Document edition 2.0.1 (March 2013)

This edition applies to the “Workflow Manager version 2.0.0” and to all subsequent releases and modifications until otherwise indicated in new editions.
# Contents

## Chapter 1 Overview

### Chapter 2

2.1 Element Type

2.1.1 Container Element Type

2.2 Element List Structure

2.3 Filter Type

2.4 Property

2.5 Action Descriptor

2.6 Tool Descriptor

## Chapter 3 Data Model Details

3.1 Enabled Condition

3.2 Variable References

3.3 Application Options

3.3.1 Relationship: Application Options have Specific Attributes

3.4 Action Descriptor

3.4.1 Has_Element List

3.4.2 Related Save-/Close Action

3.4.3 Relationship: Action has Tool

3.4.3.1 Parameter Override

3.5 Element List Structure

3.5.1 Relationship: Element List has Action

3.5.2 Relationship: Element List has Column

3.5.3 Relationship: Element List has Property

3.6 Element Type

3.6.1 Relationship: Element has Action

3.6.2 Relationship: Element has Filter

3.6.3 Relationship: Element has Property

3.7 File Descriptor

3.7.1 Relationship: File Descriptor has Property

3.7.2 Relationship: File Descriptor has Action

3.8 Filter Type

3.8.1 Relationship: Filter has Action

3.8.2 Relationship: Filter has Filter Criterion
Contents

6.5.1 ISPF Table Display support................................................................. 93
6.5.2 Special processing for ISPF Edit or ISPF Browse................................. 94
6.5.3 ISPF Dialog Exits ............................................................................. 95

Chapter 7 Example AWM: PDS Explorer .................................................. 96
7.1 Concept .............................................................................................. 96
7.2 Creating the AWM .............................................................................. 97
7.2.1 Application Options ....................................................................... 97
7.2.2 Properties ....................................................................................... 98
7.2.3 File Descriptors ............................................................................ 99
7.2.4 Icons ............................................................................................. 101
7.2.5 Actions & Tools ............................................................................ 101
7.2.6 Container Element Types ............................................................... 105
7.2.6.1 Setting the Root Containers Action .......................................... 106
7.2.7 Element Types ............................................................................... 106
7.2.7.1 ID Definition ........................................................................... 107
7.2.8 Filter Type ..................................................................................... 108
7.2.9 Element List Structure ................................................................ 108
7.3 REXX Procedures ............................................................................. 109
7.3.1 Determining projects (TAUPDGPR) .............................................. 109
7.3.2 Determining Datasets with Properties (TAUPDGPD) .................... 110
7.3.3 Determining Datasets without Properties (TAUPDGPL) ............... 112
7.3.4 Exit for the creation of files (TAUPDXA1) ...................................... 112
7.3.5 Determining Member Lists (TAUPDELI) ........................................ 113
7.3.6 Determining Element Properties (TAUPDEPR) ............................. 114
7.4 Editing the Master Configuration ......................................................... 115

Chapter 8 Commissioning ......................................................................... 116
8.1 Starting for the first time .................................................................... 116
8.2 Further development and maintenance .............................................. 117
8.2.1 Maintaining the AWM .................................................................. 118
8.2.1.1 Changes to ID Definitions ....................................................... 120
List of Figures

Figure 1.1 The major data model objects and relationships (simplified) ......................................................... 3
Figure 2.1 Referencing of Properties .................................................................................................................. 8
Figure 2.2 Relationships of an Action to Tools .................................................................................................... 8
Figure 2.3 Tool Descriptor and its relationships .................................................................................................. 10
Figure 2.4 Attachment of a "Delete File" Tool ................................................................................................... 11
Figure 3.1 Example of a "Unique" attribute [U] .................................................................................................... 14
Figure 3.2 Modelling Action Enabled Condition ............................................................................................... 16
Figure 3.3 Creation of an Element List ............................................................................................................. 23
Figure 3.4 Element Properties in the Properties View ....................................................................................... 30
Figure 3.5 Link between File Content and Properties ...................................................................................... 33
Figure 3.6 Display names of Container Elements ............................................................................................ 42
Figure 3.7 Retrieve button in a Filter dialog ...................................................................................................... 44
Figure 3.8 Transfer of Parameters between tools .............................................................................................. 50
Figure 4.1 Closing the Welcome window .......................................................................................................... 66
Figure 4.2 Creating a new project ...................................................................................................................... 66
Figure 4.3 Opening the "Open Perspective" dialog ............................................................................................ 66
Figure 4.4 Newly created AWM in the Model Editor ......................................................................................... 67
Figure 4.5 Properties View for a selected Property .......................................................................................... 68
Figure 4.6 Dialog for specific attributes for the definition of several attribute values. .................................. 68
Figure 4.7 Creating a "Properties" category ....................................................................................................... 69
Figure 4.8 Help text for selected attributes ...................................................................................................... 69
Figure 4.9 Labelling in the Model Editor .......................................................................................................... 69
Figure 4.10 Checking the AWM for errors ........................................................................................................ 70
Figure 4.11 Drag & Drop ................................................................................................................................. 72
Figure 5.1 The zExplorer within the Remote Systems View .............................................................................. 73
Figure 6.1 An input dialog sample .................................................................................................................... 78
Figure 7.1 Modelling Properties ...................................................................................................................... 98
Figure 7.2 Modelling File Descriptors ............................................................................................................ 100
Figure 7.3 Modelling Tool Descriptors ........................................................................................................ 103
Figure 7.4 Modelling Action Descriptors ...................................................................................................... 104
Figure 7.5 Modelling two Container Element Types ....................................................................................... 105
Figure 7.6 Modelling Element Types ........................................................................................................... 106
Figure 7.7 Modelling ID Definitions .............................................................................................................. 107
Figure 7.8 Modelling Filter Types ................................................................................................................ 108
Figure 7.9 Modelling a Element List Structure .............................................................................................. 108
Figure 8.1 Changing to the Team Developer Perspective .............................................................................. 116
Figure 8.2 Creating a new System entry .......................................................................................................... 117
Figure 8.3 System entry and Applications ..................................................................................................... 117
List of Figures

Figure 8.4 Application cache ................................................................. 118
Figure 8.5 Determining the file path of the Cache ........................................ 120
Figure 8.6 Local Application Data after an ID Definition Update .................. 121
List of Tables

Table 1-1 Basic expressions used ........................................................................................................ 2
Table 2-1 Properties and Property Values of the “Text document” file type ........................................ 7
Table 3-1 Attribute symbols and their meanings .................................................................................... 13
Table 3-2 Attributes of an Action Enabled Condition ......................................................................... 15
Table 3-3 Attributes of a Simple Enabled Condition ........................................................................... 16
Table 3-4 Operators of a “Simple Enabled Condition” ....................................................................... 17
Table 3-5 List of all supported variables ............................................................................................. 18
Table 3-5 Attributes of the Application Options .................................................................................. 20
Table 3-6 Relationships of Application Options .................................................................................... 20
Table 3-7 Attributes of the Relationship “Application Options have Specific Attributes” ..................... 20
Table 3-8 Specific attributes supported ............................................................................................... 21
Table 3-9 Attributes of an Action Descriptor ....................................................................................... 22
Table 3-10 Relationships of an Action Descriptor ................................................................................ 22
Table 3-11 Attributes of the "Action has Tool" relationship ................................................................. 24
Table 3-12 Attributes of a Parameter Override ................................................................................... 24
Table 3-13 Attributes of an Element List Structure ............................................................................ 25
Table 3-14 Relationships of an Element List Structure ....................................................................... 25
Table 3-15 Attributes of the "Element List has Action" relationship .................................................. 26
Table 3-16 Attributes of the "Element List has Column" relationship ................................................ 26
Table 3-17 Attributes of the "Element List has Property" relationship ................................................. 27
Table 3-18 Attributes of an Element Type ............................................................................................ 27
Table 3-19 Relationships of an Element Type ....................................................................................... 28
Table 3-20 Attributes of the "Element has Action" relationship ........................................................... 29
Table 3-21 Attributes of the "Element has Filter" relationship .............................................................. 29
Table 3-22 Attributes of the "Element has Property" relationship ....................................................... 31
Table 3-23 Attributes of a File Descriptor ............................................................................................ 32
Table 3-24 Relationship of a File Descriptor ........................................................................................ 32
Table 3-25 Attributes of the "File Descriptor has Property" relationship .............................................. 33
Table 3-26 Attributes of a Filter Type .................................................................................................. 34
Table 3-27 Relationships of a Filter Type .............................................................................................. 34
Table 3-28 Attributes of the "Filter has Action" relationship .............................................................. 34
Table 3-29 Attributes of the "Filter has Filter Criterion" relationship ................................................ 35
Table 3-30 Attributes of a Global Action ............................................................................................... 36
Table 3-31 Attributes of the complex attribute "ISA Editor Action" .................................................... 37
Table 3-32 Attributes of the complex attribute "Selection Property Mapping" ..................................... 38
Table 3-33 Attributes of the complex attribute "ISA Workbench Action" ........................................... 38
Table 3-34 Predefined icons ............................................................................................................... 40
List of Tables

Table 3-35 Attributes of an Icon ................................................................. 40
Table 3-36 Attribute of an ID Definition ....................................................... 40
Table 3-37 Relationship of an ID Definition ................................................. 40
Table 3-38 Example of an ID Definitions ..................................................... 41
Table 3-39 Attributes of the "ID has Key Property" relationship............................ 41
Table 3-40 Attributes of a Property ............................................................... 45
Table 3-41 Possible Resource Operations of a Tool Descriptor ......................... 48
Table 3-42 Attributes of a Tool Descriptor .................................................... 49
Table 3-43 Relationships of a Tool Descriptor ................................................ 49
Table 3-44 Attributes of the "Tool has Input Parameter" relationship .................... 52
Table 3-45 Attributes of the "Tool has Output Parameter" relationship .................. 53
Table 3-46 Attributes of an ISA_Plugin .......................................................... 60
Table 3-47 Attributes of an ISA_ISPFTool .................................................... 60
Table 3-48 Attribute of the ISPF_Panel relationship ....................................... 61
Table 3-49 Attributes of an ISA_ISPFPanel Attribute ..................................... 62
Table 3-50 Attribute of the Has_Panel_Field relationship ............................... 62
Table 4-1 Key words of the Master Configuration ......................................... 65
Table 4-2 INFO entries and their effects ....................................................... 65
Table 4-3 Table of all important shortcut keys and mouse operations ................ 71
Table 5-1 List of fixed objects within the zExplorer model ............................... 75
Table 5-2 List of z/Server tools within the zExplorer model ............................. 76
Table 6-1 Effects of specific dialog attributes ................................................. 79
Table 6-2 Attributes of the "Tool has Input Parameter" relationship and their effects on the Input Dialogs ... 80
Table 6-3 Valid OS Command return statements ............................................ 81
Table 6-4 Supported Resource Operations in the ISPF Interface Table TAUTOTBL ........................................... 87
Table 6-5 ISPF Exits .......................................................................................... 89
Table 6-6 REXX Trace options ........................................................................ 91
Table 7-1 Configuration of the Application Options ....................................... 98
Table 7-2 Configuration of the most important properties ............................... 99
Table 7-3 Modelling File Descriptors ........................................................... 101
Table 7-4 Modelling Icons ............................................................................ 101
Table 7-5 All Action Descriptors of the PDS Explorer .................................... 102
Table 7-6 Modelling Tool Descriptors and Relationships ............................... 104
Table 7-7 Modelling Container Element Types and Relationships .................... 106
Table 7-8 Modelling Root Containers Action ................................................. 106
Table 7-9 Modelling the major Element Types and relationships .................... 107
Table 7-10 Modelling a Filter Type ............................................................... 108
Table 7-11 Modelling Element List Structure ................................................ 109
Table 8-1 Cache Files ..................................................................................... 119
List of Abbreviations

API. . . . . . . . . . . . Application Programming Interface
AWM. . . . . . . . . . . . Application Workflow Model
COBOL. . . . . . . . Common Business Oriented Language
GUI. . . . . . . . . . . . Graphical User Interface
IBM. . . . . . . . . . . . International Business Machines Corporation
ISPF. . . . . . . . . . . . Interactivity System Product Facility
OS. . . . . . . . . . . . . Operating System
PC. . . . . . . . . . . . . Personal Computer
SCLM. . . . . . . . . . . . Software Configuration and Library Manager
SCM. . . . . . . . . . . . Software Configuration Management
TSO. . . . . . . . . . . . . Time Sharing Option
UI. . . . . . . . . . . . . User Interface
XML. . . . . . . . . . . . Extensible Markup Language
z. . . . . . . . . . . . . . (for IBM products such as System z, z/OS,..) Zero Downtime
z/OS. . . . . . . . . . . . (Zero Downtime) Operating System
About this manual

Who should read this manual?

This manual is intended to serve as a reference work, as well as for creating workflow models and for the programming of tool attachment interfaces.

The manual has not been designed for users who have not yet worked with the application workflow data model. It is recommended that such people should attend a training course or a coaching session.

Preconditions

An installation of Eclipse Classic including Eclipse Target Management and the “Workflow Manager” is a precondition. Details about this can be found in the Installation Manuals.

Programming skills will be required in order to understand “Chapter 6 - Attaching Tools” and “Chapter 7.3 - REXX Procedures.

If the application modelling and attachment of the tools are carried out by different people, precise design matching will be necessary between the modelling work and the tool attachment.

Typographical conventions

Illustrations

Illustrations are numbered according to their chapter and are listed on the List of Illustrations. If there is no description of a figure in its caption, the relevant description will be found in the text. Captions are shown in italic script.

Highlighting

Expressions marked in bold symbolize the key words in paragraphs. In most cases, these expressions will be explained in more detail in the corresponding passage, and will appear several times in the manual. Examples are highlighted in italic script.

Lists

Two symbols are used for lists in this manual:

- Normal bullets (Level 1)
  - Subsidiary points (Level 2)

Numerical lists represent a sequence.
Structure of the manual

Chapter 1  Overview

Provides the reader with a general overview of the Workflow Manager.

Chapter 2  Introduction to the data model

Provides an overview of the data model. Expressions that appear frequently in the following chapters are explained in detail here.

Chapter 3  The AWM Data Model

Describes each model class of the Application Workflow Model in detail.

Chapter 4  Setting up the Application Workflow Model (AWM)

Explains the best way to create and edit the AWM using the AWM Editor.

Chapter 5  Customizing the zExplorer

Describes how to customize the zExplorer model.

Chapter 6  The attachment of tools to the AWM

Describes how to design the tool attachment for tool calls defined in the AWM.

Chapter 7  Example: PDS Explorer

Includes a complete example. This includes the development of the Host Tools, as well as the definition of the AWM.

Chapter 8  Commissioning

Contains methods and tips for the maintenance of AWM.
Chapter 1

Overview

The Workflow Manager supports the development of z/OS applications in an Eclipse based environment.

Over the years, the Mainframe development tools have frequently been increasingly adapted to the needs of the developer. Home-grown user interfaces, as well as complex Compile and Release procedures mean that the change to a modern development framework based on Eclipse becomes an extensive project.

With the Workflow Manager the effort integrating existing remote and local development tools in Eclipse and the future maintenance of the tool integration is considerably simplified. The Workflow Manager has been designed so that, where required, the existing development procedures and tools can be integrated in Eclipse with a minimum effort. The attachment is thereby based on a model-driven approach. With the help of an AWM Editor, an XML file will be edited by an administrator. During the runtime the AWM will be dynamically interpreted under Eclipse and that supports both the modelled user interface and the modelled tool attachment.

The Workflow Manager uses the following, very general expressions in order to be able to describe the tool integration as flexibly as possible:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>An Element usually describes an object that is managed in the tool that is to be integrated in the AWM. A file or a database record is usually connected to it, and there are tools processing the elements.</td>
</tr>
<tr>
<td></td>
<td>Example:</td>
</tr>
<tr>
<td></td>
<td>COBOL program TEST4711</td>
</tr>
<tr>
<td></td>
<td>Project: PRJ4711</td>
</tr>
<tr>
<td>Element list</td>
<td>An Element List is a list of elements with defined properties that is drawn up according to specific criteria. An Element List is presented in the form of a table.</td>
</tr>
<tr>
<td>Expression</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| Container  | A Container is a specific element which contains or references other elements, directly or indirectly. Containers can also include other containers. The Container structure is presented in the form of a tree view.  
Example:  
PDS file: USER01.TEST.COBOL (contains PDS members)  
Project: PRJ4711 (contains applications)  
Order: CHG4711 (contains sources connected to this change record) |
| Filter     | A Filter describes criteria from which a list of elements can be determined.  
Example:  
All PDS members in USER01.TEST.COBOL, whose names begin with TEST. |
| Action     | An Action is an integrally closed function that can be triggered in the user interface, e.g. a function in the context menu of an element or a container. An Action can be made up of several working steps (tools), e.g. an input dialog, processing on the mainframe, any subsequent processing on the client and an output dialog.  
Example: Compile (in the context menu for Element TEST4711) |
| Tool       | A Tool is the smallest processing step that can be described in the AWM, and generally processes inputs and creates outputs.  
Example:  
The REX1234 procedure for executing a Compile  
The input dialog to enter compile parameters |
| Property   | Properties are the characteristics of objects. Properties occur with different meanings, e.g. as the property of an element or container, as filter criterion, as the input or output parameter of a tool, as the input or output field in a dialog window. |
| Resource   | A Resource is an object that can be selected in the application user interface, that usually has properties, and on which actions can be carried out. Resources in the sense of this definition are, for example, elements, containers and filters. |

**Table 1-1 Basic expressions used**

A complete Eclipse based user interface for the integration of tools can be described with the help of the AWM Editor and the expressions defined above.

**Example:**

*Several Elements* can be identified in an SCM system. Elements can either represent the files managed by the SCM system or Containers in which the files are structured (e.g. projects, stages, folders). All Elements have *Properties*. A Property is an attribute of a type and is assigned a value at runtime (e.g. file name). *Tools*, on the other hand, use Properties as input parameters and assign properties to their output parameters. Tools are components of *Actions*, to which Elements can be assigned (e.g. Check-out, Edit, Compile, Delete).

