays that the organisational process respect the rule.

- U2 view, the procedure execution result gives:

  `6 > 5 -> TestResult = false`

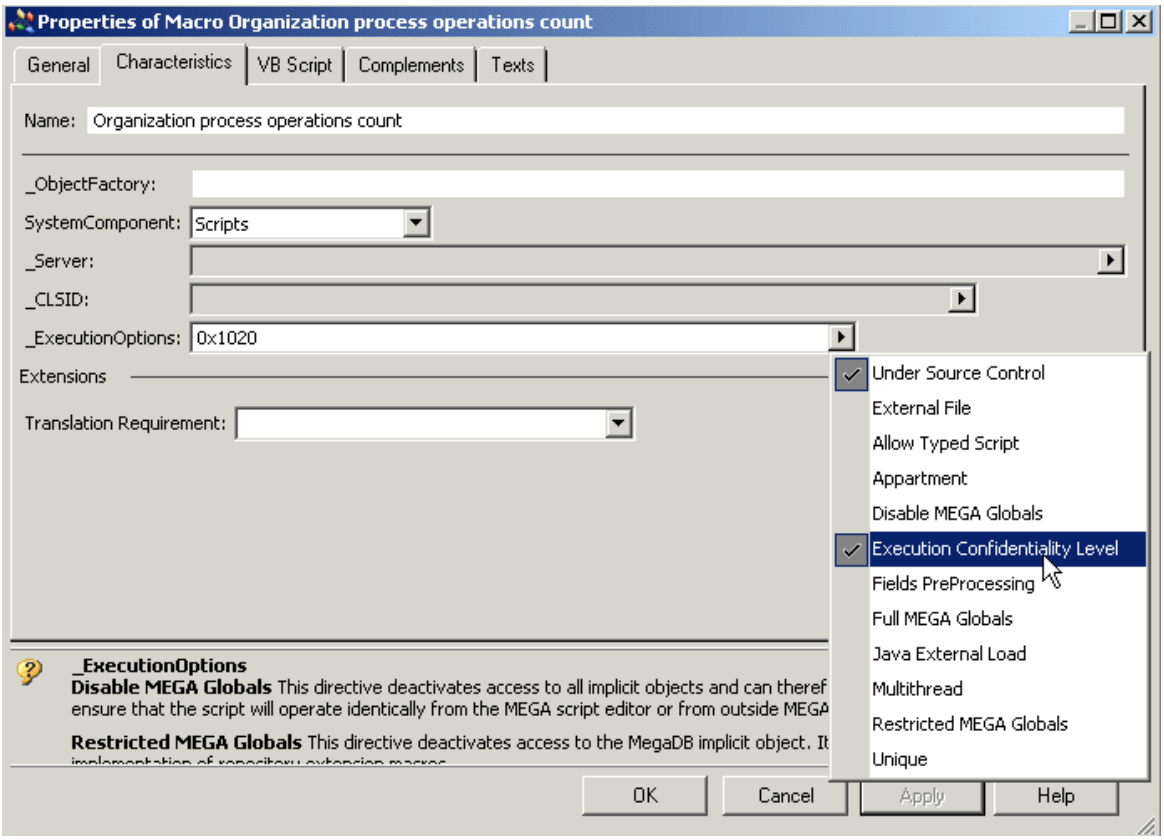
  The regulation rule is not valid and displays that the organisational process does not respect the rule.

➔ Results are different for user U1 and user U2

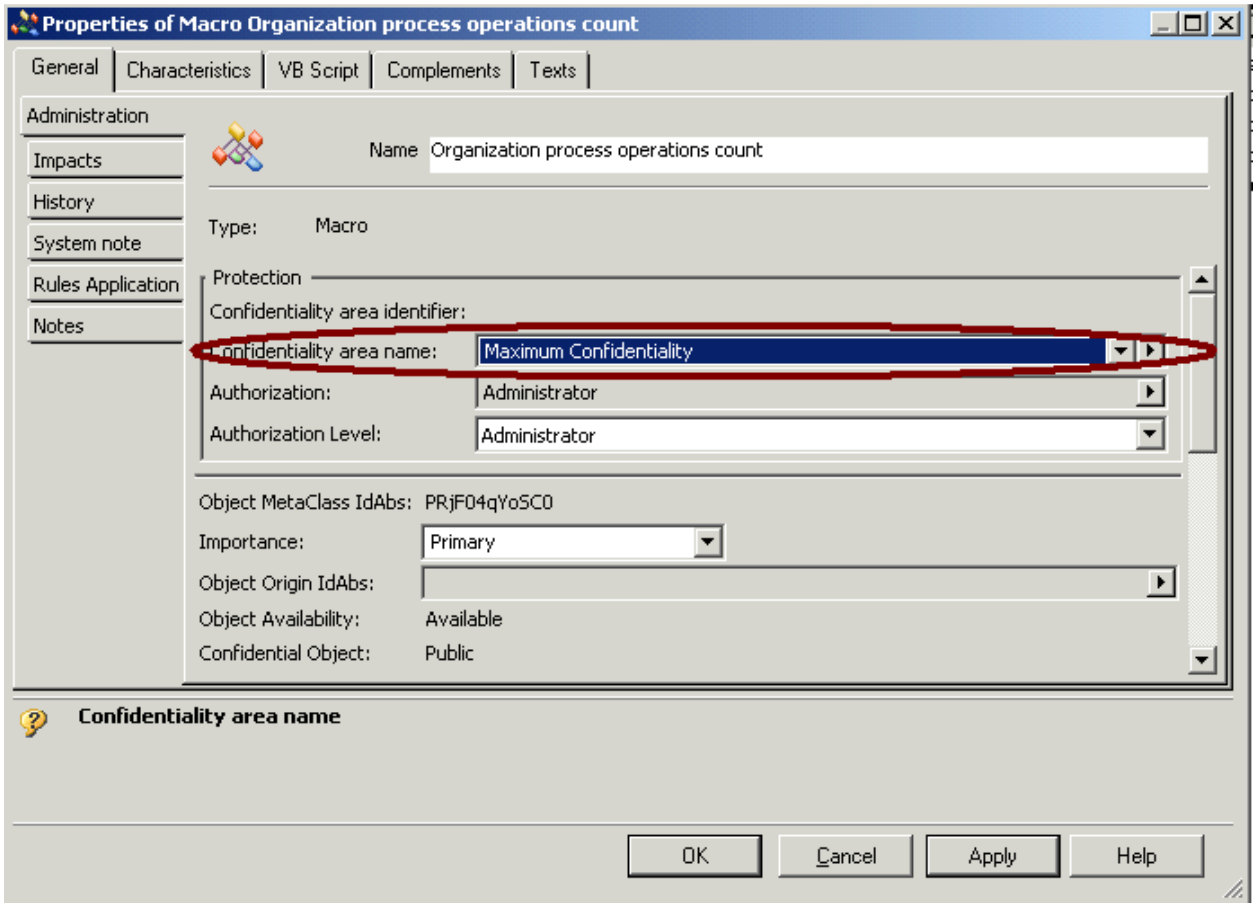
This script is not correct: the rule must give the same result for all the users.

To configure the macro related to the organizational process

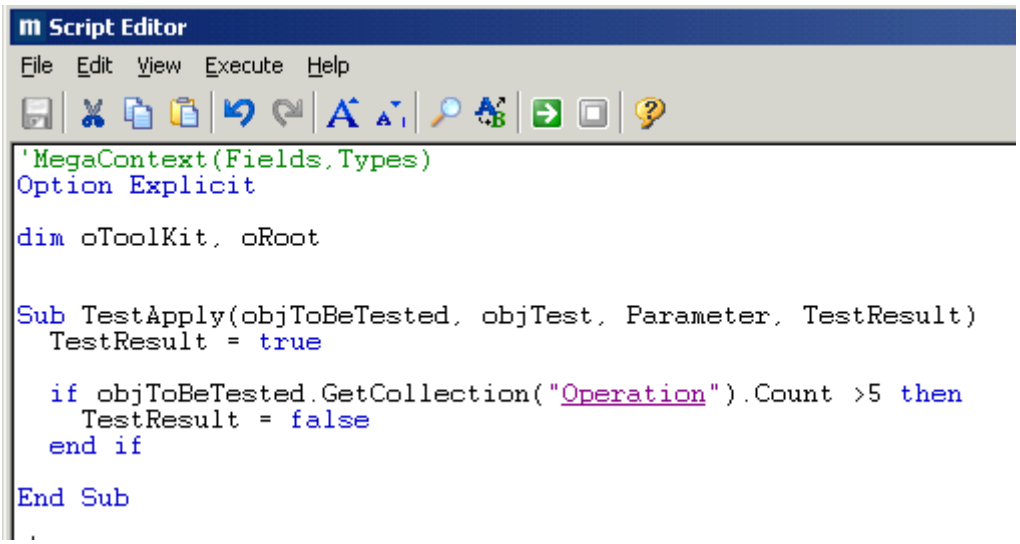
1. In the macro **Properties** window, **Characteristics** tab, set **_ExecutionOptions** to "Execution Confidentiality Level".



2. In the macro **Properties** window, **General** tab, **Administration** sub-tab, set the macro Confidentiality level to "Maximum confidentiality" level



3. Enter the code:



The code is executed as if the end user had the maximum confidentiality level.

When the above script is executed with U1 view, the procedure execution result gives:

`6 > 5 -> TestResult = false`

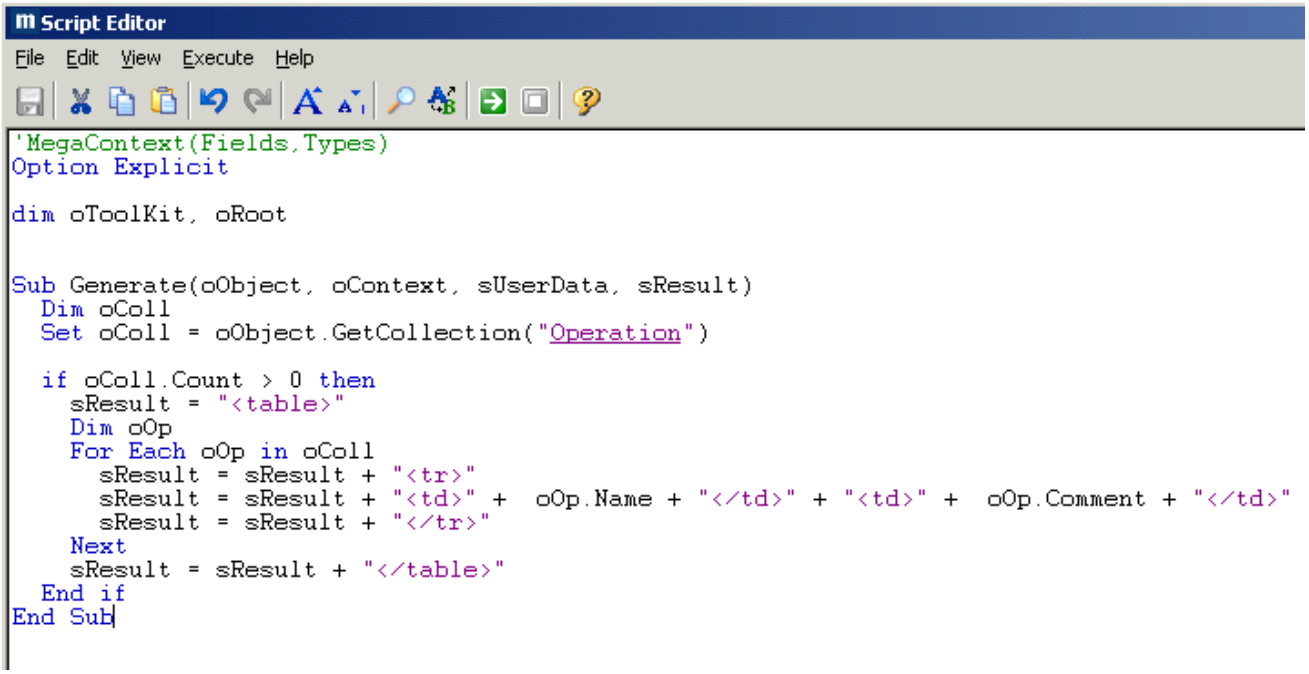
➔ For both users, the regulation rule fails and displays that the organisational process does not respect the rule.