A Delete Tool, for example, requires the file name of the file to be deleted as an input parameter. The file exists as an Element, and has the property “File name”. The tool is modelled in the AWM so that it uses the
“File name” property as an input parameter. Used on an Element, its file name will be transferred to the tool, and the file will be deleted.

The user can define the elements that he would currently like to work with using Filter during runtime. The criteria by which elements can be searched for can be entered in Filter. The corresponding elements can be returned via an action. Returned elements are managed in Element Lists.

All the object types are shown in Figure 1.1

*Figure 1.1 The major data model objects and relationships (simplified)*
Chapter 2
Introduction to the Data Model

The modelling of the user interface and the tool attachment using the Workflow Manager requires a detailed understanding of the data model. This is necessary in order to embed the tools in the development interface in the most user-friendly and high-performing way possible.

Through the classification and formal description of the terms introduced in the previous chapter, a model is produced, which is stored in an XML file, the AWM File. A model of this kind can be created for every application system that you wish to connect.

This chapter will begin by giving an overview of the most important classes, which will then be described in more detail in the following chapter.

2.1 Element Type

A class of elements is called the Element Type.

The element types are derived through analysis of the application systems to be connected. The classification of elements can be rough or fine. The advantage of a fine categorization is that properties and actions can be explicitly assigned to every Element Type in further modelling processes.

Examples:

- **Element types for an SCM system**
  - Editable element (rough categorization)
  - Non-editable element (rough categorization)
  - Program source (fine categorization)
  - COBOL program (very fine categorization)

2.1.1 Container Element Type

Any Element Type can be modelled as a Container Element Type. Container Element Types represent elements that are able to contain or reference other elements. Therefore Container Element Types are used to define a hierarchical structure of a connected system.
Chapter 2  Introduction to the Data Model

Examples:

- **Container Element Types for an SCM system**
  - Project (a project may contain development stages as elements)
  - Stage (a development stage may contain source types)
  - Source (an element may contain Includes or Copies)
  - COBOL Copy (a COBOL Copy may contain other Copies)

2.2  Element List Structure

In order that elements can be shown in a tabular form in the user interface, a formal description of this table structure is needed.

To do this, Element Lists with the same properties are described as an **Element List Structure**.

Examples:

- **Structure of a Work Item List with the properties**:
  - Work item number
  - File name
  - Change date
  - Last editor

- **Structure of a Version List with the properties**:
  - File name
  - Version number
  - Creation date of the version

2.3  Filter Type

Filters with the same types of filter criteria are referred to as **Filter Types**.

Filter Types are described by Filter Criteria, which determine the rules by which elements will be searched in an application system for further application processing.

Examples:

- **Work Item filter**
  - Criteria in order to be able to determine all files that are allocated to a work item

2.4  Property

A **Property** in the data model is the description of an attribute whose value can be determined at runtime.
A "Text document" element type is assigned the properties “File name”, “Change date” or “Write protected”. These characteristics are called **Properties**. They represent individual attributes, and describe how their values are to be interpreted. The **Property Value** can only be determined at runtime.

**Example:**

<table>
<thead>
<tr>
<th>Property</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Model</strong></td>
<td><strong>Data</strong></td>
</tr>
<tr>
<td>File name</td>
<td>MyDocument</td>
</tr>
<tr>
<td>Change date</td>
<td>2010/10/25</td>
</tr>
<tr>
<td>Write protected</td>
<td>true</td>
</tr>
</tbody>
</table>

**Table 2-1 Properties and Property Values of the “Text document” file type**

Table 2-1 indicates the three properties that have been assigned to the “Text document” element type. The individual Property Values have different meanings. “File name” can contain any type of characters, “Change date” is a date field and “Write protected” is a Boolean value. At runtime these values can be interpreted in different ways in a tool attachment, whereas an exact description of the properties is required in the data model.

Properties can be referenced from different model classes. Depending on the class in which the “File name” property is referenced, it will be processed differently. If the property is referenced from an element type, the “My Document” property value could be the labelling of the element. A Tool can also reference the same property, and use it as an input parameter.

The meaning of a property will only become clear in context with the reference. In most cases, Properties are referenced via **Relationships** (see Figure 2.1).
2.5 **Action Descriptor**

An Action is an integrally closed function that can be triggered in the user interface. An Action can consist of a number of Tools. In the data model, the description of an action is called the **Action Descriptor**. An Action Descriptor must refer to at least one Tool Descriptor. The execution number (SeqNo) (see Figure 2.2) that determines the processing sequence of the tool in the Action is determined in the relationship to a Tool Descriptor.

![Figure 2.1 Referencing of Properties](image)

*Model classes (top) reference Properties (bottom) through Relationships (centre)*

![Figure 2.2 Relationships of an Action to Tools](image)

*Figure 2.2 Relationships of an Action to Tools*
The execution of an action is usually actively triggered by the user. In some cases, an action can also be carried out automatically (for example, to determine the properties of an element).

At runtime Actions occur at the following locations:

- As an “Expand” function of structural Elements
- In the context menus of Filters
- As a standard action for filters (“Refresh” or double-click)
- In the context menus of Elements
- As a standard action for Elements (double-click)
- In the context menus of tables derived from a structured file descriptor
- In the Eclipse menu bar
- In the context menus of editors
- In an Element properties update
- In an Element List update

The actions that should be carried out for the above-listed types are defined in the AWM through the relationships of the types (Element type, Filter type, …) to Action Descriptors. Actions which do not belong to a certain type must be modelled as Global Actions (see 3.9).

### 2.6 Tool Descriptor

A Tool is an executable program that can communicate with the Client application via input/output parameters. The prerequisites for the attachment of Tools are the implementation of interfaces for the Tool call-up and the communication with the target system (see 6.3).

A tool is defined in the AWM by a **Tool Descriptor**.

A Tool Descriptor describes a Tool in such a way that the model interpreter can execute it and can react to its outputs.

The following are defined in a Tool Descriptor:

- The general attributes of the Tool
- Resource operations (e.g. Insert element into View)
- The logical name with which the implementation of the call-up interface is assigned (see Figure 2.3: Tool Type)
- Tool-specific attributes (see Figure 2.3: ISA_PlugIn)
- Input parameters
- Output parameters
- File parameters
Example:

The only parameter of the REXX Tool “Delete file” is the name of the file that should be deleted. In order to attach this Tool, a Tool Descriptor must be defined. The parameter for the file name is described via the “Has Input Parameter” relationship. The relationship should reference a Property that fulfills the logical meaning of the tool itself, namely a Property that stands for the file name of an Element. If the Tool Descriptor (as a component of an Action Descriptor) is executed on an Element at runtime, the Property Value of the file name property will be transferred to the Tool as an input parameter (see Figure 2.4).
If the input parameter is not related to the selected Element (or to the selected resource), the “origin” of a parameter can be defined in more detail in the AWM. An input parameter can also be derived from the output of a previous tool.

Figure 2.4 Attachment of a "Delete File" Tool

The “Delete File” Tool is shown on the left, together with an associated z/OS file. This situation is transferred to a Workflow configuration on the right-hand side. The input parameter of the Tool Descriptor references the same Property that the Element Type has.

If the input parameter is not related to the selected Element (or to the selected resource), the “origin” of a parameter can be defined in more detail in the AWM. An input parameter can also be derived from the output of a previous tool.
Chapter 3
Data Model Details

The following chapter contains a detailed description of all data model object types, relationships and attributes.

All the attributes are listed in a table, and more complex attributes and relationships will be explained in more detail in the corresponding sub-chapters. The most important attribute properties are represented in the tables by symbols (see Table 3-1 Attribute symbols and their meanings).

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Mandatory attribute. This attribute must be assigned a value.</td>
</tr>
<tr>
<td>[C]</td>
<td>(Choice) Occurs if there are at least two attributes for a type or relationship. Only one of these attributes may be assigned a value or set to true.</td>
</tr>
<tr>
<td>[D]</td>
<td>(Dependency) Occurs if there are at least two attributes for a type or a relationship. These attributes depend on each other and must be modelled in specific ways.</td>
</tr>
<tr>
<td>[O]</td>
<td>(Optional) Optional attribute. Optional attributes do not have a standard value and can be left empty.</td>
</tr>
<tr>
<td>[U]</td>
<td>(Unique) The value of this attribute must be unique among the siblings of the same type (see Figure 3.1).</td>
</tr>
<tr>
<td>[UC]</td>
<td>(Unique choice) Precisely one type or one relationship among the siblings must set this attribute to true. Only occurs with Boolean attributes.</td>
</tr>
</tbody>
</table>

Table 3-1 Attribute symbols and their meanings
Chapter 3  Data Model Details

Certain attributes, such as **ID**, **Label** and **SeqNo**, are assigned to several object types and relationships. In principle, these always have the same meaning:

- **ID**
  Mandatory attribute. The value of this attribute must be unique in the complete AWM. The use of a naming convention is therefore recommended.

- **Label**
  Optional attribute. Labels are used for the display of object and fields in the user interface. If the model interpreter does not find a label for an object of a type, the next matching attribute will be taken, which could be a Label for a higher hierarchy level or the ID of an object type.

  Example:
  
  Action Descriptor \(\text{ACT\_DELETE}\) \(\text{Label} = \text{“Delete Member”}\)
  Element \(\text{ELE\_Member}\) has Action Descriptor (relationship): \(\text{Label} = \text{“”} \) (empty)
  Element \(\text{ELE\_Task}\) has Action Descriptor (relationship): \(\text{Label} = \text{“Delete Task”}\)

  For elements of the type “\text{ELE\_Member}”, the label “Delete Member” matches the action, which should delete a member. Elements of the type “\text{ELE\_Task}” should be deleted via the context menu with “Delete Task”, even if the same Action Descriptor is hidden behind it. In this case, the label of the Action Descriptors can be “overwritten” with the label of the relationship. If the Action Descriptor “\text{ACT\_DELETE}” has no label, its ID will be used.

- **SeqNo**
  Only occurs in relationships. Defines the position of a referenced type in a sequence. For example, this could be the arrangement of the table columns or the sequence of Tool parameters. Only numbers are valid values, and the lowest number means that the referenced object should appear first in the sequence (first table column or first Tool parameter). The same sequence numbers among siblings of the same type generate no errors as long as the “Unique” ([U]) attribute is not defined. Use of the same sequence numbers leads to a random arrangement of the referenced objects.

Figure 3.1 Example of a "Unique" attribute [U]

Each attribute value of “**SeqNo**” must be unique among the same types of a parent mode.
Complex Attributes are made up of several individual attributes. Relationships and complex attributes can appear several times within a type.

Cardinalities of the following relationships and complex attributes will be described in tables in the next sub-chapters:

- 0..1 = Optional
- 0..* = Any number (also none)
- 1..* = At least one required
- 1..1 = Exactly one required

### 3.1 Enabled Condition

Enabled Condition is a general model class in which relationships can be linked to Actions, Tools or Tool Input Parameters. This determines the conditions under which the execution of an Action or Tool is permitted or an Input Parameter is passed to a tool.

Modelling an Enabled Condition, the administrator defines the allowed values for certain properties, so that the action can be executed. By doing this, simple and complex rules can be defined, where complex rules link the simple rules via Operators.

Example of a Simple Enabled Condition:
“If Property “Stage” has the value “DEV1”, the action can be executed”.

A Complex Enabled Condition contains any number of simple and complex Enabled Conditions, and links these via Operators.

In addition, the number of resources that can/must be selected so that an action is executable can be determined for an Enabled Condition. This value is ignored for Tool Enabled Conditions and for Input Parameter Enabled Conditions.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection_Count [O]</td>
<td>String</td>
<td>Number of resources that must be selected for the referenced action to be executable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Permitted values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Number of resources is irrelevant</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n = Exactly n (n = 1,2,3,...) resources must be selected</td>
</tr>
<tr>
<td>Complex_Enabled_Condition (0..1)[D]</td>
<td>Complex Condition</td>
<td>Complex condition. Consists of several conditions brought together.</td>
</tr>
<tr>
<td>Simple_Enabled_Condition (0..1)[D]</td>
<td>Simple Condition</td>
<td>Simple condition. Checks a Property Value for a specific value.</td>
</tr>
</tbody>
</table>

Table 3-2 Attributes of an Action Enabled Condition
Chapter 3  Data Model Details

16

A Simple Enabled Condition checks a Property Value for a defined value or status, and activates or deactivates the action depending on the result of the check.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID</td>
<td>Property</td>
<td>Reference to the Property whose value should be checked.</td>
</tr>
<tr>
<td>Operator</td>
<td>Selection</td>
<td>The check operator.</td>
</tr>
<tr>
<td>Value [D]</td>
<td>String</td>
<td>A fixed value that will be compared with that of the Property.</td>
</tr>
</tbody>
</table>

Table 3-3 Attributes of a Simple Enabled Condition

Figure 3.2 Modelling Action Enabled Condition

The Enabled Condition for an Element is defined here. It begins with a Complex Enabled Condition (center, bottom), which links the two Simple Enabled Conditions (lower left and right) with the AND Operator. Workflow Manager interprets this configuration as follows:

Action ACT_X from ElementType ELE1 can only be executed if the Property Value of PROP_Protected is false and corresponds to the Property Value of Property PROP_Group “TEST”. In addition, exactly one Element must be marked.

A Simple Enabled Condition checks a Property Value for a defined value or status, and activates or deactivates the action depending on the result of the check.
3.1 Enabled Condition

The meanings of the various operators are shown in the following table:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equals…</td>
<td>Checks whether the Property Value of the referenced Property has the same value as specified in “Value”.</td>
<td>The action can only be executed if the selected element is a COBOL program:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TargetID = PROP_Type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operator = Equals…</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value = COBOL</td>
</tr>
<tr>
<td>Equals not…</td>
<td>Checks whether the Property Value of the referenced Property has a value different from the specified “Value”.</td>
<td>The action can only be executed if the selected element is not in production:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TargetID = PROP_Stage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operator = Equals not…</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value = PROD</td>
</tr>
<tr>
<td>NULL</td>
<td>Checks whether the referenced Property has no Property Value.</td>
<td>The action can only be executed if the selected element has no access key:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TargetID = PROP_AccessKey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operator = NULL</td>
</tr>
<tr>
<td>NOT_NULL</td>
<td>Checks whether the referenced Property has any Property Value.</td>
<td>The action can only be executed if the selected element has a change date:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TargetID = PROP_ChangeDate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operator = NOT_NULL</td>
</tr>
<tr>
<td>TRUE</td>
<td>Checks whether the referenced Property has the Boolean value “true”.</td>
<td>The action can only be executed if the selected element can be edited:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TargetID = PROP_Editable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operator = TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>Checks whether the referenced Property has the Boolean value “false”.</td>
<td>The action can only be executed if the selected element is not write-protected:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TargetID = PROP_Protected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operator = FALSE</td>
</tr>
</tbody>
</table>

Table 3-4 Operators of a “Simple Enabled Condition”
Chapter 3  Data Model Details

3.2  Variable References

Some attributes in the Workflow Manager Model support the usage of variable references. In most cases, references are not specified manually, as the Model Editor (see Chapter 4) can automatically generate the references from a selection list. Manually referencing of Properties and System variables is only possible for special attributes:

- Attributes "Static_Parm" and "Default Value" of the relationship “Tool has Input Parameter” (see 3.13.1)
- Attribute "New_Static_Parm" of the relationship “Parameter Override” (see 3.4.3.1)
- Attribute “Name_Physical” in “File Descriptor” (see 3.7)
- Attribute “Value” in “Simple Enabled Condition” (see 3.1)

References can be defined by using the Start Reference and End Reference symbols (default “&” and “.”) which are defined in the Application Options (see 3.3). It is possible to define a value containing several references as well as static characters. Table 3-5 shows the syntax of all variables that are available.

<table>
<thead>
<tr>
<th>Variable/Syntax</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>%&lt;env-var&gt;</td>
<td>Reference to an environment variable on the client's local operating system</td>
<td>&amp;%PUBLIC. = C:\Users\Public</td>
</tr>
<tr>
<td>&lt;property ID&gt;</td>
<td>Reference to a property in the execution context</td>
<td>&amp;PROP_Membername. = MEMXYZ</td>
</tr>
<tr>
<td>userid</td>
<td>Client's user ID on the remote system</td>
<td>&amp;userid. = USER123</td>
</tr>
<tr>
<td>wspath</td>
<td>Path of the active Eclipse Workspace on the client's local machine</td>
<td>&amp;wspath. = C:\workspaces\workspace1</td>
</tr>
</tbody>
</table>

Table 3-5 List of all supported variables

The following example shows the definition of a dynamic file path:

Attribute value of “Name_Physical” (File Descriptor):

```
&userid.\SOURCE(&PROP_Membername.)
```

Resolved value:

```
USER123.\SOURCE(MEMXYZ)
```

3.3  Application Options

Global settings that apply to the complete application can be defined under the Application Options.

One attribute of the Application Options is Version. At runtime the version of the local AWM is compared with the version entry in the Master Configuration file (see 4.1).

The attributes Start Reference and End Reference are used for parsing variable references (see 3.2). By default these have the values “&” and “.”.
In addition, the tool execution can be controlled from the Application Options. The maximum return code that the Tool can return without its execution being interpreted as “failed” is specified in the attribute MaxRC. Failed Tools end with an error message, and prevent the execution of further tools in the Tool sequence of the Action.

With the EXITPARM attribute, the administrator can define a parameter value for an initial exit called before a tool is started. This can be used, for example, for the dynamic allocation of additional libraries to assure that the tool attachment runs in a test environment (see 6.4.7. Exit TAUTOXA1).

The Root Containers Action attribute can be used to refer to an action which returns the root elements used to build up the first hierarchical view of the application which will be visible in the Tree View. The referenced Action Descriptor must return an element list whose match name property typically references a Container Element Type (see 3.4.1).

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default_Generic_Symbol</td>
<td>String</td>
<td>The standard generic symbol. Usually means “all”. Generic input fields will be automatically filled in with this symbol.</td>
</tr>
<tr>
<td>End_Reference [D]</td>
<td>String</td>
<td>Symbol or character String that should be interpreted as meaning the end of a reference (see above).</td>
</tr>
<tr>
<td>EXITPARM</td>
<td>String</td>
<td>Permits the transfer of a parameter to the Tool call-up exit TAUTOXA1 (see 6.4.7).</td>
</tr>
<tr>
<td>Generic_Symbols [O]</td>
<td>String</td>
<td>Permits characters in linked systems for the generic search. The characters given here are used for test purposes in the input dialogs. The characters are entered directly, one after the other. In the MVS file system, these are the characters “*” and “%”.</td>
</tr>
<tr>
<td>Example: *%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max_RC *</td>
<td>Integer</td>
<td>The maximum permitted return code of a tool that is called from the model interpreter. The tool execution will be interpreted as being faulty if this value is exceeded.</td>
</tr>
<tr>
<td>Root Containers Action [O]</td>
<td>ActionDescriptor</td>
<td>The selected action will be executed when the Application entry in the tree view is expanded by the user in order to retrieve Container Elements of the first hierarchy level.</td>
</tr>
<tr>
<td>Start_Reference [D]</td>
<td>String</td>
<td>Symbol or character String that should be interpreted as meaning the start of a reference.</td>
</tr>
<tr>
<td>Example: &amp;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Chapter 3  Data Model Details

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version *</td>
<td>Double</td>
<td>Version of the AWM.</td>
</tr>
<tr>
<td></td>
<td>(e.g. 0.123)</td>
<td></td>
</tr>
</tbody>
</table>

*Table 3-6 Attributes of the Application Options*

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has_Specific_Attribute</td>
<td>0..4</td>
<td>Relationship to a specific attribute.</td>
</tr>
</tbody>
</table>

*Table 3-7 Relationships of Application Options*

#### 3.3.1 Relationship: Application Options have Specific Attributes

The relationship `Has_Specific_Attribute` has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name *</td>
<td>String</td>
<td>Attribute name.</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td>Attribute value.</td>
</tr>
</tbody>
</table>

*Table 3-8 Attributes of the Relationship “Application Options have Specific Attributes”*

This relationship is provided to overwrite software specific attribute values with customer specific values.
The following specific attribute names are supported:

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dialog_Ok_Label</td>
<td>The label of the OK button in all modelled dialogs except ISPF dialogs.</td>
</tr>
<tr>
<td>Dialog_Cancel_Label</td>
<td>The label of the CANCEL button in all modelled dialogs except ISPF dialogs.</td>
</tr>
<tr>
<td>ISPF_Ok_Label</td>
<td>The label of the OK button in modelled ISPF dialogs.</td>
</tr>
<tr>
<td>ISPF_Cancel_Label</td>
<td>The label of the CANCEL button in modelled ISPF dialogs.</td>
</tr>
</tbody>
</table>

Table 3-9 Specific attributes supported

3.4 **Action Descriptor**

An Action Descriptor describes a sequence of Tools. Actions can be executed at the following locations:

- In the context menus of Elements, Filters, File Descriptors and Jobs. The context menus will be modelled for each type using the relationship “Has_Action”.
- Determining/Updating the Property Values of an Element (“Get_All_Properties” attribute of the Element Type).
- Creating/Updating an Element List or expanding a container element within tree view.
- Saving/Closing the editor during an editing procedure.
- Determining the permitted values for a Property.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID *</td>
<td>ID *</td>
<td>The ID of the Action Descriptor</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>The labeling of the Action Descriptor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The label of an Action Descriptor is mainly seen in the context menu.</td>
</tr>
<tr>
<td>Execution Mode</td>
<td>Selection</td>
<td>Valid values are MODAL or MODELESS. You have to make sure that no parallel ISPF requests are sent to one ISPF session. For ISPF Dialog Tools it is recommended to use the MODELESS execution mode.</td>
</tr>
<tr>
<td>Has_Element List [O]</td>
<td>Element List Structure</td>
<td>Only to be set if one of the Tools returns an Element List. The Element List will then be assigned to the selected Element List Structure (see 3.4.1).</td>
</tr>
<tr>
<td>Related_Save_Action [O]</td>
<td>ActionDescriptor</td>
<td>To be set if dealing with an Edit action. The selected Save Action will be executed when saving the editor.</td>
</tr>
</tbody>
</table>
### Chapter 3  Data Model Details

#### Table 3-10 Attributes of an Action Descriptor

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Related_Close_Action [O]</td>
<td>ActionDescriptor</td>
<td>To be set if dealing with an Edit action. The selected Close Action will be executed when closing the editor.</td>
</tr>
<tr>
<td>Refresh_List</td>
<td>Boolean</td>
<td>If set to true, the current Element List will be refreshed at the end of the action executed on an Element (via the same action that was used to create the list). Also an Element List Action always creates a new List (means that an already cached Element List is not used). Exceptions: Filter Actions and Actions expanding the Tree View are always using the cached lists. Standard value: false</td>
</tr>
<tr>
<td>Icon [O]</td>
<td>Icon</td>
<td>The Icon of an Action Descriptor can be seen in context menus.</td>
</tr>
</tbody>
</table>

#### Table 3-11 Relationships of an Action Descriptor

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has_Tool</td>
<td>1..*</td>
<td>Relationship to a Tool Descriptor. The Tool sequence for an action arises through these relationships.</td>
</tr>
</tbody>
</table>

### 3.4.1 Has_Element List

Actions that generate new Element Lists will be defined by a reference in an Element List Structure by an Action Descriptor. Note that precisely one Tool must output one file, in which all the elements for the Element List are contained.

The output file must contain as a minimum all values of the Key Properties (see 3.11.1) and the Match Name (see 3.6) of the corresponding Element Type. Additionally output Property Values can be assigned to the corresponding Element and be shown in the table if these properties have also been assigned to the Element Type. Every line of the output file will be interpreted as a String of Property Values for an Element (see Figure 3.3).
3.4.2 Related Save-/Close Action

A series of Edit scenarios can be modelled. Editing can take place locally or remotely. With local editing, all changes are stored locally and are only written back to the remote system when the user explicitly requests this. When editing on the remote system, all changes will be written back to the remote system immediately when saving or closing the editor.

These actions will be modelled via the references Related_Save_Action and Related_Close_Action in the corresponding Edit action. The Save action will be carried out every time there is a request to save the file. The Close action will be carried out when the editor is closed (this also happens when closing Eclipse).

3.4.3 Relationship: Action has Tool

The relationships of an Action Descriptor to the Tool Descriptors form the Tool sequence of an Action. A Tool sequence can consist of different Tool types. An action can start, for example, with an input dialog (Plug-In), call up two Host procedures and end with an output dialog (Plug-In). The Tool sequence will be interrupted as soon as a Tool runs incorrectly or if the user cancels an input dialog.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID *</td>
<td>ToolDescriptor</td>
<td>The Tool Descriptor that should be referenced via this relationship.</td>
</tr>
</tbody>
</table>
Chapter 3  Data Model Details

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SeqNo * [U]</td>
<td>Integer</td>
<td>Sequence number at which the referenced Tool should be called up. The Tools will be called up one after the other according to their sequence numbers. The faulty run of a Tool interrupts the complete sequence. Sequential numbering (1, 2, 3, …) is not obligatory.</td>
</tr>
<tr>
<td>Affected Resource name [O]</td>
<td>String</td>
<td>For future use</td>
</tr>
<tr>
<td>Parameter.Override [O]</td>
<td>Parameter.Override</td>
<td>Relationship to an input parameter of the referenced Tool Descriptor. This makes adaptations to the input parameters possible for precisely one action, so that the re-usability of Tool Descriptors will be increased (see 3.4.3.1).</td>
</tr>
<tr>
<td>Enabled Condition [O]</td>
<td>Enabled.Condition</td>
<td>Relationship to a Tool Enabled Condition which allows to define the conditions which are checked at runtime and must be true to call the tool (see 3.1 for details how to define an Enabled Condition)</td>
</tr>
</tbody>
</table>

Table 3-12 Attributes of the "Action has Tool" relationship

3.4.3.1 Parameter Override

With a Parameter Override, specific attributes of an input parameter can be overwritten or adapted for exactly one Action Descriptor.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parm_To_OVERRIDE</td>
<td>Tool_Has_InputParameter</td>
<td>The Input Parameter of the referenced Tool Descriptor should be adapted.</td>
</tr>
<tr>
<td>New_Parm_From</td>
<td>Selection</td>
<td>Overwrites the “Parameter_From” attribute of the input parameter (see 3.13.1).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Possible values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Context (standard value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Previous_Tool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>User_Input</td>
</tr>
<tr>
<td>New_Static_Value [C]</td>
<td>String</td>
<td>Overwrites the “Static_Value” attribute of the input parameter (see 3.13.1).</td>
</tr>
<tr>
<td>Remove_Parm [C]</td>
<td>Boolean</td>
<td>Ignores these input parameters for this Action Descriptor. Is only possible if the input parameter is declared as optional (see 3.13.1).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard value = false</td>
</tr>
</tbody>
</table>

Table 3-13 Attributes of a Parameter Override

The Parameter Override attributes mutually exclude each other. Only one attribute can therefore be interpreted. The New_Parm_From attribute, for example, only works if no static parameter (New_Static_Value) has been set. Similarly, a parameter can only assume a static value if it should not be ignored (Remove_Parm).
3.5 Element List Structure

An Element List Structure describes a table structure for the table view or a tree table view. This is basically the definition of columns, in which the Property Values of Elements can be displayed at runtime (See 3.6.3 for more information about Element properties).

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID *</td>
<td>ID *</td>
<td>The ID of the Element List Structure.</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>The Display Name of the Element List Structure. The user only sees the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>display names in the Column Configuration dialog. If no label is stated,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the ID will be used.</td>
</tr>
</tbody>
</table>

Table 3-14 Attributes of an Element List Structure

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has_Column</td>
<td>1..*</td>
<td>Relationship to a Property that describes a Table column of the Element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>List Structure</td>
</tr>
<tr>
<td>Has_Property</td>
<td>0..*</td>
<td>Relationship to a Property that describes an attribute of the Element List</td>
</tr>
<tr>
<td>Has_Action</td>
<td>0..*</td>
<td>Relationship to an Action Descriptor that has an effect on any elements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>in the list.</td>
</tr>
</tbody>
</table>

Table 3-15 Relationships of an Element List Structure

3.5.1 Relationship: Element List has Action

**Element List Actions** can be defined via the “Element List has Action” relationship. Element List Actions are peculiar in that they can appear in the context menu of the Element List, but are defined independently of the Element Types.

This makes it possible to define actions that:

- can be used on all the Elements of an Element List
- are only activated if no Elements have been selected in the list, or
- can always be carried out.

This is dependent on the **Selection Count** attribute of the **Action Enabled Condition** (see 3.1). If no Action Enabled Condition is defined, the action can always be carried out.

Examples of Element List Actions:

- Opening an Element (all Element Types) in the Browse mode  
  (Selection Count = 1)
- Inserting an Element into an empty Element List  
  (Selection Count = 0)
### Chapter 3  Data Model Details

#### 3.5.2 Relationship: Element List has Column

Columns for the Element List Structure are defined via the “Element List has Column” relationship. All the property values of an Element that match the properties of the table columns will be displayed in the table. The modelled structure of the column definition (position, visibility) is only a user default. Columns can be individually interchanged, displayed or hidden via a Column Configuration dialog.

As a rule, most properties that are referenced here are also assigned to the Element, i.e. they are also properties for the affected Element types.

However, properties that are only relevant in the context of the Element List can also be referenced, i.e. they cannot be directly assigned to an Element Type, as:

- o  the property does not match the Element Type logically
- o  the Property Values can differ for the same Element in different lists

#### 3.5.3 Relationship: Element List has Property

Properties can be assigned to an Element List Structure that describes the properties of an Element List. A property has the following meaning for an Element List:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID *</td>
<td>Property</td>
<td>The Reference to a Property. The Property Value of this Property is then displayed in the column of the Element List at runtime.</td>
</tr>
<tr>
<td>ColNo [O]</td>
<td>Integer</td>
<td>Position number of the table column (1 = at the extreme left).</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>The display name and/or the column title for the table column. If no label is stated, the label of the referenced Property will be used.</td>
</tr>
<tr>
<td>Visible</td>
<td>Boolean</td>
<td>Indication of whether the table column is displayed as standard. Columns can be displayed or hidden by the user at any time.</td>
</tr>
<tr>
<td>Width</td>
<td>Integer</td>
<td>Standard value = true Column width in the table view in Pixel</td>
</tr>
</tbody>
</table>

---

Table 3-16 Attributes of the "Element List has Action" relationship

Table 3-17 Attributes of the "Element List has Column" relationship

Table 3-18 Attributes of the "Element List has Property" relationship
It applies to all the Elements that are included in the list
It can be used as an input parameter for Tools that are listed in connection with Elements.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID *</td>
<td>Property</td>
<td>The reference to a Property that describes a property of the Element List Structure.</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>The display name of the referenced Property in the Properties view. If no label is stated, the label of the referenced Property will be used.</td>
</tr>
<tr>
<td>SeqNo [O]</td>
<td>Integer</td>
<td>Position number of the property in the Properties view.</td>
</tr>
</tbody>
</table>

Table 3-18 Attributes of the "Element List has Property" relationship

### 3.6  Element Type

An Element Type describes a class of Elements, including their properties and applicable actions.

Every Element must belong to an Element Type at runtime. You assign it via the **Match Name**. In addition, each Element Type has a Match Name attribute, under which a unique name for the type must be defined. A Tool that creates a new Element must return this name under a designated property (see Figure 3.3).

You can configure Element Types hierarchical so that they can be displayed in a Tree View. This is done using the **Get_Children** attribute.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID *</td>
<td>ID *</td>
<td>The ID of the Element Type Relationship to an action that returns all the Property Values of an Element.</td>
</tr>
<tr>
<td>Get_All_Properties</td>
<td>ActionDescriptor</td>
<td>The Icon of the Element Type. The name by which the Element Type can be determined.</td>
</tr>
<tr>
<td>Icon [O]</td>
<td>Icon</td>
<td>The ID Definition of the Element Type (see 3.11). The properties of the Element Type that uniquely identify an Element within an application will be described via this definition.</td>
</tr>
<tr>
<td>Matchname *</td>
<td>String</td>
<td>Reference to an action that returns the children elements. If you expand an element in the Tree View this action will display the next hierarchical level of elements.</td>
</tr>
</tbody>
</table>

Table 3-19 Attributes of an Element Type
### 3.6.1 Relationship: Element has Action

Using the “Element has Action” relationship, an Element Type can be assigned actions that can be executed on elements of this type. Action Descriptors referenced via this relationship appear in the context menu of all elements of this Element Type, but could also be deactivated there. An **Action Enabled Condition** (see 3.1) determines when an action should be activated.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID *</td>
<td>ActionDescriptor</td>
<td>The reference to the Action Descriptor.</td>
</tr>
<tr>
<td>ActionEnabledCondition</td>
<td>ActionEnabledCondition</td>
<td>This complex attribute lets you define rules as to when the action is activated. Non-activated actions will be shown in grey, and cannot be executed by the user.</td>
</tr>
<tr>
<td>Default_Action [U]</td>
<td>Boolean</td>
<td>Defines whether this action should be carried out by a double-click on the Element. Only one action of an Element Type can be declared as the Default_Action. Standard value: false</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>Labeling of the action in the context menu. If no label is stated, the label of the referenced Action Descriptor will be used.</td>
</tr>
<tr>
<td>Separator_After</td>
<td>Boolean</td>
<td>Determines whether a separation line should be inserted behind the context menu entry. Standard value: false</td>
</tr>
<tr>
<td>SeqNo [O]</td>
<td>Integer</td>
<td>The Position Number of the Action in the context menu of an Element.</td>
</tr>
</tbody>
</table>
Table 3-21 Attributes of the "Element has Action" relationship

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show_In_Table_View</td>
<td>Boolean</td>
<td>Determines whether the referenced action should be displayed in the context menu of the Element Table View.</td>
</tr>
<tr>
<td>Show_In_Tree_View</td>
<td>Boolean</td>
<td>Determines whether the referenced action should be displayed in the context menu of the Tree View.</td>
</tr>
</tbody>
</table>

3.6.2 Relationship: Element has Filter

The user can create an Element Filter from the context menu via the relationships of an Element Type to the Filter Type. The Filter dialog modelled in the Filter Type will be automatically called up, which will pre-assign appropriate input fields of the Filter criteria with the Property Values of the Element.

Table 3-22 Attributes of the "Element has Filter" relationship

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID *</td>
<td>FilterType</td>
<td>The Filter Type that should be referenced via this relationship. A Filter dialog of this type is called up as soon as the user wants to create a filter via the referenced Element Type.</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>Label in the context menu for the creation of a new filter.</td>
</tr>
<tr>
<td>SeqNo [O]</td>
<td>Integer</td>
<td>The position of the entry in the context menu of the Element. Filter action are always positioned before other Element actions</td>
</tr>
<tr>
<td>Show_In_Table_View</td>
<td>Boolean</td>
<td>Determines whether the “Create Filter” action should be displayed in the context menu of the Element Table View.</td>
</tr>
<tr>
<td>Show_In_Tree_View</td>
<td>Boolean</td>
<td>Determines whether the “Create Filter” action should be displayed in the context menu of the Tree View.</td>
</tr>
</tbody>
</table>

3.6.3 Relationship: Element has Property

The properties of an Element Type will be referenced via the “Element has Property” relationship.

Element properties can be seen both in the columns of the Element Table view and in the Properties view when an Element has been selected (see Figure 3.4). They are also essential for the implementation of actions on Elements, as they can be used as parameters for Tools (see 3.12.1).
### Chapter 3  Data Model Details

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID *</td>
<td>Property</td>
<td>The reference to the Property.</td>
</tr>
<tr>
<td>Category [O]</td>
<td>String</td>
<td>The category under which the property is displayed in the Properties view.</td>
</tr>
<tr>
<td>Element Name [U]</td>
<td>Boolean</td>
<td>Indicates whether the Property Value of this Property contains the name of the Element. Among other things, the Element name will be used as editor title.</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>Labeling of the property in the Properties view. If no label is stated, the label of the Property will be used.</td>
</tr>
<tr>
<td>SeqNo [O]</td>
<td>Integer</td>
<td>Position number of the property in the Properties view. Sequence numbers that appear twice result in a random arrangement of the Properties in the Property View. Note: The Category always has the higher priority in the arrangement of a Property in the Properties view.</td>
</tr>
</tbody>
</table>
3.7 File Descriptor

A File Descriptor describes a file and, optionally, the structure of the file. The file can be in both the Eclipse Workspace and in the system to be attached.