Example2: Properties

When no algoritm is needed, execute the macro at the user confidentiality level.

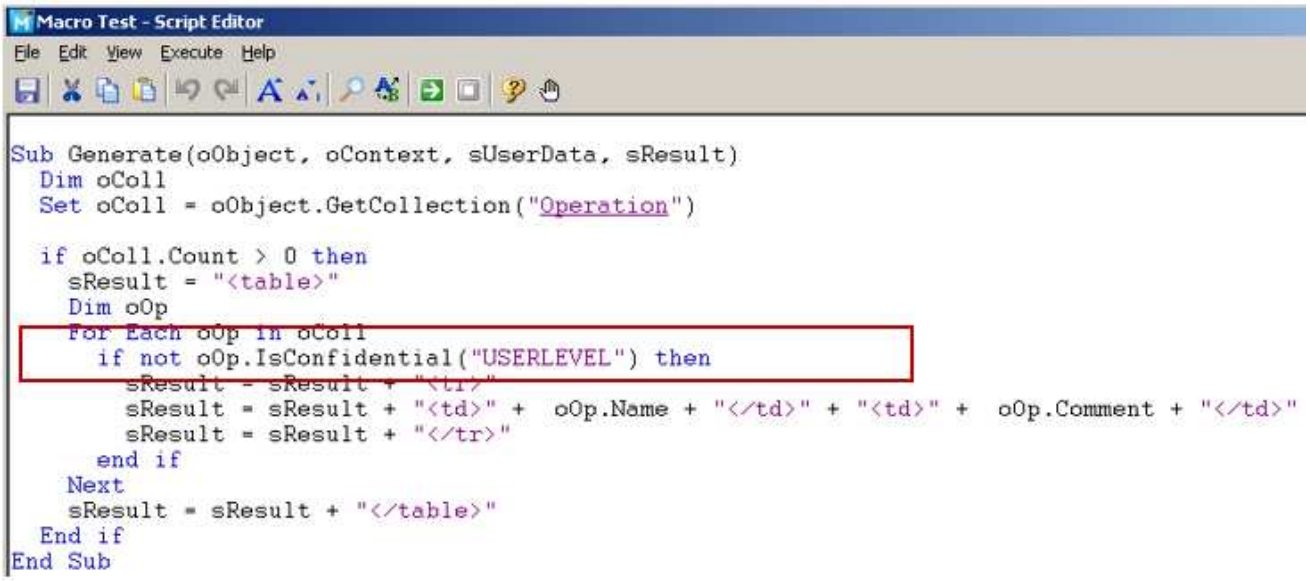
To display the name and comment of an organizational process operations

1. In the Script Editor, enter the following code:



When the code is executed at the maximum confidentiality level, with user U1, the result is wrong as Op3 is confidential for U1.

2. For the confidential objects not to be displayed use IsConfidential("UserLevel").



The result is good, Op3 is not displayed:

P1::Op2

P1::Op5

P1::Op1

P1::Op4

P1::Op6

Example2: Matrix

When algorithm is required, execute the macro at the maximum confidentiality level and take into account the confidentiality with the IsConfidential("USERLEVEL") function.

All the Operations have the operation type "Decision".

The purpose is to display the IT Services linked to the organizational process through the operations of Decision type.

1. In the Script Editor, enter the following code:

```
Dim oColl
Set oColl = oObject.GetCollection("Operation")

if oColl.Count > 0 then
  sResult = "<table>"
  Dim oOp
  For Each oOp in oColl
    if oOp.GetProp("Operation Type")="D" then
      Dim oService
        For Each oService in oOp.GetCollection("IT Service")
          if not oService.IsConfidential("USERLEVEL") then
            sResult = sResult + "<tr>"
            sResult = sResult + "<td>" +  oService.Name + "</td>" + "<td>" +  oService.Comment + "</td>"
            sResult = sResult + "</tr>"
          end if
        next
    end if
  Next
  sResult = sResult + "</table>"
End if
```

2. When the code is executed:
    - at the User level, with the user U1

      IT Service-2

      IT Service-5

      IT Service-1

      IT Service-4

      IT Service-6

      The result is wrong.

      Even if Op3 is confidential, the "IT Service-3" IT service linked to P1 should be displayed.

    - at the Maximum confidentiality level, with the user U1,

      IT Service-2

      IT Service-5

      IT Service-1

      IT Service-4

      IT Service-3

      IT Service-6

      The result is correct. Even if Op3 is confidential, the IT service "IT Service – 3" linked to P1 is displayed.

11.2 Performances

This section gives some advices to improve code performance. For these recommendations to be useful the code algorithmic needs to be well written. These recommendations are mainly useful for code handling a large amount of data. They concern:

- access to Mega data
- language code

The way the code access data can determine the overall performance of the process. Each query to the database costs in term of performance so it is better to do the minimal queries or to use cache systems.