The physical name (attribute Name_Physical) of the file does not have to be given in order to describe a file. This can be set by a Tool output parameter at runtime. If the physical name of the file already exists, a static value can be entered under this attribute. If only the structure of the file path has been defined, a dynamic path can be entered into the AWM using dynamic references (see 3.2).

A further feature of a File Descriptor is the Type. The “workspace_File” type indicates files that should be saved in the Workspace or are to be read from the Workspace.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID *</td>
<td>ID *</td>
<td>The ID of the File Descriptor.</td>
</tr>
<tr>
<td>Name_Physical [O]</td>
<td>String</td>
<td>The physical name of the file under which the target system can find the file (file path + file name). Dynamic references are allowed.</td>
</tr>
<tr>
<td>PropDelimiter</td>
<td>String</td>
<td>The delimiter that should be used for the interpretation of structured file content. Standard value: “ “ (Blank)</td>
</tr>
<tr>
<td>Structure</td>
<td>Selection</td>
<td>Describes the structure of the file. Permitted values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Unstructured (standard value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Table</td>
</tr>
</tbody>
</table>

Table 3-23 Attributes of the “Element has Property” relationship

3.7 File Descriptor

A File Descriptor describes a file and, optionally, the structure of the file. The file can be in both the Eclipse Workspace and in the system to be attached.

The physical name (attribute Name_Physical) of the file does not have to be given in order to describe a file. This can be set by a Tool output parameter at runtime. If the physical name of the file already exists, a static value can be entered under this attribute. If only the structure of the file path has been defined, a dynamic path can be entered into the AWM using dynamic references (see 3.2).

A further feature of a File Descriptor is the Type. The “workspace_File” type indicates files that should be saved in the Workspace or are to be read from the Workspace.
### Chapter 3  Data Model Details

#### Table 3-24 Attributes of a File Descriptor

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Selection</td>
<td>The Type of the File Descriptor. This value defines how the described file should be interpreted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Permitted values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- InputStream</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Local file</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Workspace_File (standard value)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MVS_PO</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MVS_PO_Member</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- MVS_SEQ</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- USS_File (can only be used if the server supports access to the z/OS USS file system)</td>
</tr>
</tbody>
</table>

#### Table 3-25 Relationship of a File Descriptor

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has_Property</td>
<td>0..*</td>
<td>Relationship to a Property. Property Values can be assigned to a file content, which enables the further editing of the contents. This relationship can only be used for the description of structured files (table structure).</td>
</tr>
<tr>
<td>Has_Action</td>
<td>0..*</td>
<td>Relationship to an Action Descriptor. Used only if the file descriptor is of structure table. Then the actions appear in the context menu of the table items.</td>
</tr>
</tbody>
</table>

#### 3.7.1 Relationship: File Descriptor has Property

In addition to the creation and reading of files, it is possible to interpret and process the content of a file. File contents can also be linked with properties. This can be modelled via the “File has Property” relationship.

The linking of file contents to properties is only possible if the file consists of records with a defined structure (see Figure 3.5 Link between File Content and Properties).

#### Table 3-26 Attributes of a Property

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID</td>
<td>Property</td>
<td>The reference to a Property whose Property Value is part of the structured file</td>
</tr>
<tr>
<td>Label</td>
<td>String</td>
<td>The labelling of a table column if the file should be displayed in a table. If no Label is stated, the name will be used as a label.</td>
</tr>
<tr>
<td>SeqNo</td>
<td>Integer</td>
<td>The Sequence Number of the property value in the structured file.</td>
</tr>
<tr>
<td>Visible</td>
<td>Boolean</td>
<td>Indicator of whether this Property should be displayed as a column in a table view. Standard value: true</td>
</tr>
</tbody>
</table>
Table 3-26 Attributes of the "File Descriptor has Property" relationship

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID *</td>
<td>ActionDescriptor</td>
<td>The reference to the Action Descriptor.</td>
</tr>
<tr>
<td>Default_Action [U]</td>
<td>Boolean</td>
<td>Defines whether this action should be carried out by a double-click on the table item. Only one action of a file descriptor of structure table can be declared as the Default_Action. Standard value: false</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>Labeling of the action in the context menu. If no label is stated, the label of the referenced Action Descriptor will be used.</td>
</tr>
<tr>
<td>Separator_After</td>
<td>Boolean</td>
<td>Determines whether a separation line should be inserted after the context menu entry. Standard value: false</td>
</tr>
<tr>
<td>SeqNo [O]</td>
<td>Integer</td>
<td>The Position Number of the Action in the context menu of a table item.</td>
</tr>
</tbody>
</table>

Table 3-23 Attributes of the "File Descriptor has Action" relationship

3.7.2 Relationship: File Descriptor has Action

If the File Descriptor is defined as a structured table the actions to appear in the table’s context menu can be modelled via the relation between the File Descriptor and an Action Descriptor.
3.8 Filter Type

A Filter Type describes a class of Filters. It contains all the information to generate a filter dialog, through which filters can be created. The Element Lists can be generated from a filter using the relationships of the Filter Type to Action Descriptors.

At runtime Filters can be created and edited via the Filter view. If Element Types have been defined with relationships to Filter Types, Filters can also be created via the context menu of an Element. The input fields of the Filter dialog will be initialised with the Property Values of an Element.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID *</td>
<td>ID *</td>
<td>The ID of the Filter Type.</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>The label of the Filter in the context menu of the Filter view to define a new Filter. If not used, the ID will be displayed.</td>
</tr>
</tbody>
</table>

Table 3-27 Attributes of a Filter Type

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has_Action</td>
<td>1..*</td>
<td>Relationship to an Action Descriptor, via which an Element List can be generated using the Filter criteria.</td>
</tr>
<tr>
<td>Has_FilterCriterion</td>
<td>1..*</td>
<td>Relationship to a Property that will be interpreted as a filter criterion.</td>
</tr>
</tbody>
</table>

Table 3-28 Relationships of a Filter Type

3.8.1 Relationship: Filter has Action

Actions that can be carried out on the Filter will be defined via the “Filter has Action” relationship. These are suitable for generating the corresponding Element Lists from Filters. It is possible to assign several actions to a Filter Type that return different Element Lists. One relationship to an Action Descriptor must set the “Default_Action” attribute to true, so that the user can generate an Element List by double-clicking the Filter.

If an Element List already exists for the action to be carried out, this will be displayed. Refreshing the content of the Element List is only possible via the “Refresh” function of the Element Table view.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID *</td>
<td>ActionDescriptor</td>
<td>The reference to an Action Descriptor that is to be carried out by the Filter Action. Preferably an Action Descriptor that returns an Element List.</td>
</tr>
</tbody>
</table>
### 3.9 Global Action

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID *</td>
<td>Property</td>
<td>Reference to a Property that should be used as a filter criterion.</td>
</tr>
<tr>
<td>Mandatory</td>
<td>Boolean</td>
<td>Indicates whether the criterion is mandatory for the creation of a filter. If true, the user must specify a value in the generated input field in the Filter dialog. Mandatory filter criteria are marked with the symbol * in the input fields. Note: a space value is a valid mandatory input.</td>
</tr>
<tr>
<td>SeqNo [O]</td>
<td>Integer</td>
<td>The position number of the Filter criterion input field within a Filter dialog.</td>
</tr>
<tr>
<td>Generic</td>
<td>Boolean</td>
<td>Indicates whether wildcard characters (see 3.2) are permitted as Filter criterion. Standard value: false</td>
</tr>
</tbody>
</table>

Table 3-30 Attributes of the "Filter has Filter Criterion" relationship

### 3.9.2 Relationship: Filter has Filter Criterion

The “Filter has Filter Criterion” relationship references Properties that will be interpreted as Filter criteria. Filter dialogs can be generated displaying the Filter criteria as input fields.

When carrying out an action on the corresponding filter, the values of the Filter criteria (and/or the Property Values of the referenced properties) can be used as input parameters.

As a rule, a Filter Type has at least one action that returns an Element List on the basis of the Filter criteria.

### 3.9 Global Action

A Global Action describes the location of an executable action which is described by an Action Descriptor. Global Actions are usually used for Action Descriptors which do not necessarily belong to a certain type (Element Type, Filter Type, ...).

There are two types of Global Actions: Workbench Actions and Editor Actions.
Chapter 3  Data Model Details

- Workbench Actions
  - Appear in the Eclipse menu bar
  - The action context (initial Property Values) will be created from the user’s selection

- Editor Actions (see 3.9.1)
  - Appear in context menus of text editors
  - The action context will be created from the opened element and the selection in the editor (depending on the modelled rules)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID *</td>
<td>ID</td>
<td>The ID of the Global Action.</td>
</tr>
<tr>
<td>TargetID *</td>
<td>Action Descriptor</td>
<td>Reference to an Action Descriptor.</td>
</tr>
<tr>
<td>ISA Editor Action [C]</td>
<td>ISA_Editor_Action</td>
<td>Complex attribute for editor actions (see 3.9.1).</td>
</tr>
<tr>
<td>ISA Workbench Action [C]</td>
<td>ISA_Workbench_Action</td>
<td>Complex attribute for workbench actions (see 3.9.2).</td>
</tr>
<tr>
<td>ActionEnabled-Condition [O]</td>
<td>ActionEnabled-Condition</td>
<td>This complex attribute makes it possible to define rules as to when the action is activated. Non-activated actions will be shown in grey, and cannot be executed by the user.</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>The label of the Global Action which appears in context menus or the menu bar.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If not specified, the Label of the referenced Action Descriptor will be used.</td>
</tr>
<tr>
<td>SeqNo [O]</td>
<td>Integer</td>
<td>The position number of the Global Action in menus.</td>
</tr>
<tr>
<td>Submenu [O]</td>
<td>String</td>
<td>The name of a submenu in which the action should appear. If not specified the action is created as a top level menu entry</td>
</tr>
<tr>
<td>Icon [O]</td>
<td>Icon</td>
<td>The icon of the Global Action to be displayed in menus.</td>
</tr>
</tbody>
</table>

Table 3-3 | Attributes of a Global Action

3.9.1 Global Action ISA Editor Action

By specifying the complex attribute ISA Editor Action, the corresponding Global Action acts as Editor Action.

An Editor Action will appear in the context menu of any editor if the file suffix of the opened file matches one of the modelled Editor Suffixes.
Editor Actions are able to use the editor’s context if the editor was opened by a modelled action (see 3.13.3).

It is also possible to dynamically use the user’s cursor position or text selection to add Properties to the action context. To do this, a Selection Property Mapping must be modelled. It specifies which value from the user’s editor selection (or cursor position) should be mapped to which property.

Parsers can be used to return Keywords from specific syntax. For example, a COBOL parser can return the name of a copybook if the cursor is placed on a valid copy statement. The advantage of this is that the corresponding Editor Action is only enabled if the cursor is placed on such a Keyword.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Editor Suffixes *</td>
<td>String[]</td>
<td>Blank separated list of file suffixes for which the editor action applies (case insensitive). If the action should appear in all text editors, an asterisk (*) must be used. Example: COB CBL. Result: The editor action will appear in the context menu of any text editor for files with the suffix COB and CBL (e.g. MYPROG.cbl). Standard value: *</td>
</tr>
<tr>
<td>Selection Property Mapping [O]</td>
<td>Selection Property Mapping 0..*</td>
<td>Complex attribute which maps a specific text selection in the editor to a Property, which can be used as an input parameter for tools within the action.</td>
</tr>
</tbody>
</table>

Table 3-32 Attributes of the complex attribute "ISA Editor Action"

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID *</td>
<td>Property</td>
<td>Reference to a Property to which the text selection should be mapped.</td>
</tr>
</tbody>
</table>
### 3.9.2 Global Action ISA Workbench Action

By specifying the complex attribute ISA Workbench Action, the corresponding Global Action acts as a Workbench Action.

A Workbench Action appears under its modelled label or submenu in the Eclipse menu bar. The context of a Workbench Action is created from the user selection in the Eclipse workbench.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool Tip [O]</td>
<td>String</td>
<td>The help text that is displayed when the mouse is hovered over the Global Action.</td>
</tr>
<tr>
<td>Type</td>
<td>Selection</td>
<td>The type of the Workbench Action. The only available type is currently “Menubar”.</td>
</tr>
</tbody>
</table>

*Table 3-34 Attributes of the complex attribute "ISA Workbench Action"*
The Eclipse menu bar with modelled Actions

This sample above shows a Workbench Action submenu (1) and a Workbench Action without submenu (2). The “Browse Member” action is disabled due to an Action Enabled Condition.

3.10 Icon

Icons can be used to assign symbols to various object types. This helps in the differentiation of the displayed types at runtime or for orientation in context menus, table views, tree views etc..

Modelled icons can appear for example at the following places at runtime:

- In the Element Table view for Element Types
- In the context menu for Elements (Element Table view)
- In the Tree View for Element Types

When configuring an icon, a selection of the available icons is displayed.

The following icons are available:

<table>
<thead>
<tr>
<th>Name of the icon</th>
<th>Graphic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action_Browse</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Action_Build</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Action_ChckIn</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Action_CheckOut</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Action_Edit</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Action_Insert</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Action_Promote</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Action_Remove</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Action_ShowHistory</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Application</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Container_Group</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Container_Project</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Database</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>ElementEditable</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Element_LockEntry</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Element_NonEditable</td>
<td><img src="image" alt="Icon" /></td>
</tr>
<tr>
<td>Element_Package</td>
<td><img src="image" alt="Icon" /></td>
</tr>
</tbody>
</table>
Chapter 3  Data Model Details

<table>
<thead>
<tr>
<th>Name of the icon</th>
<th>Graphic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element_VersionEntry</td>
<td><img src="image1.png" alt="Icon" /></td>
</tr>
<tr>
<td>Job_Running</td>
<td><img src="image2.png" alt="Icon" /></td>
</tr>
<tr>
<td>Library</td>
<td><img src="image3.png" alt="Icon" /></td>
</tr>
<tr>
<td>Package_Folder</td>
<td><img src="image4.png" alt="Icon" /></td>
</tr>
</tbody>
</table>

Table 3-35 Predefined icons

Action Icons are shown in grey if the action cannot be executed due to an Action Enabled Condition (see 3.1).

Icons are modelled by being defined once and then referenced from the supported Object classes.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID *</td>
<td>ID *</td>
<td>The ID of the Icon.</td>
</tr>
<tr>
<td>Predefined</td>
<td>Selection</td>
<td>This attribute is used to select an icon (according to Table 3 32).</td>
</tr>
</tbody>
</table>

Table 3-36 Attributes of an Icon

3.11  ID Definition

An ID Definition is a description of an identifier for the elements of one or more Element Types.

In order to be able to guarantee the identification of an element, criteria must be defined that make an Element unique. The criteria, the **Key Properties**, refer to the properties of an Element Type. The Key Properties will also be used to persistently administer Elements in the workspace.

Tools that return Elements or Element Lists must return at least all the Key Property Values of the Element. An Element cannot be created without the existence of all the Key Property Values.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID *</td>
<td>ID *</td>
<td>The ID of the ID-Definition.</td>
</tr>
</tbody>
</table>

Table 3-37 Attribute of an ID Definition

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has_Key_Property</td>
<td>1..*</td>
<td>Relationship to a Property. The referenced Property must also be assigned to the corresponding Element Types.</td>
</tr>
</tbody>
</table>

Table 3-38 Relationship of an ID Definition
Example of an ID Definition:

<table>
<thead>
<tr>
<th>PDS Member under z/OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data set name</td>
</tr>
<tr>
<td>Member name</td>
</tr>
</tbody>
</table>

Table 3-39 Example of an ID Definitions

### 3.11.1 ID has Key Property

With the relationship “ID has Key Property”, the Properties are referenced so that, together, they form an ID for Elements of one or more Element Types.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetID *</td>
<td>Property</td>
<td>Reference to a Property that is a Key Property of an ID Definition.</td>
</tr>
<tr>
<td>SeqNo *</td>
<td>Integer</td>
<td>The sequence number of the Key Property.</td>
</tr>
</tbody>
</table>

Table 3-40 Attributes of the "ID has Key Property" relationship
### 3.12 Property

A Property describes an attribute that can be assigned to nearly all data model types. The value of a Property (Property Value) will only be assigned at runtime (see 2.4).

Each referenced Property will be displayed with its Property Value (if available) in the Property View when the referencing object is selected. The display names of Elements are also Property Values of Properties that have been referenced with special attributes in a relationship (see Figure 3.6).

**Figure 3.6 Display names of Container Elements**

The containers (left) all originate from the Container Type (center) and share properties (right). The Property Values of the Container can be different, however. The display name of the container will be defined via a relationship (top center).

Different features can be defined when configuring a Property. One is the **Type** of the Property. This attribute defines how a Property or its Property Value is to be interpreted. The following Property Types are supported:

- **Type = String**
  This type is the default value. The Property Value can contain text and is displayed or entered in dialog input fields.

- **Type = Boolean**
  Boolean value: The Property Value will be interpreted as 1 (true) and 0 (false), and can be displayed or selected via a control box in dialogs.

- **Type = Select_Value**
  Behaves like a String, except valid values can be selected from a list in a dialog. The values that the user can select are defined by **Value.Keys**. As an option, **Label.Keys** can also be given, which will overwrite the display text of the Value.Keys. If Label.Keys are used, the number must match with the number of Value.Keys. It is also possible to define Property-dependent values (see 3.12.1)

- **Type = Date**
3.12 Property

Date Properties can contain a date using the format “YYYYMMDD”. In Input Dialogs with date fields, the date can be chosen from a graphical calendar.

- **Type = Time**
  Time Properties are used to support time specifications in an Input Dialog. The Property value will be formatted using the format “HHmmSS”.

- **Type = Numeric**
  Numeric Properties are used to support numeric values. The Input Dialog validates numbers and adds leading zeros if the “Value_Min_Length” attribute of the property is set to “2” or higher.

- **Type = Text Area**
  A Text Area Property can be used to enter formatted text in an Input Dialog. It can be modelled using the “Max_Lines” and “Chars_Per_Line” attributes of this Property. Depending on their values, the Input Dialog generates a multi-line text field. On dialog confirmation, each non-empty line will be filled with blanks, depending on the “Chars_Per_Line” attribute value.

- **Type = Text Array**
  Text Array Properties can be used to store a variable number of values. They can be used as input parameters for TSO/ISPF tools. There are two ways to set the values of a Text Array Property:

  1) Configuring a File Input Parameter for an Input Dialog. The “Structure” attribute must be set to “Table” and at least one “Has Property” relationship must be defined, referencing a Text Array Property.

  2) Configuring a Text Array Property as an input parameter for the Input Dialog. A “Retrieve all values” Action must be referenced by the Property, to retrieve valid values.

  Both scenarios will generate a selection table within the Input Dialog, so that the user can choose which values to apply.

For more information about dialogs and dialog fields, see 6.2 Configuring Input/Output Dialogs.

A further Property attribute that is used in context with dialogs is **Retrieve_All_Values**. Here, as an option, an Action Descriptor can be referenced associated with a Tool which returns a list of all valid values for this Property (one line per value). The action will be called up via a “Retrieve” button in the input dialog, and returns the valid values as a selection list in the combo box field. The Tools of the referenced Action Descriptor have access to all the input fields of the dialog. User inputs can therefore be used as input parameters.
## Data Model Details

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID *</td>
<td>ID *</td>
<td>The ID of the Property.</td>
</tr>
<tr>
<td>Default_Value [O]</td>
<td>String</td>
<td>The standard value of the Property serves as the default entry for input fields. As soon as a Tool has assigned a Property Value to this Property, the default value (for the corresponding type) will no longer be used.</td>
</tr>
<tr>
<td>Special_Property</td>
<td>Selection</td>
<td>Indicates whether this Property has a special meaning (see 3.12.1).</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>The display text of the property in the Properties view.</td>
</tr>
<tr>
<td>Value_Keys [D]</td>
<td>String []</td>
<td>The permitted Property Values for a Property if it is of the “SelectValue” type. If no “Label_Keys” have been defined, the user can select from a list of these values in the input dialog.</td>
</tr>
</tbody>
</table>

*Figure 3.7 Retrieve button in a Filter dialog*

The combo box will be filled with the returned values after the execution of the Retrieve action. Input fields will be included in the Search for parameter values.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label_Keys [D]</td>
<td>String []</td>
<td>Defines the display texts for the “Value_Keys” of this Property.</td>
</tr>
<tr>
<td>Retrieve_All_Values [O]</td>
<td>Action Descriptor</td>
<td>Reference to an Action Descriptor that returns the valid values of this Property.</td>
</tr>
<tr>
<td>Upper case [O]</td>
<td>Boolean</td>
<td>The value of this Property is always converted to upper case. Standard value: false</td>
</tr>
<tr>
<td>Type</td>
<td>Selection</td>
<td>The Type of the Property. Permitted values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ String</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ Boolean</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ SelectValue</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ Date</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ Time</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ Numeric</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ Text Area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ Text Array</td>
</tr>
<tr>
<td>Max_lines</td>
<td>Integer</td>
<td>Limits the number of lines of a Text Area Property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard value: 0 (unlimited)</td>
</tr>
<tr>
<td>Chars_per_line</td>
<td>Integer</td>
<td>Defines the length of every line of a Text Area Property. Non-empty lines will be filled with blanks on dialog confirmation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard value: 80</td>
</tr>
<tr>
<td>Value_max_length [O]</td>
<td>Integer</td>
<td>Limits the permitted length of an entry within an input field for this Property to a maximum value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard value: 0 (unlimited)</td>
</tr>
<tr>
<td>Value_min_length [O]</td>
<td>Integer</td>
<td>Defines a minimum length for an entry within an input field for this Property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard value: 0</td>
</tr>
<tr>
<td>False Value [O]</td>
<td>String</td>
<td>Only relevant for Properties of type Boolean; specifies the value returned when the Property value is false.</td>
</tr>
<tr>
<td>True value [O]</td>
<td>String</td>
<td>Only relevant for Properties of type Boolean; specifies the value returned when the Property value is true.</td>
</tr>
</tbody>
</table>

Table 3-41 Attributes of a Property
3.12.1 Special Properties

A Property can have one of several special meanings in the runtime context. These can be modelled by the „Special_Property“ attribute:

- **Element Type** – Indicates whether the Property Value of this Property contains a match name (see 3.6). This information is used for Element creation to associate the correct Element Type.

- **Element Suffix** – Indicates whether the Property Value of this Property contains a file suffix. This information is used by edit/browse actions.

- **Marker Line Number** – Indicates whether the Property Value of this Property contains a line number used to mark the line in an Eclipse Editor.

- **Property Group** - Indicates whether the Property Value of this Property contains the name of a Property Set.

3.12.2 Property dependent Values

Property dependent values can be specified for “Select Value” Properties. Using a specific syntax, the administrator can define “Value Keys” for a Property that depends on another Property value:

```
<Start_Ref><Dependent Property ID><End_Ref>=<Dependent Value>:<Value>
```

*Example:*  
`&PROP_Language.=COBOL:COBPROC1`  
*Means:* If the Property with the ID PROP_Language has the value “COBOL”, one of the Value Keys is COBPROC1.

Property dependent values only apply to dialog fields within an Input Dialog.

3.13 Tool Descriptor

A Tool Descriptor precisely describes a Tool of any platform. Tool Descriptors can only be referenced by Action Descriptors, where they are defined in a fixed sequence. The faulty running of a Tool prevents the execution of the next Tool. Output parameters of Tools can be used as input parameters for subsequent Tools.

The basic requirement for the implementation of a Tool is an implementation of a Tool Interface. The connection between Tool Descriptors and communication interfaces will be set up via the **ToolType** attribute. By means of the value entered here, an Eclipse Extension\(^1\) must be found which implements this Tool Type.

The following valid Tool Types are available:

- **ISPFCOMMAND**  
  Used to implement z/OS ISPF Tools (see 6.3 and, for modelled examples, 7.2.4) without ISPF dialog functions

\(^1\) Eclipse Extension: Implementation of a special interface of Eclipse plug-ins (Extension Point).
3.13 Tool Descriptor

- **ISPFDIALOG**
  Used to implement z/OS ISPF Tools which contain ISPF dialog functionality (e.g. DISPLAY PANEL, TBDISPL)

- **ISPF_PANEL_COMMAND**
  Used to implement z/OS ISPF panel commands in the context of a running ISPF dialog application.

- **SWTDIALOG**
  Used to define Input or Output dialogs (see 6.2) in an Eclipse environment. The dialogs are dynamically created from the model definitions.

- **PLUGIN**
  Used to call external Eclipse plug-ins (see 6.1)

- **OSCOMMAND**
  Used to call local Commands (e.g. Windows Commands)

If a Tool is to make changes to one or more Elements on the remote system, this must be specified in the Tool Descriptor, so that the Client application can react to such operations. This will be modelled via the **ResourceProcessing** attribute.

All configurable Resource Operations are based on the following rules:

- Resource Operations are supported for Elements.
- If an action generates new Elements in the Element Table view, these will be inserted either under the selected Resource or, if the list has been sorted, according to the sorting criteria.
- Inserted Elements must not match the Filter Criteria after inserted but refreshing the list will show only elements consistent with the Filter Criteria.
- Tools that insert new Elements must, as a minimum, return all the Key Property Values and the match name of the corresponding Element Type.
- Property Values that will be returned when inserting a new Element will be assigned to the new Element insofar as it contains the Properties. Any other returned Property Values will be assigned to the Element List or the Table rows.
- Resource Operations will not be carried out if the corresponding Tool execution returns an error code.

The following Resource Operations can be modelled for a Tool Descriptor:

<table>
<thead>
<tr>
<th>Resource Operation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>COPY</td>
<td>A new resource of the selected type will be created, and all the Property Values will be copied first. Property Values that are returned by the Tool will be assigned to the copied resource.</td>
</tr>
<tr>
<td>DELETE</td>
<td>Removes the selected resource from all lists. It does not necessarily mean that the remote resource is deleted. This depends on the remote tool.</td>
</tr>
<tr>
<td>INSERT</td>
<td>Inserts a new resource into the corresponding view.</td>
</tr>
<tr>
<td>MOVE</td>
<td>COPY and DELETE the selected resource.</td>
</tr>
<tr>
<td>READ</td>
<td>Standard value. Does not make any changes to resources.</td>
</tr>
</tbody>
</table>
Chapter 3  Data Model Details

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFRESH</td>
<td></td>
<td>Updates all Property Values of the selected resource by means of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Get_All_Properties action assigned to the Element Type.</td>
</tr>
<tr>
<td>UPDATE</td>
<td></td>
<td>Updates the selected resource with the values returned by the Tool. Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Property Values cannot be updated.</td>
</tr>
<tr>
<td>TOOL_DEFINED</td>
<td></td>
<td>The rules which Resource Operations are to be used for this Tool Descriptor are done in the attached Tools (see 6.4.6) not in the model. This makes dynamic implementation of Resource Operations possible. It also allows dealing with more than one resource.</td>
</tr>
</tbody>
</table>

Table 3-42 Possible Resource Operations of a Tool Descriptor

Long-running remote tools can be carried out in the background as a “Batch Job”, however. This can be modelled via the Run_As_Job attribute and is supported by ISPF Tools.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID *</td>
<td>ID *</td>
<td>The ID of the Tool Descriptor.</td>
</tr>
<tr>
<td>Refresh_Resource</td>
<td>Boolean</td>
<td>Indicates whether the selected resource should be updated after the successful execution of the Tool. This means that the “Get_All_Properties” action of the Element is always called after a successful execution of the Tool (see 3.6). Standard value: false</td>
</tr>
</tbody>
</table>
| Resource_Processing      | Selection        | Defines the operations that the Tool carries out with resources, so that the model interpreter can react to this. Permitted values:  
  - COPY  
  - DELETE  
  - INSERT  
  - MOVE  
  - READ (standard value)  
  - REFRESH  
  - TOOL_DEFINED  
  - UPDATE |
| Run_As_Job [UC]          | Selection        | Indicates whether the Tool should run in the background. Permitted values:  
  - NO  
  - Modelled job submit  
  - Tool job submit |
| After_Job_Action [D]     | ActionDescriptor | Reference to an Action Descriptor that supports an Action after the Job end in the context menu of the job list. |
3.13 Tool Descriptor

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrated_Tool</td>
<td>Selection</td>
<td>Declares this Tool Descriptor to be an “Integrated Tool Descriptor”. Integrated Tools are tools which are available and supported within the Client. You will find a detailed description of all integrated tools in chapter 3.13.3.</td>
</tr>
<tr>
<td>ToolType</td>
<td>String</td>
<td>Defines the type of the Tool via a logical name. The link to the implementation of the Tool communication interface will be set up via this name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Permitted values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ISPFCOMMAND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ISPFDIALOG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- ISPF PANEL_COMMAND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- PLUGIN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- SWTDDIALOG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- OSCOMMAND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In the case of additional implemented communication interfaces, other permitted values can be added.</td>
</tr>
<tr>
<td>Specific_Attribute [C]</td>
<td>SpecificAttribute</td>
<td>Allows the definition of new attributes that are not predefined in the data model.</td>
</tr>
<tr>
<td>ISA_&lt;tooltype&gt; [C]</td>
<td>ISA_&lt;tool type&gt;</td>
<td>Bundles together the platform dependent attributes of a Tool Type (see 3.13.3).</td>
</tr>
</tbody>
</table>

Table 3-43 Attributes of a Tool Descriptor

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Cardinality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has_Input_Parm</td>
<td>0..*</td>
<td>Relationship to a Property. The Property Value is passed as an input parameter to the tool.</td>
</tr>
<tr>
<td>Has_Output_Parm</td>
<td>0..*</td>
<td>Relationship to a Property. The Property Value is passed as an output parameter to the tool.</td>
</tr>
</tbody>
</table>

Table 3-44 Relationships of a Tool Descriptor

3.13.1 Relationship: Tool has Input Parameter

The “Tool has Input Parameter” relationship describes exactly one input parameter of a Tool. The reference of the relationship is a Property whose Property Value can be passed as parameter to the Tool at runtime.

The Parameter_From attribute determines where the value of the parameters should be obtained from. There are 3 ways of determining the origin of parameters:
Parameter From = Context
The context includes all Property Values that are available at the start of the execution of an action, and does not change during the Tool sequence. If you carry out an action on an Element, for example, all the Property Values of the Element, the Element List in which it is located and the resources where is it found (e.g. Filter, Element) are available. The Properties view at runtime is a good way of obtaining an overview of the available Property Values of a selected resource.

Parameter From = Previous_Tool
This setting is intended for the transfer of parameters between Tools that will be temporarily managed in a Parameter Memory. No Property Values from this memory are available to the first Tool in a Tool sequence. As soon as a tool creates an output (see 3.13.2), the output properties and their values are available for subsequent Tools. In the Parameter Memory, only one Property Value will be saved for a Property. A Tool that outputs a Property Value for a Property that already exists in the memory will replace this value (see Figure 3.8).

Parameter From = User Input
User Input means that the parameter should be entered by the user during the execution of the Tool. This is used in Tools with a user interface, such as the input dialog. In this case, parameter value is transferred to the Tool.

---

**Figure 3.8 Transfer of Parameters between tools**

The “Parameter Memory” is the temporary storage area for output Tool parameters. The values “123”, “ABC” and “XYZ” are the concrete outputs of the respective Tools and will be saved as Property Values. The illustration shows how the Property PROP_2 has been referenced for two Tools as an output parameter. The second output (XYZ) overwrites the first (ABC).

---

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetIDFile</td>
<td>[D]</td>
<td>File Descriptor. Reference to a File Descriptor. Only important if the Type Attribute has been set to “File”.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TargetIDParm [D]</td>
<td>Property</td>
<td>Reference to a Property. Only important if the Type Attribute has been set to “Element” or “String”.</td>
</tr>
<tr>
<td>Default_Value [O]</td>
<td>String</td>
<td>The standard value of the Parameter. This value is only used if no other Property Value can be found for the referenced Property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Has no meaning for File parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initialises a field in an input dialog.</td>
</tr>
<tr>
<td>Parameter_from</td>
<td>Selection</td>
<td>Determines from where the Property Value of the Parameter property should be obtained.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Permitted values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ Context (default)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ Previous_Tool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ User_Input</td>
</tr>
<tr>
<td>SeqNo * [U]</td>
<td>Integer</td>
<td>Position of the Parameter. Depending on the ToolType, this can be important for the correct execution of the Tool. In dialogs, the sequence numbers are used to position the dialog fields.</td>
</tr>
<tr>
<td>Static_Value [O]</td>
<td>String</td>
<td>Sets a fixed value for the parameter. This can only be overwritten via Parameter Override (see 3.4.3.1).</td>
</tr>
<tr>
<td>Type</td>
<td>Selection</td>
<td>The Type of the Parameter. A Property or a File Descriptor must be referenced on the basis of this attribute.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Permitted values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ Element (+ TargetIDParm)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ File (+ TargetIDFile)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>➢ String (+ TargetIDParm)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Standard value: String</td>
</tr>
<tr>
<td>UI_Category [O]</td>
<td>String</td>
<td>Categorizes the parameter in dialogs. Categorized parameters are enclosed in a labelled frame. Successive parameters with the same categories will be enclosed by the same frame and ordered by sequence number.</td>
</tr>
</tbody>
</table>
### Attributes of the "Tool has Input Parameter" relationship

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI_Tab [O]</td>
<td>String</td>
<td>Allows the assignment of parameters to Tabs. Tabs are individual dialog pages which are activated by clicking the tab label at the top of the dialog. Successive parameters with the same tabs will be shown in the same tab page. This works only for Input Dialogs (see 6.2.1).</td>
</tr>
<tr>
<td>UI_Editable</td>
<td>Boolean</td>
<td>Indicates whether the parameter can be edited in input dialogs. Fields are never editable in an output dialog. Standard value: true</td>
</tr>
<tr>
<td>UI_Generic_Allowed</td>
<td>Boolean</td>
<td>Indicates whether wildcard characters (see 3.2) are permitted as user input. If false, the input dialog cannot be confirmed if wildcard characters are present in the input field. Standard value: false</td>
</tr>
<tr>
<td>UI_Mandatory</td>
<td>Boolean</td>
<td>Indicates whether the field is mandatory in input dialogs. If true, the dialog cannot be confirmed when the field contains no values. Mandatory fields are marked with an asterisk *. Standard value: true</td>
</tr>
<tr>
<td>UI_Form [O]</td>
<td>String</td>
<td>Specifies formatting information for the field in input dialogs. Permitted values so far are: BOLD, HIDDEN, RED, GREEN, YELLOW, BLUE, GRAY</td>
</tr>
<tr>
<td>UI_Help_Text [O]</td>
<td>String</td>
<td>Specifies help text for the field in input dialogs which is displayed when the question mark button beside the field is pressed or the F1 key is pressed in the field.</td>
</tr>
<tr>
<td>UI_Tool_Tip [O]</td>
<td>String</td>
<td>Specifies a tool tip for the field in input dialogs.</td>
</tr>
<tr>
<td>Enabled Condition [O]</td>
<td>Enabled Condition</td>
<td>Relationship to a Parameter Enabled Condition which allows defining the conditions which are checked at runtime. This condition must be true to use this input parameter (see 3.1 for details how to define an Enabled Condition).</td>
</tr>
</tbody>
</table>

Table 3-45 Attributes of the "Tool has Input Parameter" relationship

### 3.13.2 Relationship: Tool has Output Parameter

Output parameters of Tools are described by the “Tool has Output Parameter” relationship. A Property is referenced that should be assigned to the output value of a Tool.

The output parameters of a Tool will be temporarily saved during the execution of an action, so that subsequent tools can use them as input parameters (see Figure 3.8).

The way in which the output of a Tool should be interpreted can be modelled via the Type attribute. Depending on the type, a Property or a File Descriptor must be referenced via the relationship.
3.13 Tool Descriptor

- **Type = String**
  Standard value. The Output will be assigned as Property Value of the referenced Property.