11.2.1 Navigating through the metamodel with APIs

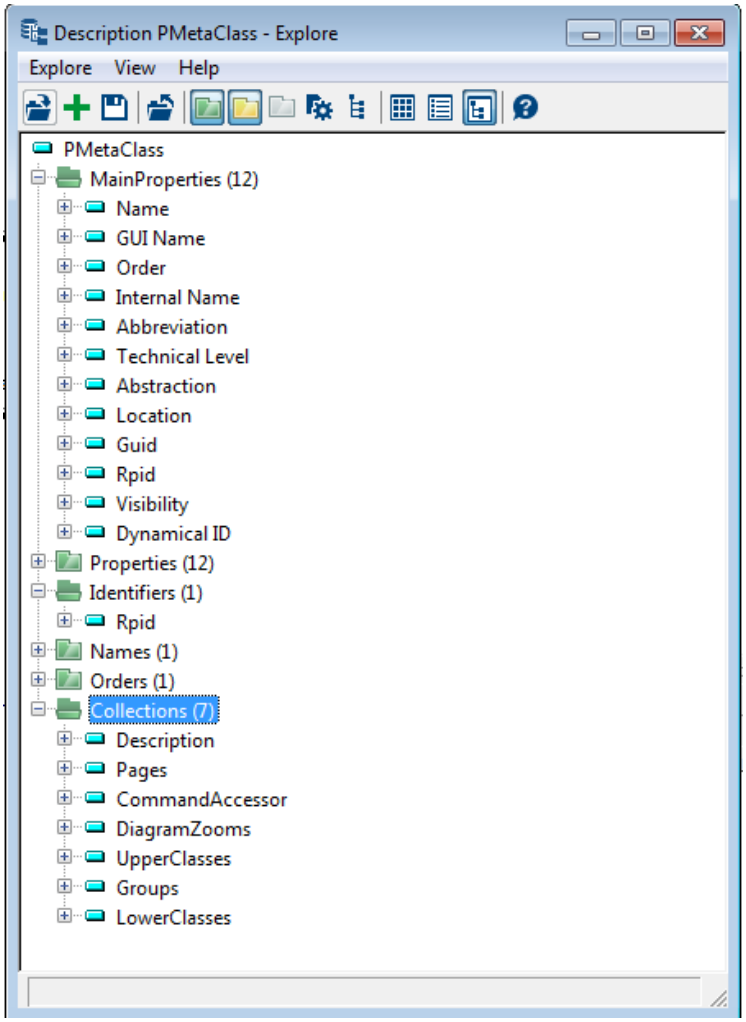
If you need to access to metamodel data (MetaClasses, MetaAttributes, MetaAssociations,…), you must know that this kind of data is rather static and Mega offers a metamodel cache to improve navigation performance. Specific APIs are available to access the metamodel, see Accessing the metamodel description using APIs p. 95.

When navigating through the metamodel, use these APIs (instead of standard APIs) to avoid access to repository and improve application performance.

To:

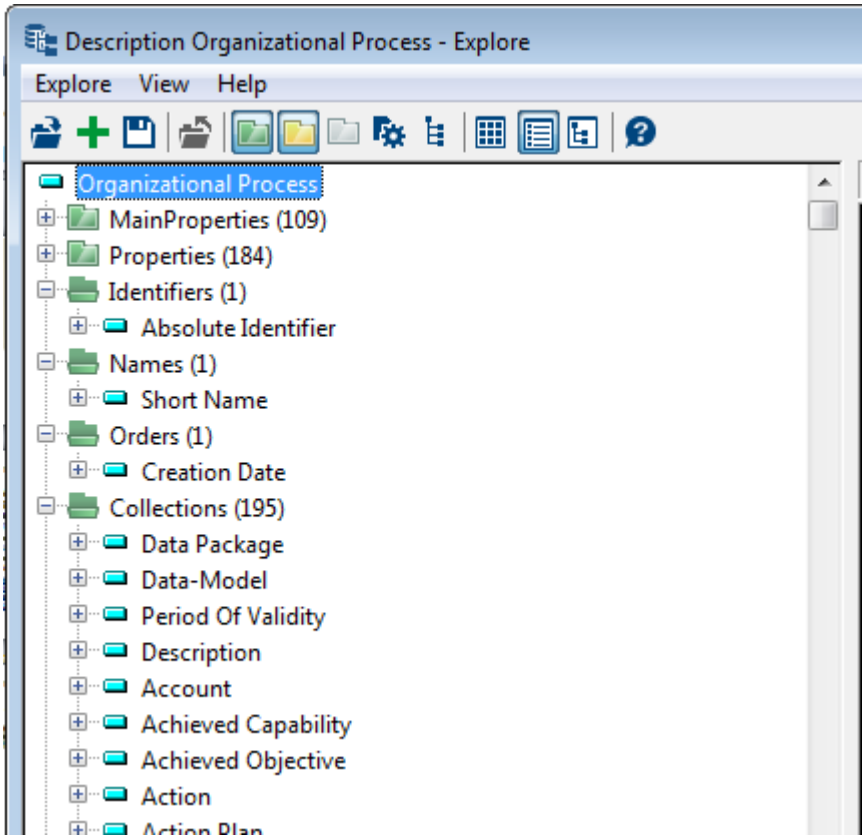
- get a different view on the metamodel, use:

```
getroot.GetClassDescription("~gsUiU9B5iiR0[Organizational
Process]").GetTypeObject().Explore
```



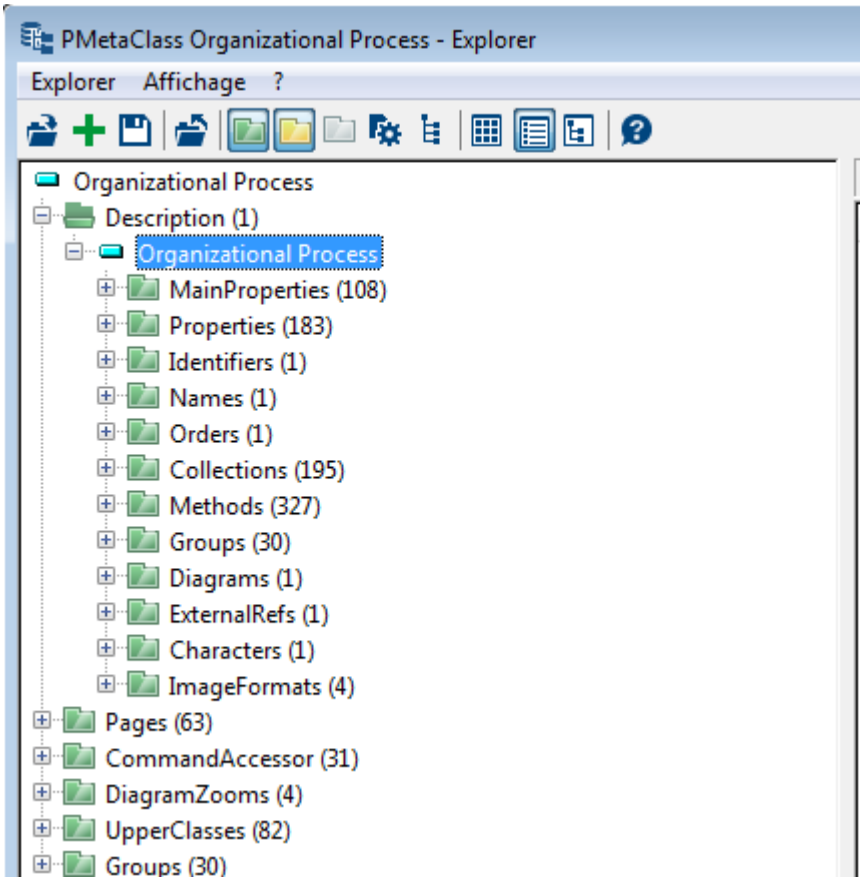
- access the Organizational Process MataClass, enter:

```
getroot.GetCollection(("~gsUiU9B5iiR0[Organizational
Process]").GetTypeObject.Explore
```