- **Type = Element**
  Only relevant for “TOOL-DEFINED” Resource operations. All output parameters which are Element properties have to be modelled with the Type “Element” (see 6.4.6).

- **Type = File**
  Tool output that is modelled with the “File” type will be interpreted as a file name. You must reference a File Descriptor. If a physical name (`Name_Physical`) is modelled for the referenced File Descriptor, it will be temporarily overwritten. The output parameter will contain the name of the file and the referenced File Descriptor defines the type and the structure of the file.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TargetIDFile [D]</td>
<td>File Descriptor</td>
<td>Reference to a File Descriptor. Only important if the Type Attribute has been set to “File”.</td>
</tr>
<tr>
<td>TargetIDParm [D]</td>
<td>Property</td>
<td>Reference to a Property. Only important if the Type Attribute has been set to “Element” or “String”.</td>
</tr>
<tr>
<td>Label [O]</td>
<td>String</td>
<td>The display text of the parameter in the output dialog.</td>
</tr>
<tr>
<td>SeqNo * [U]</td>
<td>Integer</td>
<td>The position number of the parameter in the output dialog (1 = at the very top).</td>
</tr>
<tr>
<td>Type</td>
<td>Selection</td>
<td>Determines how the output parameter should be interpreted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Permitted values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Element</td>
</tr>
<tr>
<td></td>
<td></td>
<td>File</td>
</tr>
<tr>
<td></td>
<td></td>
<td>String</td>
</tr>
<tr>
<td>To_Output_Dialog</td>
<td>Boolean</td>
<td>Indicates whether the Parameter should be displayed in the output dialog at the end of the action. This only takes place if there is an output dialog modelled at the end of the Tool sequence.</td>
</tr>
<tr>
<td>UI_Category</td>
<td>String</td>
<td>Categorizes the parameter in the output dialog. Categorized parameters are enclosed in a labeled frame. Successive parameters with the same categories will be enclosed by the same frame in the output dialog.</td>
</tr>
</tbody>
</table>

Table 3-46 Attributes of the “Tool has Output Parameter” relationship

3.13.3 Integrated Tools

Integrated Tools are standard tool solutions for functions integrated in the product. They can be flexibly attached to actions. The following sections contain a description of the Integrated Tools that are available, including the rules as to how they are to be modelled.
Chapter 3  Data Model Details

3.13.3.1  Edit Opener

The Edit Opener Tool is a component of the Edit action. It is used to open any file in the corresponding editor. The prerequisite for this is that the Source to be edited is in the local workspace in the Eclipse project.

In an action, the Integrated “Local File Locator” Tool must precede the Edit Opener Tool if the local file has been created with the “Local File Creator”:

The Edit Opener can also be used on temporary local files, however. In this context, it is intended, for example, for Remote Edit scenarios, in which the Save and Close actions will be used.

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to EditOpener.
- A file input parameter must be modelled. The associated File Descriptors Type must be “Workspace File” and will typically correspond to the output parameter of the Integrated Tools “Local File Locator” or “Local Temp File Creator”.

3.13.3.2  Edit Closer

The Edit Closer Tool is typically a component of the Close action. The Tool checks whether the file for the given file input parameter is still open in the Editor, and, where necessary, closes the editor. If the editor contains unsaved changes, the user will be asked if he wants to save first.

For safety’s sake, this Tool must also always be integrated into an Element action if the action must not be executed when the Editor is open, for example, during a Check-In action.

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to EditCloser.
- A file input parameter must be modelled. The Input Parameter normally corresponds to that of the Integrated Tool Edit Opener.

3.13.3.3  Edit Cleanup

The Edit Cleanup Tool is used to clean up temporary information that has arisen during a local Edit. The main action is to delete the associated File from the workspace.

If a file is temporarily edited in the context of an action, the Edit Cleanup is typically a component of the Close action. If a persistent local file is created with the help of the Local File Creator, another action will usually be used (e.g. Check-In) that contains the Edit Cleanup.

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to EditCleanup.
- A file input parameter must be modelled, which references the local file to be deleted. Usually a File Descriptor will be used, which was the output parameter of an earlier executed Local File Creator or Local Temp File Creator.
3.13.3.4 Error Feedback

The Error Feedback Tool analyzes the Error Feedback File that is generated from Remote Syntax Check or a Remote Compile, and places the messages contained in it into an internal list that will be displayed in the Error List view.

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to ErrorFeedback.
- A file input parameter must be modelled, which references the Error Feedback XML file of the Remote Compiler.

3.13.3.5 Local File Locator

The Local File Locator Tool is used to locate the file for the selected Element that is stored in the local workspace by the Local File Creator. The path of the file will be transferred to the Property from the Workspace File Type that is given as an Output Parameter. This Output Parameter is an Input Parameter for other Integrated Tools (e.g. Edit Opener, Edit Closer).

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to LocalFileLocator.
- The Local File Creator Tool works only for Element Actions (see 3.6.1), which means that an Element must be selected during the execution of this tool.
- A file output parameter must be modelled, which references the local file for the selected Element.

3.13.3.6 File Compare

The File Compare Tool is used to open two files in an Eclipse compare editor. The tool can be modelled for only comparing the two files or for merging them into one file. In either case, two file input parameters must be modelled, representing the two files. The left side of the editor (which contains the file of the first file input parameter) becomes editable as soon as a file output parameter is modelled. Its targeted file will contain the contents of the left side of the editor and is intended to be used for save/close actions (see 3.4). If the implemented compare editor supports syntax highlighting (the standard Eclipse compare editor doesn't), a file suffix can be modelled by an additional string input parameter.

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to FileCompare
- Two file input parameters are mandatory for configuring a File Compare
- An optional string input parameter can be modelled for the file suffix
- An optional file output parameter can be modelled to refer to a file containing the contents of the left editor part. It can be used by a following save- or close action.

3.13.3.7 File Upload

The File Upload Tool transfers a file from one platform to another.
Chapter 3  Data Model Details

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to FileUploader.
- Two file input parameters must be modelled. The first one must reference the source of the file transfer while the second one references the target file.

3.13.3.8  SubmitJCL

The Submit JCL Tool can be used to submit JCL from a file or an active editor.

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to SubmitJCL.
- To submit JCL out of an active editor, no input parameters must be modelled (recommended to be used within an Editor Action (see 3.9.1)).
- To submit JCL from a file, a file input parameter must be modelled.

3.13.3.9  Local File Creator

The Local File Creator Tool is used to create a local file permanently in the Eclipse project in the workspace. The local file will be initialized by the contents of the file referenced by the Input Parameter that was also transferred as a Property of the File type. The path information for the local file will be internally flagged by the selected Element and can be used later on with the Integrated Tool Local File Locator.

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to LocalFileCreator.
- The Local File Creator Tool works only for Element Actions (see 3.6.1), which means that an Element must be selected during the execution of this tool.
- A file input parameter must be modelled, which references the Source file on the Remote System.

3.13.3.10  Local Temporary File Creator

This Tool is used to temporarily create a local file in the Eclipse workspace. The local file will be initialised by the contents of the file referenced by the Input Parameter that was also transferred as a Property of the File type. The path information for the local file will also be returned as a File Descriptor by the Tool, and can be used by later Tools of the same action as an Input. The file name and the suffix of the file can be determined through input parameters.

The file must be explicitly deleted with the Edit Clean Up Tool.

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to LocalTempFileCreator.
- A file input parameter must be modelled, which references the Source file on the Remote System.
- The second optional Input Parameter is of type String. Its value will be used for the file name of the temporarily stored file.
3.13 Tool Descriptor

- The third optional Input Parameter is of type String and requires the existence of the second parameter. The value is used for the file suffix.
- A file output parameter must be modelled, which will contain the file name of the created temporary file after the execution of the Tool.

3.13.3.11 Local Syntax Check

**Note:** this integrated tool is only supported if a local language compiler is installed which supports Error Feedback.

The Local Syntax Check Tool carries out a local syntax check on a file in the workspace. This requires that a Property Set is assigned to the associated local or remote Eclipse resource. The current product version does not include this feature, but it is possible to use this feature in combination with 3rd party products.

Property Sets define the Compile Parameters that will be needed for the local syntax check.

**Configuration notes:**

- The Integrated Tool attribute of the Tool Descriptor must be set to LocalSyntaxCheck.
- A file input parameter must be modelled, which references the local file that needs to be checked.

3.13.3.12 Property Set Creator

**Note:** this integrated tool is only supported if Eclipse remote resources can be linked to Property Sets. The current product version does not include this feature, but it is possible to use this feature in combination with 3rd party products.

The Property Set Creator creates a **Property Set** with a logical name and adds the SYSLIB attribute used for the Syntax Check support.

**Configuration notes:**

- The Integrated Tool attribute of the Tool Descriptor must be set to PropertySetCreator.
- Three input parameters must be modelled:
  1. The full system library (SYSLIB) path
  2. The programming language (COBOL, PL1)
  3. The logical name of the Property Set

3.13.3.13 Property Set Allocator

**Note:** this integrated tool is only supported if Eclipse remote resources can be linked to Property Sets. The current product version does not include this feature, but it is possible to use this feature in combination with 3rd party products.

The Property Set Allocator associates a Property Set to an Element.

**Configuration notes:**
Chapter 3  Data Model Details

- The Integrated Tool attribute of the Tool Descriptor must be set to `PropertySetAllocator`.
- An Element must be selected
- Two input parameters must be modelled:
  1. File Input Parameter which contains the full path of the selected file
  2. Logical name of the Property Set that should be associated

3.13.3.14 Property Set Remover

**Note:** this integrated tool is only supported if Eclipse remote resources can be linked to Property Sets. The current product version does not include this feature, but it is possible to use this feature in combination with 3rd party products.

The Property Set Remover removes the Property Set with the given logical name.

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to `PropertySetRemover`.
- A String input parameter must be modelled. Its value must be the name of an existing Property Set.

3.13.3.15 Refresh Dependencies

The Refresh Dependencies Tool is used to determine the dependencies (Copybooks) of the associated Elements. The dependent Elements will be downloaded from the remote system and will be saved in the in the workspace under the path of the associated Element in a folder with the name of the Element.

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to `RefreshDependencies`.
- A file input parameter must be modelled that references the Error Feedback XML file of the remote Compiler output.
- A second file input parameter must be modelled which references an MVS partitioned dataset (PDS) which contains all Copybooks referenced by the Element.

3.13.3.16 Remote Browse

The Remote Browse Tool is used to open the remote file belonging to the selected Element in the Editor in Read mode (Browse).

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to `RemoteBrowse`.
- A file input parameter must be modelled to reference the file on the Remote System that needs to be opened in browse mode.
3.13.3.17 Remote Edit

The Remote Edit Tool is used to open the remote file belonging to the selected Element for editing in the Editor.

Configuration notes:

- The Integrated Tool attribute of the Tool Descriptor must be set to **RemoteEdit**.
- A Property of the File type must be modelled as an input parameter to reference the file on the Remote System that needs to be opened in an editor.

3.13.3.18 Example Configuration

In the following, the use of some integrated tools will be presented as examples.

**Scenario:** Checkout → local Edit → Checkin

Checkout:

- ✷ Action Descriptor ACTCheckOut
  - Tool #1: TOOL_REX_CheckOut
  - Tool #2: TOOL_INT_LocalFileCreator

Local Edit

- ✷ Action Descriptor ACTEdit
  - Tool #1: TOOL_INT_LocalFileLocator
  - Tool #2: TOOL_INT_EditOpener

Checkin:

- ✷ Action Descriptor ACTCheckIn
  - Tool #1: TOOL_INT_LocalFileLocator
  - Tool #2: TOOL_INT_EditCloser
  - Tool #3: TOOL_INT_FileUploader
  - Tool #4: TOOL_REX_CheckIn
    - Parameter Override Upload only (Static Value: - > False)
  - Tool #5: TOOL_INT_EditCleanUp

3.13.4 Tool Descriptor ISA

An ISA (is an) attribute generally consists of several attributes brought together and contains all the platform-dependent information about the Tool.

The following ISA attributes are supported:

- **ISA_Plugin**
- **ISA_ISPFTool**
- **ISA_ISPFPanel**
- **ISA_ISPFPanelCommand**
3.13.4.1 Tool Descriptor ISA Plug-In

ISA_Plug-In is a complex attribute that can be modelled for a Tool Descriptor if the Tool to be attached is an Eclipse Plug-In that implements the specific interface (IToolExecutor).

The only mandatory attribute is the Plug_In_ID. When executing this kind of Tool Descriptor, an installed Plug-In with this ID is searched. If no Plug-In is found or the Plug-In is incorrectly implemented, the action is terminated with an error message.

The attributes Description and Title

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plug_In_ID</td>
<td>String</td>
<td>The ID of the Eclipse Plug-In.</td>
</tr>
<tr>
<td>Description</td>
<td>String</td>
<td>A description of the Plug-In or instructions to the user.</td>
</tr>
<tr>
<td>Title</td>
<td>String</td>
<td>The Title or Name of the Plug-In (freely selectable).</td>
</tr>
</tbody>
</table>

Table 3-47 Attributes of an ISA_Plugin

3.13.4.2 Tool Descriptor ISA ISPF Tool

ISA_ISPFTool is a complex attribute that can be modelled for a Tool Descriptor. It contains attributes needed to call a module or a procedure in an ISPF environment.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program_ID</td>
<td>String</td>
<td>Unique name by which the attached Host System can find the corresponding Tool.</td>
</tr>
<tr>
<td>In_Parm_Separator</td>
<td>String</td>
<td>Separator to be used when transferring Parameter values. If the value contains blanks, it must be enclosed in double quotes.</td>
</tr>
<tr>
<td>Out_Parm_Separator</td>
<td>String</td>
<td>Separator to be used for the output parameters. The value must be enclosed in double quotes.</td>
</tr>
<tr>
<td>Invoke_Form</td>
<td>Selection</td>
<td>Invocation form for the ISPF Tools (see 6.3)</td>
</tr>
<tr>
<td>ISPF_Panel</td>
<td>0,...,n</td>
<td>A relationship to the ISPF panel dialog definitions. This relationship is mandatory if the tool descriptor is of type ISPFDIALOG</td>
</tr>
</tbody>
</table>

Table 3-48 Attributes of an ISA_ISPFTool

The relationship to ISPF_Panel supports the following attribute:
### 3.13 Tool Descriptor

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target_ID</td>
<td>Tool Descriptor</td>
<td>A Tool Descriptor of type SWTDIALOG which is used to map an ISPF panel to an Eclipse dialog.</td>
</tr>
</tbody>
</table>

*Table 3-49 Attribute of the ISPF_Panel relationship*
3.13.4.3 Tool Descriptor ISA ISPF Panel

ISA_ISPFPanel is an attribute that can be configured for a Tool Descriptor.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program_ID</td>
<td>String</td>
<td>A reference to the ISPF module or procedure which calls the ISPF panel.</td>
</tr>
<tr>
<td>Panel Name *</td>
<td>String</td>
<td>The name of the ISPF panel</td>
</tr>
<tr>
<td>Has_Panel_Field</td>
<td>0..*</td>
<td>Relationship to a Property. The Property is used to exchange its value between the ISPF Panel and the Eclipse Dialog. This relationship is optional. If not used the parameters exchanged between the Client Dialog and the ISPF panel are taken from the dialog input parameters.</td>
</tr>
</tbody>
</table>

Table 3-50 Attributes of an ISA_ISPFPanel Attribute

The relationship Has_Panel_Field supports the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seq_No*</td>
<td>Integer</td>
<td>Position of the panel field in the transferred parameter list from the ISPF panel (ISPF variable TAUPARMS)</td>
</tr>
<tr>
<td>Target_ID *</td>
<td>Property</td>
<td>Relationship to a Property.</td>
</tr>
</tbody>
</table>

Table 3-51 Attribute of the Has_Panel_Field relationship

3.13.4.4 Tool Descriptor ISA ISPF Panel Command

ISA_ISPFPanelCommand is a complex attribute that can be modelled for a Tool Descriptor with the Tool Descriptor Type “ISPF_PANEL_COMMAND”. It configures the execution of a panel or line command within an ISPF panel.

ISA_ISPFPanelCommand has the following attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Name</td>
<td>String</td>
<td>Value of the command (e.g. I, U, D, B for line commands or INSERT for a panel command).</td>
</tr>
<tr>
<td>Command Variable</td>
<td>String</td>
<td>Property in which the command is expected to be returned. This Property must also be modelled as a ISPF Panel field.</td>
</tr>
</tbody>
</table>
Chapter 4
Setting up the Application Workflow Model

4.1 Master Configuration

Several remote systems with any number of applications can be attached. Exactly one file belongs to each AWM. The applications that will be supported by the corresponding modelled configurations will be defined in a central Master Configuration File for each remote system.

There must be a Master Configuration File on every remote system on which at least one modelled application is located. A unique logical name must be given to it for the remote system.

In addition, the Master Configuration File stores the following information about each application:

- The logical name of the application
- Reference to the central storage location for the AWM
- The version of the current AWM.
- Locally cached data that is to be deleted when the AWM is updated.
- Static Property Values
- Optional the restriction to specific USERIDs which are allowed to access a modelled application.

With this information, the version of the currently available AWM can be compared with the installed AWM every time a user calls the first time an remote AWM action after starting Eclipse. If the user is working with an old AWM, he will be asked to update it. If persistent data will have to be deleted for consistency when changing a AWM version, back-up copies will automatically be created in the workspace.

The Master Configuration File will be read by a REXX procedure that must be located under the SYSEXEC or SYSPROC allocation. The name of this REXX procedure is TAUZCAPP.

If the user attaches a new remote system to the Tree View, the new system entry will receive the name returned from the master configuration file. The REXX procedure will also be called up to determine or update all AWM entries which are listed in the Master Configuration File. At this time a specific AWM is not loaded.

This REXX procedure is also called each time the user carries out an action for the first time during an Eclipse session that requires a link to the remote system. The following checks run at this call-up:

- REXX procedure found?
Chapter 4  Setting up the Application Workflow Model

Could this REXX procedure be found on the remote system? If not, the action will be cancelled with an error message.

- Correct system?

Does the system name returned by the REXX procedure match with the system name entry stored in the Eclipse workspace? If not, the action will be cancelled with an error message.

- Correct Application Configuration version?