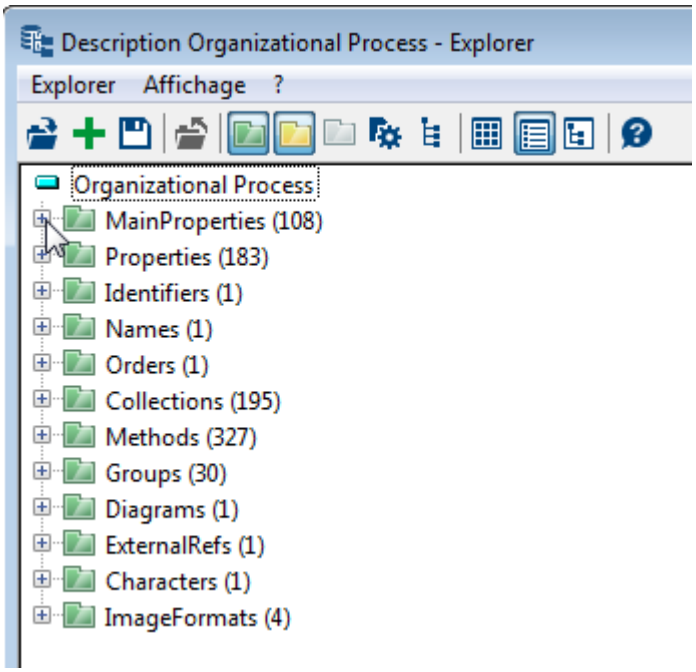
- get a full description of the MetaClass, use **GetClassdescription**:

```
getroot.GetClassDescription("~gsUiU9B5iiR0[Organizational Process]").Explore
```



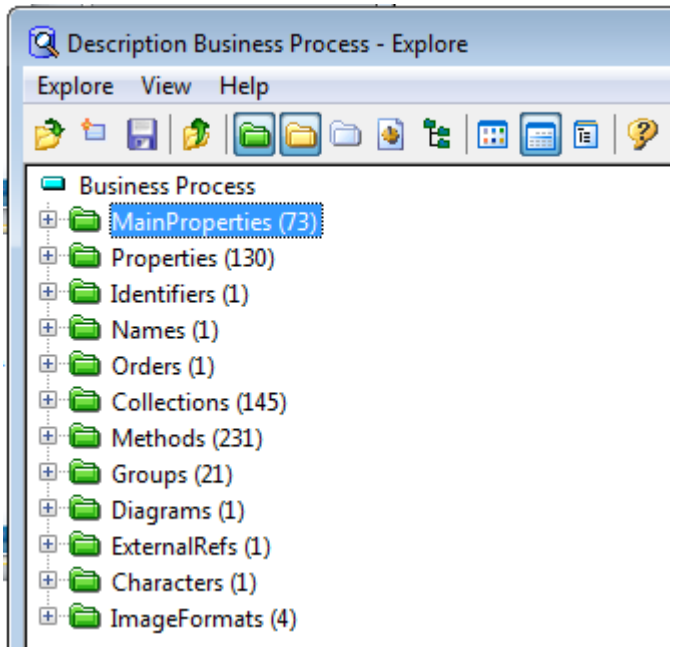
- get a description of the MetaClass only, use GetCollectionDescription

```
getroot.GetCollectionDescription("~gsUiU9B5iiR0[Organizational Process]").
Explore
```

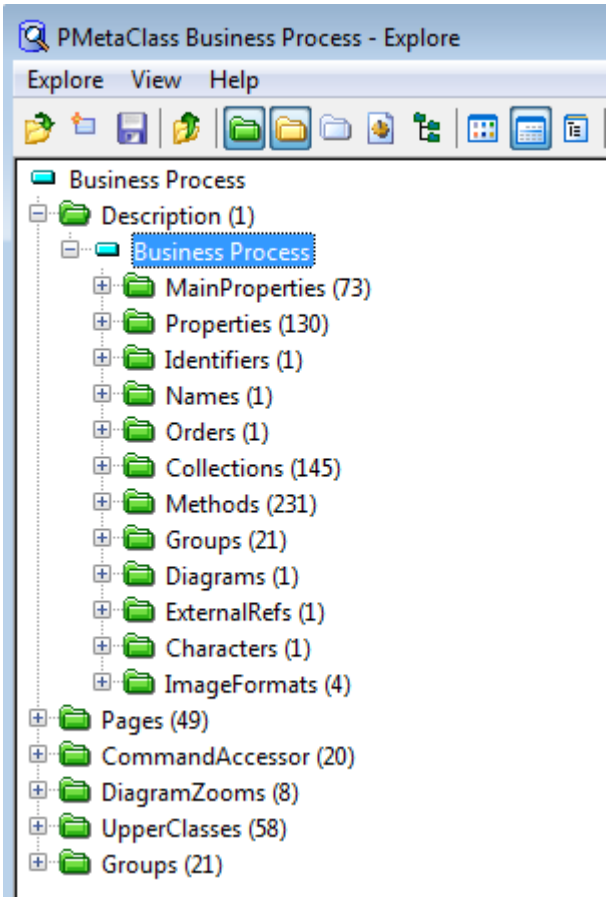


- navigate from an occurrence, use **GetTypeObject** or **GetClassObject**:

```
GetRoot.GetObjectFromID("~TK0gySBDCjl0[1. Training
Program]").GetTypeObject.explore
```




```
GetRoot.GetObjectFromID("~TK0gySBDCjl0[1.  Training
Program]").GetClassObject.explore
```



VB Script

```
for each oUpperClass in oRoot.getClassDescription(oMetaClass.GetId).UpperClasses
    set oUpperClass = oRoot.getObjectFromId(oUpperClass.GetId)
    for each omoAnalysisParameter in oUpperClass.Getcollection
("~8yHXTTuE31W0[Report Parameter]")
      set ocAnalysisTemplate.Insert = omoAnalysisParameter.Getcollection
("~kyHXAOuE3jN0[Parameterized Report Template]")
    next
  next
```

Java

```
MegaObject odMetaClass = oRoot.getClassDescription("~" + sClassId);
MegaObject odCollDescrip = odMetaClass.getCollection("Description").get(1);
```

11.2.2 Navigating through data with APIs

This case is the most often encountered.

Megafield usage

A megafield is a string containing the absolute identifier of an object followed by its name. It identifies a unique object. The name between the brackets is optional and can be replaced for example by "[X]".

Example of megafield:

`~MEsJ0p5rATw0[MyApplication]` is the same as `~MEsJ0p5rATw0[X]`

The megafield can be built manually or retrieved using the api megafield available on MegaObjects:

VB Script	Java
id = mgObject.MegaField	id = mgObject.megaField();
Returns ~MEsJ0p5rATw0[MyApplication]	
id = mgObject.MegaField()	id = mgObject.megaUnnamedField();
Returns ~MEsJ0p5rATw0[X]	

If you do not need the name, choose always the second one which improves the performance (it does not compute the name of the object).

The usage of megafields is compulsory for all Mega APIs.

Example:

```
mgObject.getProp("~310000000D00[Absolute Identifier]")
```

is better than

```
mgObject.getProp("Absolute Identifier").
```

This is the same for all Mega APIs (getProp, getCollection, etc.).

Searching an object

It is not recommended to use GetObjectFromId function. If you know exactly the MetaClass which instantiates what you are looking for, prefer writing this:

```
getCollection("~MrUiM9B5iyM0[Application]").item("~MEsJ0p5rATw0[MyApplication]")
;
```

When searching for an instance with its name, do not use a query with getSelection API, but the following code which uses dedicated indexes:

```
getCollection("~MrUiM9B5iyM0[Application]").item("MyApplication");
```

Manipulating identifiers

To compare the identifiers of two objects do not convert them in the same format and then compare them. Prefer the usage of the sameID API. With this one you can use whatever identifier (absolute identifier, hexa identifier or getId). It is available on MegaObject and it can be used as follows:

```
mgObject.sameID("~MEsJ0p5rATw0[MyApplication]");
```

If you have two identifiers, you can use the sameID API available on the MegaToolkit:

```
mgToolkit.sameID("MEsJ0p5rATw0","B1EDB2562C14016F");
```

If you have an identifier in a given format and you need it in anther format, do not go to the MegaObject and ask for the good identifier format. Instead use the APIs available on the MegaToolKit that allow converting identifiers:

```
getString64FromID(myID);
getStringFromID(myID);
getIDFromString(myStringID);
generateID();
```

Loops

In almost all the algorithms we can find loops that allow navigating through objects. If a code like this is needed:

```
for (int i = 1; i <= mgobj.getCollection(…).count(); i++) {
  ...
}
```

or

```
while (i <= mgobj.getCollection(…).count()) {
  i++;
  ...
}
```

Be careful to store the count of the collection before using it in the loop. Otherwise the count function will be evaluated at each loop.

Prefer syntax like this:

```
int iCount = mgobj.getCollection(…).count();
for (int i = 1; i <= iCount; i++) {
```

11.2.3 Processes going slower (Tracking down non released instances)

Some behaviors concern codes dealing with many object instances. The process seems to be slower as it progresses. In most cases, it is due to a large amount of living mega objects.

As a reminder, the following code:

```
MegaOject mgObj = mgRoot.getObjectFromId("xxxxxxxxxx");
```

instantiates a MegaObject class and connect it to the Mega repository object. As long as the java instance is not destroyed, the reference to the Mega object still exists. If a lot of these references lives at the same time, they are notified each time a modification is made in the repository. So the more you have this kind of references in memory the more slowly the process goes.

To solve the problem, release explicitly the instances as soon as you do not need them anymore.

To release an instance, enter:

VB Script
```
Set mgObject = Nothing
```

Java
```
mgObj.release();
```

This problem concerns MegaObjects or MegaCollections but also all Classes provided by Mega (MegaEnvironment, MegaRoot, MegaToolkit…).

Especially in Java, beware of unsuspected instantiations.

For example, the following code:

```
String codeTemplate = mgRoot.currentEnvironment().resources().codeTempla
te("xxxxxx", "");
```

instantiates two objets: one for the currentEnvironment and one for the resources.

These two objects should be released and you should replace the above code by the following one:

```
MegaCurrentEnvironment mgCurrEnv = mgRoot.currentEnvironment();
MegaResources mgRes = mgCurrEnv.resources();
String codeTemplate = mgRes.codeTemplate("xxxxxx", "");
mgRes.release();
mgCurrEnv.release();
```

Releasing all objects is really something important in big programs.