Does the version of the local AWM for the active application agree with the version of the AWM for this application returned by the REXX procedure? If this is not the case, the user will receive a confirmation dialog in which he will be informed that the action has been cancelled due to the different AWM version. The action will be cancelled when the dialog is confirmed. The currently active application will be unloaded. A back-up of the active application will be carried out in the workspace. Depending on the returned INFO entry, cached files in the workspace inconsistent with the new AWM will be deleted where necessary. The new AWM will then be loaded.

- Master Configuration File present and consistent?

The REXX procedure checks the consistency of the Master Configuration File. If any error is detected here, they will be communicated to the user in an error dialog. Warnings will be written into the Eclipse "log" file.

As standard, the Master Configuration will be expected in the following sequential file:

hlq.ZSERVER.MASTER

If this file name conflicts with the existing naming conventions, the adapted name must be entered into this REXX procedure.

The Master Configuration File consists of lines with key attributes, together with the respective values. The keywords are separated from the values by a colon and at least one space.

As an example, a row in the Master Configuration File will look like this:

System: System_name

Lines beginning with an * are interpreted as comments.

The following table provides an overview of the permitted keywords and their meaning.

<table>
<thead>
<tr>
<th>Key word</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>System</td>
<td>Name of the remote system as it is displayed in the Tree View.</td>
</tr>
<tr>
<td>zExConf</td>
<td>Location of the zExplorer model (see 5.2).</td>
</tr>
</tbody>
</table>
4.1 Master Configuration

<table>
<thead>
<tr>
<th>Key word</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appl</td>
<td>Name of the modelled application as it is displayed in the Tree View.</td>
</tr>
<tr>
<td>Conf</td>
<td>Platform, Path and Name of the AWM.</td>
</tr>
<tr>
<td>Version</td>
<td>Version of the AWM.</td>
</tr>
<tr>
<td>INFO</td>
<td>Information about the data that will be deleted for consistency in the Eclipse Workspace when loading a new version of the AWM.</td>
</tr>
<tr>
<td>User</td>
<td>Limits the corresponding application(s) to one or more RACF users. Multiple users can be separated with spaces.</td>
</tr>
<tr>
<td>Property: &lt;Property ID&gt;=&lt;Value&gt;</td>
<td>Static Property values for an application.</td>
</tr>
<tr>
<td>EndUser</td>
<td>End of an user limitation.</td>
</tr>
<tr>
<td>EndAppl</td>
<td>End of an application description.</td>
</tr>
</tbody>
</table>

Table 4-1 Key words of the Master Configuration

A Master Configuration file can be defined on every z/OS remote System to which a connection can be made, via Eclipse RSE. The logical system name that is entered there must be unique over all systems.

Any number of modelled applications can be listed for a remote system. The Master Configuration File must contain at least one defined block for the description of an application.

The System entry is mandatory. The entries Appl and Conf are mandatory for each application. The Version entry is preset with 1.0 if it is missing. The Info entry is preset with spaces if it is missing.

**Structure of the Conf entry**

The Conf entry consists of the specification of the platform together with the platform-dependent fully-qualified or absolute file name. Permissible platform designations are local mvs and uss.

**Examples for correct Conf data:**
- mvs:'hlq.MODEL.XML(MYCONF)'
- local:'c:\config\config.xml'

**Meaning of the INFO entry**

The INFO entry determines which persistent information held in the Eclipse Workspace will have to be deleted in the case of a version change of the modelled application, for consistency. Permissible information is filters and elements. The information can be combined in any way.

<table>
<thead>
<tr>
<th>INFO entry</th>
<th>Deleted files</th>
<th>Content of the files</th>
</tr>
</thead>
<tbody>
<tr>
<td>filters</td>
<td>ft_filters.xml</td>
<td>all filters in the application</td>
</tr>
<tr>
<td>elements</td>
<td>et_elements.xml, eb_elementsinlists.xml, li_&lt;listid&gt;, lm_Element_Lists.xml</td>
<td>all cached Elements that are assigned to the application and all Element Lists in the application</td>
</tr>
</tbody>
</table>

Table 4-2 INFO entries and their effects
4.2 Working with the AWM Editor

There is a specific GUI-controlled XML Editor available to create and maintain an AWM. The editor offers the following advantages to the user:

- All object types and relationships will be displayed categorized in a clear tree structure.
- The creation of the Object types and relationships is supported via context menus, which allows a preview of the configuration options and avoids errors.
- Validation function that recognizes both XML and semantic errors.
- Attributes are categorized so that, for example, optional attributes are identified.
- Valid values and references can be selected from drop-down lists.
- Brief descriptions for all attributes.
- Editor functions, such as Copy / Paste, Drag and Drop, Undo / Redo enable effective work, even in the case of large modifications.

This chapter explains the use of the editor, from the creation of a new AWM up to the use of the advanced editor functions.

The sample below shows, step by step, how to start in an Eclipse environment

4.2.1 First steps

Start Eclipse. If a “Welcome” window appears when starting for the first time, close it (see Figure 4.1).

In Workbench, change to the Team Developer perspective by clicking the symbol and then Other... (See Figure 4.3). The “Open Perspective” dialog appears, in which the Team Developer perspective can be activated by double-
Create a new project. Use the context menu of the **Navigator View**, with “New → Project …” (see Figure 4.2). A dialog appears in which the Project Type is to be selected. The standard type “Project” is the most suitable here, and can be found under the category “General”. Double-clicking opens a wizard for creating the project. You need only enter a project name such as “Model Configuration” into the top text field and close the wizard with **Finish**.

By right-clicking the newly created project in the Navigator View and “New → Other…” a dialog will be displayed for the creation of new files. The category “AWM” can also be found here, under which the file type “Model Configuration” is located. Double-clicking “Model Configuration” opens a Wizard, which can be closed with **Finish**. A new AWM called "CONFIG.model" appears in the Navigator View, which is then displayed in the Editor (see Figure 4.4).

The AWM Editor shows all modelled types and relationships in a tree structure. Their attributes can be viewed and edited in the Properties view. The Properties View is located in the Team Developer Perspective in the lower left corner. If the Properties View is not visible, it can be displayed by right-clicking “Show Properties View” in the Model Editor.

The first line in the Model Editor represents the resource that is currently being worked on. The next level of the hierarchy will be displayed by clicking to the left of the row. There is a “Document Root” in every AWM, in which global, non-configurable attribute definitions are stored. Another click on shows the next level, “Model Configuration”. Categories for all object types can be defined here.

As categories are **Children** (sub-elements) of “Model Configuration”, they must be created via the function “New Child” from the context menu of “Model Configuration”. To create the “Properties” category, click “New Child → Properties” (see Figure 4.7).
In the same way, a property can now be defined via the context menu of the newly created “Properties” category (“New Child → Property”). A further right-click on “Property” shows that there is no further hierarchical level under Property, as no further children can be defined. New siblings (elements of the same level) of a type can be defined via “New Sibling”, which could be additional properties in the case of “Property”.

If an object type is selected, the attributes will be displayed in the “Properties View” (see Figure 4.5). The values of the attributes can be edited in the right-hand column (Value). Depending on the type of the attribute, values can be entered or selected in the fields of the right column. If standard values exist for attributes, their fields will be automatically filled in.

The first attribute of a Property (ID) requires manual input. The second (type) allows a selection of certain values. These can be selected by mouse click, by clicking twice on the corresponding field in “Value” column.

The “Retrieve All Values” attribute requires a reference to an Action Descriptor. As none has been defined so far in this case, no value can be modelled in the field for the moment.
The “Value Keys” attribute permits the entry of several values. A click on the “Value” field of this attribute causes a button to appear (....). Another click on this button opens a dialog in which the individual values can be added or removed (see Figure 4.6). In addition, new values are entered into the left field and are added using “Add”. Existing values can be removed with “Remove” or be moved with “Up” and “Down”.

If an attribute is selected, help text appears at the lower edge of the window. This is either a short description of the attribute, or, in the case of an incorrect entry, an error message (see Figure 4.8). In the case of long Help texts, it can be useful to hover the mouse over the text in order to have the message displayed in full.

![Figure 4.8 Help text for selected attributes]

A string has been entered into a number field here. Disallowed values cannot be assigned to attributes.

Relationships and complex attributes can be configured in the same way. Both are children of the corresponding object types (e.g. **Element Type, Tool Descriptor**...), are created with “New Child”, and are modelled in the Properties View.

Relationships differ from other types in the Editor through their presentation, however. Their Target ID is shown directly in the Label.

![Figure 4.9 Labelling in the Model Editor]

**Figure 4.7 Creating a “Properties” category**
Any sequence numbers (SeqNo) that occur will also be shown in their label (see Figure 4.9).

The deletion of Object types is carried out from their context menu with the “Delete” function. Deleting an Object type also removes all levels below it. If something is deleted by mistake, it can be recovered using the “Undo” function in the context menu of the Model Editor.

4.2.2 Advanced functions

An important function of the Configuration Editor is Validate. It checks the AWM opened in the Editor for faults and problems.

The validation checks the completeness of all mandatory entries, such as the ID, as well as compliance with consistency rules, such as the uniqueness of the sequence numbers for the input parameters of a Tool. The validation process can be started via the context menu of the Model Editor or by a button in the Eclipse Toolbar (see Figure 4.10) if an AWM is opened in the Editor. If the validation is started via the context menu, however, only the selected object and the levels below it will be examined. The top level in which the Validate function can be used in the context menu is “Document Root”. The button in the toolbar always validates the complete AWM.

If modelling errors are found, an error dialog is opened. Clicking “Details »” in the dialog displays the problems found. Once the dialog has been closed with “OK”, the errors are displayed in the Problems View. The view can be displayed via the Eclipse menu with “Window → Show View → Other… → General → Problems”. Double-clicking a problem will mark the appropriate location in the Configuration Editor.

In addition, the editor supports all the usual shortcut keys and mouse operations.

Example:

*Copy all the Tool Parameters of an existing Tool Descriptor into a new Tool Descriptor with a single mouse click* (see Table 4-3 Table of all important shortcut keys and mouse operations)

<table>
<thead>
<tr>
<th>Function</th>
<th>Call-up (Shortcut key)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Delete</strong></td>
<td>Context menu for all object types (CTRL + Del.)</td>
<td>Removes the selected object types, including all subordinate elements.</td>
</tr>
<tr>
<td><strong>Copy</strong></td>
<td>Context menus incl. Properties View (CTRL + C)</td>
<td>Object types and attribute values can be copied into the clipboard using <strong>Copy</strong>, in order to insert them at another location using <strong>Paste</strong>.</td>
</tr>
</tbody>
</table>
### Table 4-3 Table of all important shortcut keys and mouse operations

<table>
<thead>
<tr>
<th>Function</th>
<th>Call-up (Shortcut key)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cut</td>
<td>Context menu for all object types (CTRL + X)</td>
<td>Works like <strong>Copy</strong> and <strong>Delete</strong>.</td>
</tr>
<tr>
<td>Paste</td>
<td>Context menus incl. Properties View (CTRL + V)</td>
<td>Pastes a copied object type or an attribute value from the clipboard. This works only in locations at which the value to be inserted matches semantically. Example: Insert “Tool Descriptor has Input Parameter” in “Tool Descriptor”.</td>
</tr>
<tr>
<td>Undo</td>
<td>Model Editor context menu (CTRL + Z)</td>
<td>Reverses (Undo) the last action. Repeated use of the function reverses earlier actions.</td>
</tr>
<tr>
<td>Redo</td>
<td>Model Editor context menu (CTRL + Y)</td>
<td>Restores an action that has been reversed (by Undo). Repeated use of the function restores later actions.</td>
</tr>
<tr>
<td>Drag &amp; Drop</td>
<td>(See description)</td>
<td>Permits the movement of object types with the mouse. In order to do this, the object type to be moved must be “dragged” to the desired position while holding the mouse button down (see Figure 4.11). Works like <strong>Cut &amp; Paste</strong>.</td>
</tr>
<tr>
<td>Clone</td>
<td>(Hold down CTRL + Drag &amp; Drop)</td>
<td>Works like <strong>Copy &amp; Paste</strong>, except fewer steps are necessary and the target position of the cloned object can be determined more accurately (through the mouse position).</td>
</tr>
<tr>
<td>Multiple Selections</td>
<td>(CTRL + left mouse button)</td>
<td>Permits the simultaneous selection of several types of objects. Each object that is clicked on will be added to the selection, or, if already selected, will be removed. Several selected object types also support the <strong>Copy/Cut &amp; Paste, Drag &amp; Drop, Delete</strong> and <strong>Clone</strong> functions.</td>
</tr>
<tr>
<td>Mass Selection</td>
<td>(SHIFT + left mouse button)</td>
<td>Selects all the objects between the current selection and the first object that was clicked. Supports <strong>Multiple Selection</strong>.</td>
</tr>
<tr>
<td>Save</td>
<td>Eclipse menu (CTRL + S)</td>
<td>Saves the AWM.</td>
</tr>
</tbody>
</table>
Chapter 4 Setting up the Application Workflow Model

Figure 4.11 Drag & Drop
Chapter 5

Customizing the zExplorer

5.1 Introduction to the zExplorer

The zExplorer provides basic functions to get access to the MVS file system as well as to Batch Jobs. Drag & Drop cross system support for remote files is available as long as this is done within Eclipse.

The zExplorer is designed to be almost fully customizable. Its whole suite of functions is set up by an internal model which makes it possible to easily change existing features or add new ones. The zExplorer is located in the Remote Systems View, which comes with the Target Management(TM) Eclipse framework and can be used after adding a z/Server connection (see Figure 5.1).

![Figure 5.1 The zExplorer within the Remote Systems View](image)

5.2 Modelling the zExplorer

The zExplorer is based on a product integrated model. This integrated model itself is not changeable, but the zExplorer is able to load an external model located on the mainframe. You will find the source of the external zExplorer model in the mainframe installation XML file (name: “zexpconf”). We recommend using this model as a basis for customization. The major restriction in modelling the zExplorer is that it is not possible to define Element Types other than the four existing ones (Dataset, Member, Job, Job Part). The external modelling of the JES Explorer as part of the zExplorer is not supported right now.
To load an external model, the `zExConf` attribute must be configured in the Master Configuration File (see 4.1), specifying the location of the model:

* Enter the location of your zExplorer AWM.
  `zExConf: HLQ.ZSERVER.XML(ZEXPCONF)`

### 5.3 The zExplorer Model

Other than a normal AWM, the zExplorer model must contain certain objects with fixed IDs in order to work properly. These IDs are mapped to several types and functions that are integrated in the zExplorer.

**Example:**
If a dataset filter is expanded at runtime, the Action with the Action Descriptor ID “ACT_FIX_MVS_FilterDataSets” will be executed.

While the object IDs must not be manipulated, the behavior of the objects themselves may be changed.

The IDs always contain the token “FIX” (fixed) or “ZS” (z/Server Tool) so they can be easily distinguished from user defined IDs.

Here is a list of fixed IDs within the zExplorer model:

<table>
<thead>
<tr>
<th>ID</th>
<th>Action Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT_FIX_MVS_FilterDataSets</td>
<td>Filters MVS datasets (expanding a dataset filter). The Property “PROP_FIX_DataSetFilter” will be available in the execution context with the current filter string as its value.</td>
</tr>
<tr>
<td>ACT_FIX_MVS_DeleteDataset</td>
<td>Deletes a dataset. If executed without errors, the selected dataset will be removed from the view.</td>
</tr>
<tr>
<td>ACT_FIX_MVS_RenameDataset</td>
<td>Renames a dataset. If executed, a rename dialog comes up for the user to enter the new name. The action will be initialized with the “PROP_FIX_NewDatasetName” Property and the entered value as its value.</td>
</tr>
<tr>
<td>ACT_FIX_MVS_DeleteMember</td>
<td>Deletes a dataset member. Displays a confirmation dialog before being executed. If executed without errors, the selected member will be removed from the view.</td>
</tr>
<tr>
<td>ACT_FIX_MVS_RenameMember</td>
<td>Renames a dataset member. If executed, a rename dialog comes up for the user to enter the new name. The action will be initialized with the “PROP_FIX_NewMemberName” Property and the entered value as its value.</td>
</tr>
</tbody>
</table>
### 5.3 The zExplorer Model

<table>
<thead>
<tr>
<th>ID</th>
<th>Action Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT_FIX_JES_FilterJobs</td>
<td>Filters JES jobs (expanding a job filter). The Property “PROP_FIX_JobFilter” will be available in the execution context with the current filter string as its value.</td>
</tr>
<tr>
<td>ELE_FIX_MVS_Dataset</td>
<td>Fixed zExplorer Element Types.</td>
</tr>
<tr>
<td>ELE_FIX_MVS_Member</td>
<td></td>
</tr>
<tr>
<td>ELE_FIX_JES_Job</td>
<td></td>
</tr>
<tr>
<td>ELE_FIX_JES_JobPart</td>
<td></td>
</tr>
<tr>
<td>IDDEF_FIX_MVS_DataSet</td>
<td>ID Definitions of the fixed Element Types.</td>
</tr>
<tr>
<td>IDDEF_FIX_MVS_Member</td>
<td></td>
</tr>
<tr>
<td>IDDEF_FIX_JES_Job</td>
<td></td>
</tr>
<tr>
<td>IDDEF_FIX_JES_JobPart</td>
<td></td>
</tr>
<tr>
<td>PROP_FIX_...</td>
<td>Fixed Properties are used for integrated z/Server tools</td>
</tr>
</tbody>
</table>

Table 5-1  List of fixed objects within the zExplorer model

#### 5.3.1 Integrated z/Server Tools

The z/Server offers several high performing tools that can be used in the zExplorer model. They have a unique Tool Descriptor ID and Tool Type.