To help you to find the non-released objects in your java code, Mega provides a class called "**com.mega.modeling.api.util.MegaDebugLivingInstances**". It is located in mj_api.jar.

```
com.mega.modeling.api.util.MegaDebugLivingInstances
```

To use this class:

1. Indicate in your code, the moment from which you want to start monitoring the Mega Classes instances. To do that, enter:

```
MegaDebugLivingInstances.activate();
```

2. You can use three APIs:

```
MegaDebugLivingInstances.getLivingInstancesCount();
```

This function allows knowing the number of living instances. It retrieves an integer.

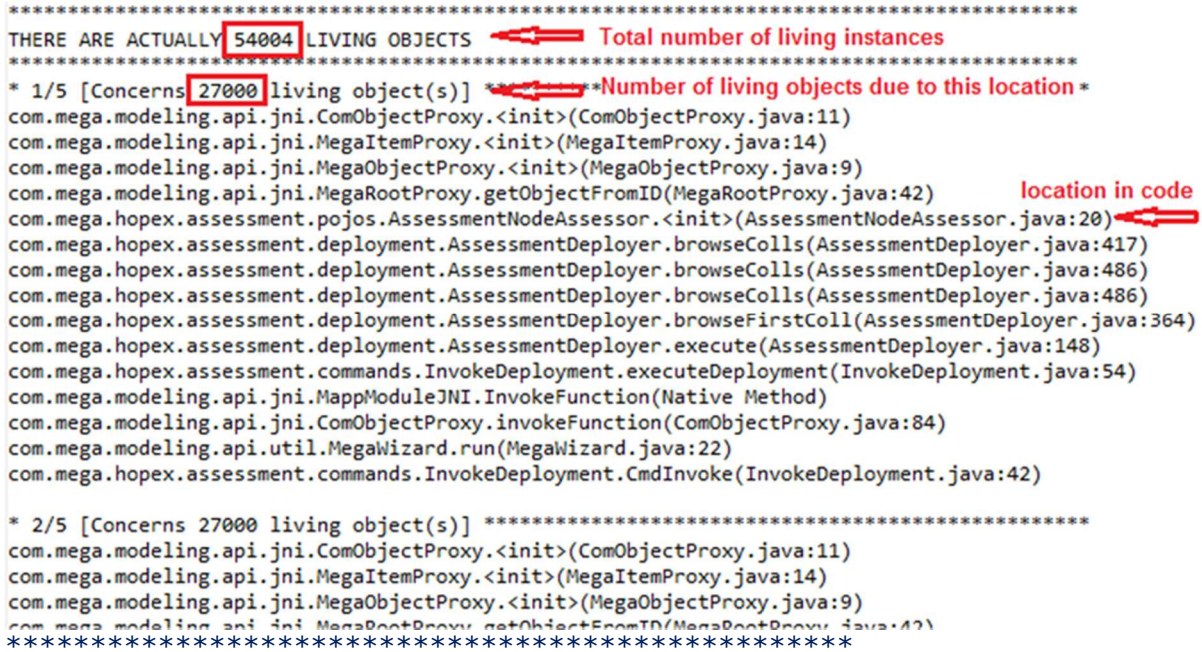
```
MegaDebugLivingInstances.getLivingInstances();
```

This function allows retrieving, as a string, the exact locations in your code and the number of instances still living due to this location.

```
MegaDebugLivingInstances.dumpLivingInstances("c:\\livingObjects.txt");
```

This function allows dumping, in a specified file, the exact locations in your code and the number of instances still living due to this location.

Dump example:



Using this class can slow down the main code so consider using it only during conception phase.

Batch mode

When entering a part of code which will handle a lot of data, it is recommended to disable some notifications for better performances. To do that, use the following API:

```
mgRoot.currentEnvironment().enterBatchUpdate();
```

To enable notifications use the following API:

```
mgRoot.currentEnvironment().leaveBatchUpdate();
```

Objects Creation

If you know the name of the occurrence you want to create, specify it as a parameter of the create API:

```
getCollection(…).create("My Name");
```

This prevents the system to allocate a temporary computed name to the object. This computation can be a cause of slow performance. This is especially useful when creating a lot of objects

If the objects you are creating are supposed to be linked to a main object, you should not have a code like this:

```
MegaObject myObject = getCollection(…).create();
myMainObject.getCollection(…).add myObject;
```

Prefer the following code which is optimized to do a create link operation:

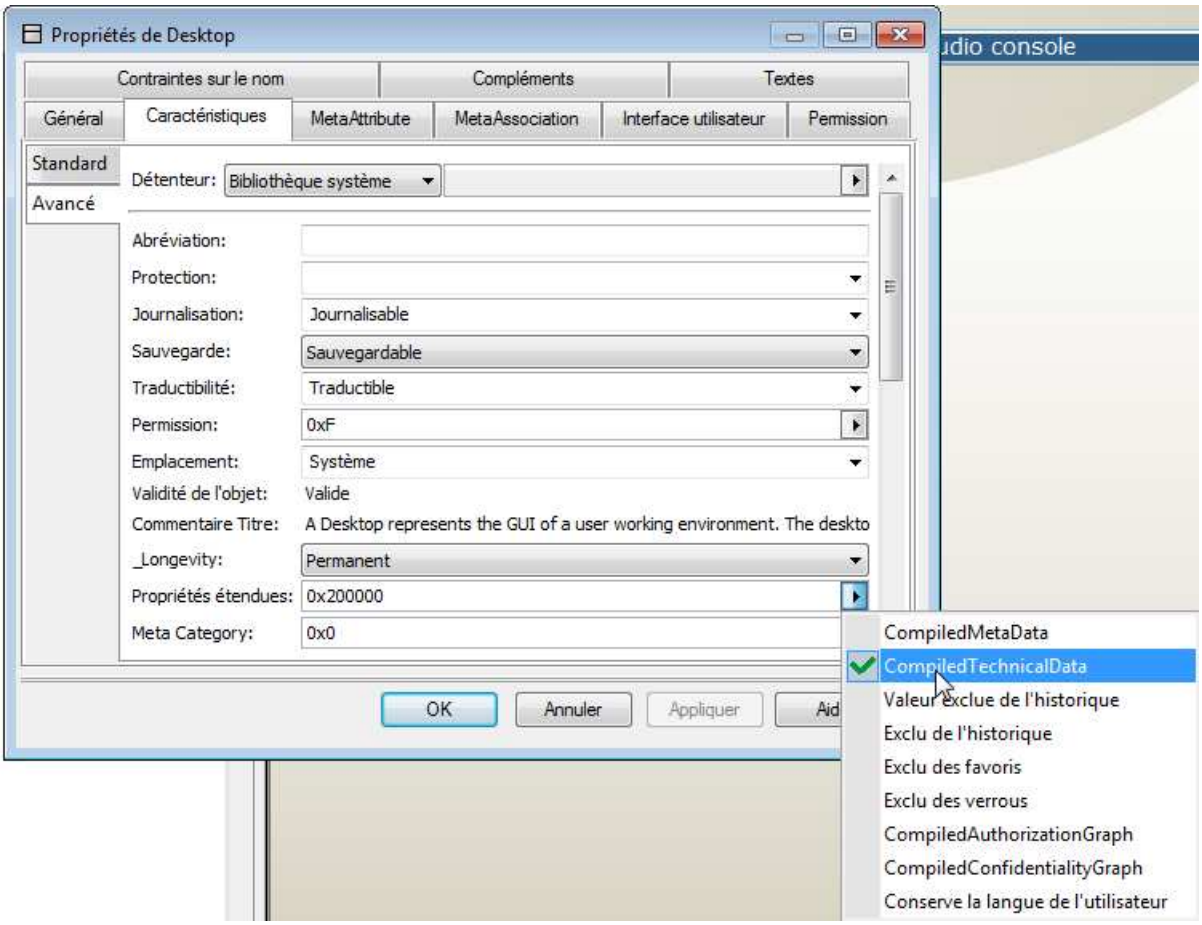
```
MegaObject myObject = myMainObject.getCollection(…).create();
```

This is really compulsory when dealing with concepts having namespace (most of the objects). These two technics can be composed to be more efficient.

11.2.4 Navigating through the technical data with APIs

Technical data is stored in the system repository.

- Data must be stable, that is to say:
  - serialized in a file
  - loaded on demand from the file
  - when a data often changes, there is no point in tagging it as “technical data”



- To navigate through the data, use the scanner as follows:

See Managing scanners p. 111.

  - Define a scanner class
  - Define public member that can be reused by the caller
  - Caller uses ScanCollection to retrieve a collection from a specific MetaAssociationEnd
  - Callee defines **onItem** procedure called for each data available in the collection

VB Script

```
Class ScannerReportTemplate
  Public mgResource
  Public mgToolkit
  Public mgData
  Public mgFunction
  Public mgName
  Public mgMetaclassidabs
  Public mgbTrouve
  Public mgIdAttribute
  Public mgFirstIdAbs
  Public  coont

  Public Sub OnItem(Context, Id)
    mgFirstIdAbs=Id
    mgbTrouve =true
    Select Case mgFunction
      Case 1
           mgData=Context.targetProperty(mgIdAttribute & ":T" )
      Case 2
           mgData = mgResource.name(Context, "~Z20000000D60[Nom court]")
    End Select
    Context.Abort
  End Sub
End Class
Function GetShortNameRT(idDepart,mgRoot As MegaRoot)
  Dim mgScanner
  Set mgScanner = New ScannerReportTemplate
  Set mgScanner.mgResource = mgRoot.CurrentEnvironment.Resources
  Set mgScanner.mgToolkit = mgRoot.currentEnvironment().toolkit()
  mgScanner.mgFunction = 2
  dim idLink
  dim listAttributes
  idLink = "~kyHXAOuE3jN0[Rapport type parametre]"
  ‘ T means the information is on the target object and not on the link
```

```
  listAttributes ="~Z20000000D60[Nom court]" & ":T"
  mgScanner.mgbTrouve = false
  mgScanner.mgResource.ScanCollection idDepart ,idLink , mgScanner ,1 ,
listAttributes
  if(mgScanner.mgbTrouve ) then
      GetShortNameRT= mgScanner.mgData
  else
      GetShortNameRT= null
  end if
 end function
```

Java

```
public class MyScannerEvent extends MyScanner implements CollectionScanner {
   private List<Event> m_lEvents = Collections.synchronizedList(new
ArrayList<Event>());
   public MyScannerEvent(final MegaRoot megaRoot, final MegaResources
megaResources, final MegaToolkit megaToolkit) {
    super(megaRoot, megaResources, megaToolkit);
   }
  public void OnItem(final MegaCollectionScannerContext context, final Object
endId) {
      Event event = new Event();
      event.setMacroId(Util.getIdAbsBase64(this.getMegaToolkit(), endId));
      event.setEvent(context.property(VocLinkDesktopLinkMacro.MA_EventType));
      synchronized (this.m_lEvents) {
      this.m_lEvents.add(event);
    }
    public List<Event> getListEvents() {
      return this.m_lEvents;
    }
    public void setListEvents(final List<Event> listEvents) {
      this.m_lEvents = listEvents;
    }
}
'Caller
MyScannerEvent myScannerEvent = new MyScannerEvent(this.getMegaRoot(),
this.getMegaResources(), this.getMegaToolkit());
ScannerProperties spEvent = new ScannerProperties();
PropertiesForObject.addPropertiesForDesktopEvents(spEvent);
this.getMegaResources().scanCollection(objDesktopID,
VocDesktop.MAE_EventBehaviorMacro, myScannerEvent, CollectionScanMode.synchrone,
spEvent.getAllProperties());
desktop.setEvents(myScannerEvent.getListEvents());
```

11.3 Log error management

In standard, MEGA logs all the errors in the megaerr file.

To get the stack written on the log file, you must not use try…catch in your code.

However, if you want to send back an unrecoverable error, use the MEGAException class.

```
public void myMethod(final String sParam) throws MegaException
{
   if (sParam.equals("bad"))
   { throw new MegaException("Bad param", Mode.APPLICATIF);
   }
```

JAVADOC

See the JavaDoc documentation:

- Reports API
- MEGA API
- Toolkit API
- Workflow Engine API