<table>
<thead>
<tr>
<th>ID and Tool Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID: TOOL_ZS_DEQUEUE</td>
<td>Schedules a DEQUEUE command on the given dataset name.</td>
</tr>
<tr>
<td>Tool Type: ZEXMVS</td>
<td></td>
</tr>
<tr>
<td>ID: TOOL_ZS_LISTDSI</td>
<td>Returns a list of MVS datasets depending on a generic filter string.</td>
</tr>
<tr>
<td>Tool Type: ZEXMVS</td>
<td>The structure of the returned dataset list is defined in the file descriptor <strong>FILE_ZS_LISTDSI</strong>.</td>
</tr>
<tr>
<td>ID and Tool Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ID: TOOL_ZS_LISTMEM</td>
<td>Returns a list of MVS dataset members depending on the given dataset name. The structure of the returned member list is defined in the file descriptor “FILE_ZS_LISTMEM”.</td>
</tr>
<tr>
<td>Tool Type: ZEXMVS</td>
<td></td>
</tr>
<tr>
<td>ID: TOOL_ZS_RECALL</td>
<td>Schedules a RECALL on the given dataset name. The WAIT option can be set through a second input parameter (Boolean, true by default).</td>
</tr>
<tr>
<td>Tool Type: ZEXMVS</td>
<td></td>
</tr>
<tr>
<td>ID: TOOL_ZS_GETJOBS</td>
<td>Returns a list of job parts depending on the given job name filter. The structure of the returned job list is defined in the file descriptor “FILE_ZS_GETJOBS”.</td>
</tr>
<tr>
<td>Tool Type: ZEXJES</td>
<td></td>
</tr>
<tr>
<td>ID: TOOL_ZS_GETJOBPARTS</td>
<td>Returns a list of job parts depending on the given job ID. The structure of the returned job list is defined in the file descriptor “FILE_ZS_GETJOBPARTS”.</td>
</tr>
<tr>
<td>Tool Type: ZEXJES</td>
<td></td>
</tr>
<tr>
<td>ID: TOOL_ZS_BROWSEJOB</td>
<td>Opens a job in an editor depending on the given job ID.</td>
</tr>
<tr>
<td>Tool Type: ZEXJES</td>
<td></td>
</tr>
<tr>
<td>ID: TOOL_ZS_BROWSEJOBPART</td>
<td>Opens a job part in an editor depending on the given job ID and job part name.</td>
</tr>
<tr>
<td>Tool Type: ZEXJES</td>
<td></td>
</tr>
</tbody>
</table>

*Table 5-2 List of z/Server tools within the zExplorer model*
Chapter 6

Attaching Tools

6.1 Attaching Plug-Ins

It is supported to attach your own Eclipse plug-ins as tools. A Plug-In that is to be attached must implement a Tool interface so that it can be started within an action.

A plug-in must be modelled as follows:

Tool Descriptor with ToolType = PLUGIN
Tool Descriptor ISA Plug-In:
  o Plug In ID = my.plugin.id

The Plug-In itself must have an Activator class which implements the Interface de.soforte.taurus.model.IToolExecutor. On execution, the model interpreter will call the execTool(...) method.

6.2 Configuring Input/Output Dialogs

You can configure input and output dialogs to make user inputs possible and to return Tool outputs to the user.

A dialog will be modelled using the following rules:

- Tool Descriptor with ToolType = SWTDialog
- Tool Descriptor ISA Plug-In:
  o Plug In ID = de.soforte.taurus.outputdialog (Output Dialog)
  o Plug In ID = de.soforte.taurus.inputdialog (Input Dialog)
  o The Title defines the title of the dialog (see Figure 6.1).
  o Description defines the message text of the dialog (see Figure 6.1).
- Input parameters generate dialog fields
- Output parameters are not needed
Every input parameter of a Dialog-Tool Descriptor generates a dialog field when executed. The appearance of a dialog field changes depending on whether a Property or a File Descriptor is referenced by a Parameter. This is shown in more detail in the following table:

<table>
<thead>
<tr>
<th>Parameter reference</th>
<th>Dialog field Input Dialog</th>
<th>Dialog field Output Dialog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property (Type = String)</td>
<td>Combo box</td>
<td>Single-line read-only text field</td>
</tr>
<tr>
<td>Property (Type = Boolean)</td>
<td>Check box</td>
<td>Read-only check box</td>
</tr>
<tr>
<td>Property (Type = Select Value)</td>
<td>Combo box with Calendar button and selection of the last user inputs</td>
<td>Single-line read-only text field</td>
</tr>
<tr>
<td>Property (Type = Date)</td>
<td>Time selection field</td>
<td>Single-line read only text field</td>
</tr>
<tr>
<td>Property (Type = Time)</td>
<td>Multi-line text field with “remaining lines” indicator</td>
<td>Multi-line read-only text field</td>
</tr>
<tr>
<td>Property (Type = TextArea)</td>
<td>Multi-line text field</td>
<td>Multi-line read-only text field</td>
</tr>
</tbody>
</table>
### 6.2 Configuring Input/Output Dialogs

#### 6.2.1 Input Dialog

The Input Dialog has many ways of entering, changing or checking parameters for further processing. It generates a dialog field for every input parameter and outputs the value of the field on confirmation in the Parameter Memory. Subsequent Tools will then have access to these parameters (via Parameter from: PREVIOUS_TOOL).

Cancelling an Input Dialog means cancelling the complete action. An Input Dialog is considered to have been cancelled if it is not confirmed with OK.

The behaviour and appearance of the dialog can be modelled more accurately via the attributes of the “Tool has Input Parameter” relationship (see: 3.13.1 Relationship: Tool has Input Parameter):

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Input Dialog field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter From</td>
<td>Makes it possible to fill the dialog field. With the “USER_INPUT” attribute value, the field will not be preset with a Property Value.</td>
</tr>
<tr>
<td>SeqNo *</td>
<td>Position number of the dialog field.</td>
</tr>
<tr>
<td>Default Value [O]</td>
<td>Fills the dialog field with this value or a reference, if (depending on the &quot;Parameter_From&quot; attribute) no value is available to the referenced property.</td>
</tr>
<tr>
<td>Static Value [O]</td>
<td>Sets a fixed value or a reference for this parameter. An input field with “Static Value” cannot be edited.</td>
</tr>
<tr>
<td>Label [O]</td>
<td>Defines the label of the dialog field</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Property (Type = TextArray)</th>
<th>Selection list</th>
</tr>
</thead>
<tbody>
<tr>
<td>A “Retrieve All Values” action is required for the TextArray Property for populating the selection list.</td>
<td></td>
</tr>
</tbody>
</table>

| Property (Type = Numeric) | Comboxbox with number validation and selection of last user inputs. If the attribute “Value_min_length” was set to “2” or higher, leading zeros will be added on dialog confirmation. |

| File Descriptor (Unstructured) | Table in which one row can be marked. All the properties of the marked row will be output. Will be read-only if the “UI Editable” attribute is set to false. |

| File Descriptor (Structure = Table) | Multi-line read-only text field with the option of opening the content in an editor after the end of the dialog. |

|  | Read-only table |

Table 6-1 Effects of specific dialog attributes
Chapter 6  Attaching Tools

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Input Dialog field</th>
</tr>
</thead>
</table>
| UI Category [O] | Groups dialog fields with the same category. Each category can contain multiple columns of dialog fields (one by default). If two or more rows were modelled for a category, parameters in its frame will be arranged in rows (see Figure 6.1). The number of columns of a category can easily be modelled by adding the number in curly brackets at the end of the UI Category attribute.  

*Example:* “Group with 4 columns{4}”  

**Hint:** You need only specify the column number for the first parameter of the group. |
| UI Tab [O]      | Assigns parameters to tabs. Tabs will be created only if at least one parameter has this attribute set. Parameters which don’t have a tab set will be shown in a tab called “General” by default. |
| UI Editable,    | See: Table 3-45 Attributes of the “Tool has Input Parameter” relationship        |
| UI Generic      | Allowed,                                                                          |
| UI Mandatory    |                                                                                   |

Table 6-2 Attributes of the “Tool has Input Parameter” relationship and their effects on the Input Dialogs

6.2.2  Output Dialog

The Output dialog is used for the display of Tool outputs, including their feedback (see 6.4.3).

The Tool outputs to be displayed are determined via the `To_Output_Dialog` attribute, which is set by preceding tools during the modelling of the output parameters (see 3.13.2). The number of dialog fields depends on the Tool Descriptors of an action. It is therefore possible to configure only one Output Dialog Tool Descriptor for all Action Descriptors. The Output Dialog must be referenced as the last Tool Descriptor of an Action Descriptor if it should display all the desired outputs at the end of the action.  

A Tool return code greater than 0 will be displayed in the Output dialog, even if it was not defined in the action.

6.3  Modelling Operating System Commands

(Unable to create table reference in change mode 😞)

Tools can be defined as Operating System Commands (OS Commands) in the workflow model. If executed they will schedule a command on the operating system on which the Workflow Modeller is running.

The command must be specified by using Input Parameters. Several Input Parameters will be concatenated to a single command string, separated by space characters.

If a program or a batch file was called by the OS Command, the Workflow Manager can parse its return values, as long as they obey certain syntax rules (see table 6-3).
### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReturnCode: &lt;returncode&gt;</td>
<td>Return code of the tool (see 6.3.1)</td>
<td>ReturnCode: 12</td>
</tr>
<tr>
<td>Message: &lt;message&gt;</td>
<td>Tool message (see 6.3.1)</td>
<td>Message: File %1 was not found!</td>
</tr>
<tr>
<td>LogMessage: &lt;message&gt;</td>
<td>Log message. Log messages will be shown in the Eclipse Error Log View after the tool was executed.</td>
<td>LogMessage: Creating file “%filename%”…</td>
</tr>
<tr>
<td>OutParm: &lt;parameters&gt;</td>
<td>Output parameters of the tool (see 6.3.1). Must be returned at the very end.</td>
<td>OutParm: value1 value2</td>
</tr>
</tbody>
</table>

*Table 6-4 Valid OS Command return statements*

### 6.3.1 Output Parameters, return code and message

Called OS commands can return a message, a return code, and output parameters (see table 6-5). All statements must be returned in separate lines.

The **return code** indicates whether the called program or batch file was executed successfully or ended with warnings or errors. If the return code is higher than 0 (default return code) a message should also be returned. If the return code is higher than the Max RC (see 3.3) an error dialog box will come up and the action in which the tool was executed will be aborted.

**Messages** returned by the OS command will always show up in a dialog box in the client after the action or tool was executed. As long as the return code is lower than the Max RC, returned messages will be collected and shown after the action is finished.

**Output Parameters** returned by an OS command will be mapped to the modelled Output Parameters of the Tool Descriptor. They will be separated by the space character. If output parameters can contain space characters the “Out Parm Separator” attribute of the “ISA ISPF Tool” complex attribute can be used to define a different separator (temporary work-around).

Output Parameters must be returned at the very end.

*Example: Windows batch file that returns a message, return code and an output parameter:*

```bash
@echo off
...
echo ReturnCode: 4
echo Message: No customers found for the given filter %1.
echo OutParm: %~dp0customers.txt
```

### 6.4 Attaching ISPF Tools

Tools can be identified as ISPF Tools in the AWM (see 3.13.4.2).

In this case, a the Client application communicates with a remote user session running in an ISPF environment. In order for this Tool interface to be used correctly, the TSO/ISPF environment must be correctly configured. You should also read the following:

- z/Server Installation Guide: Setting up the JCL for a User Server
Chapter 6  Attaching Tools

The product is delivered with several REXX procedures that are only to be used in context of the attachment of ISPF Tools. After successful installation of the REXX procedures and configuration of the TSO/ISPF environment, the defined ISPF Tools can be used.

Calling an ISPF Tool can be modelled in three different ways (see 3.13.4.2):

- REXX  (Call ...)  
- CMD   (Select CMD(...)  
- PGM   (Select PGM(...) PARM(...))

Regardless of the call-up type, you can configure whether a Tool should be executed in the ISPF online or in batch.

In this context, online means that the Client carries out the Tool execution synchronously.

Batch means the asynchronous execution of an ISPF tool through submitting a Batch job. Exits must be adapted for batch support (see 6.4.7).

In order to integrate an existing ISPF function into the Eclipse Client, it is usually necessary to develop a tool adapter that regulates the required parameter input/output and file transfers. This tool adapter will be entered as a program or procedure name in the modelled application. The following sub-chapter describes how a tool adapter of this kind is developed.

6.4.1 Tool Adapter as a REXX subroutine

If the invoke form REXX is used when modelling an ISPF Tool, the call-up of the Tool will take place as a REXX subroutine.

We recommend you use this call-up form, as the input and output parameter are then supported in the most flexible way. The REXX TAUTOSAM procedure skeleton can be used as a copy template for a new Tool Adapter.

6.4.1.1 Input Parameters

Parameters are transferred directly to the REXX procedures. The modelled Tool Descriptor defines whether there should be separators between the parameters. The standard separator is the space.

Example 1:
/* Get input parameters from the Client (blank separated list) */  
Parse Arg parm1 parm2 parm3 parm4 parm5 rest

Example 2:
/* Get input parameters from the Client (comma separated list) */  
Parse Arg parm1 '"' parm2 '"' parm3 '"' parm4 '"' parm5 '"' rest

6.4.1.2 Output Parameters and Return code

The REXX Tool Adapter must return both a Return code, and, if modelled, output parameters. As the REXX procedure is called up as a sub-routine from the Client application, the output parameters will also be transferred to the Client application via the EXIT statement in addition to the Return code. The modelled Tool Descriptor defines whether separators should be used between every two parameters for the output parameters.
6.4 Attaching ISPF Tools

The return must take the following form:

```plaintext
EXIT returncode outparmlist
```

There must be a space between the numerical Return code and the parameter list.

**Example 1:**
```plaintext
/* successful tool execution with a blank separated output parameter list */
outparm = 'PARM1 PARM2 PARM3'
exitrc = 0
Exit exitrc outparm
```

**Example 2:**
```plaintext
/* successful tool execution with a comma separated output parameter list */
outparm = 'PARM1,PARM2,PARM3'
exitrc = 0
Exit exitrc outparm
```

### 6.4.2 Tool Adapter as ISPF Service

If the `invoke` forms CMD or PGM are used when modelling an ISPF Tool, the call-up of the Tool will take place via the ISPF SELECT Service, as follows:

- SELECT CMD(toolname parmlist)
- SELECT PGM(toolname) PARM(parmlist)

With CMD, Tools can be called up that are implemented as CLIST, REXX or TSO Command. With PGM, compiled programs are called up.

The parameter transfer takes place according to modelled parameter sequence numbers. The rules and restrictions of the ISPF SELECT Service must be observed, however.

The Tool Adapter must return both a Return code, and, if modelled, output parameters to the Client application. The output parameters must be made available by the Tool in the ISPF Shared Pool Variable named TAUTOPM.

### 6.4.3 Messages

The model definitions assume a successful Tool execution. In principle, if a Tool outputs a message, this can also be modelled as an output parameter.

If the Tool ends with an error or with warnings, probably only a message should be output. To support this, the Tool has to define the specific ISPF Shared Pool Variable TAUTOMS with the desired message.
Chapter 6  Attaching Tools

REXX Example 1:
/* tool execution finished with errors */
outparm = ''
exitrc = 8

tautoms = 'Here is my error message.'
Address ISPEXEC 'VPUT (TAUTOMS) SHARED'
Exit exitrc outparm

REXX Example 2:
/* tool execution finished with warnings */
outparm = 'PARM1 PARM2 PARM3'
exitrc = 4

tautoms = 'Here is my warning message.'
Address ISPEXEC 'VPUT (TAUTOMS) SHARED'
Exit exitrc outparm

6.4.4  Input files

If a file is created within an action that is to be processed as an input by a tool, it requires no special handling. The name of the input file is modelled in the modelled Tool Descriptor as an input parameter. The Tool-Adapter receives the file name as a parameter, and can allocate and edit the file.

6.4.5  Output files

If a Tool creates output files that will later be processed by the Client application, the file names are modelled as output parameters.

REXX Example 1:
/* Create a seq. file and return the dataset name to the Client */
outdsn = CREATE_DSN()    /* subroutine to create the dataset */
Call ADD_RECORDS outdsn  /* subroutine to add records */
outparm = 'PARM1' outdsn 'PARM3'
exitrc = 0
Exit exitrc outparm

REXX Example 2:
/* Create a PDS member and return it to the Client */
outdsn = CREATE_PDS()    /* subroutine to create the PDS */
member = CREATE_MEM(outdsn) /* Create the PDS member */
outparm = 'PARM1' outdsn 'member' 'PARM3'
exitrc = 0
Exit exitrc outparm

6.4.6  Resource Processing

In principle, the modelled application determines the form in which Element information is exchanged with ISPF Tool (see 3.12.1).

The operation that a tool should execute (e.g. READ, UPDATE, INSERT, DELETE, COPY, MOVE) can be explicitly defined in the AWM. For cases in which the specific Tool operation cannot be modelled in advance, the Tool can dynamically define the operation and report back to the Client application (operation: TOOL-DEFINED).
In addition, there are specially modelled Tools that expect the Resource Lists as output.

In the following chapters, the different types of operations will be described in terms of the impact on the programming of the Tool Adapter.

### 6.4.6.1 Resource transfer to an ISPF Tool

If a modelled action that contains an ISPF Tool has an effect on a resource, the Tool will be modelled so that the attributes for identifying the resource will be transferred to the Tool as input parameters.

**REXX example:**
```rexx
/* Get element id (PDS and member name) */
Parse Arg pds member .
```

### 6.4.6.2 Resource update

For an update operation on a resource, the changed Property Values must be returned as output parameters according to the AWM.

**REXX example:**
```rexx
/* Get element type, property set and file suffix for a PDS member */
Parse Arg pds member .
type = GET_ELEMENT_TYPE(pds member)
propgroup = GET_PROPERTY_GROUP(pds member)
suffix = GET_SUFFIX(pds member)
outparm = suffix,propgroup,type
Exit 0 outparm
```

### 6.4.6.3 Creating a Resource

Several of the Tool Operations that can be defined in the AWM lead to the creation of a resource (COPY, MOVE, INSERT). In this case, the output parameters must be modelled so that all the properties for the identification of the resource are included.

**REXX example:**
```rexx
/* Copy a PDS member */
Parse Arg pds member_old member_new .
copyrc = COPY_MEMBER(pds member_old member_new)
outparm = pds member_new
Exit copyrc outparm
```

### 6.4.6.4 Deleting a resource

If the deletion of a resource is requested in the AWM, it is usually sufficient for the Tool to return a Return code.
Chapter 6  Attaching Tools

REXX example:
/* Delete a PDS member */
Parse Arg pds member .
del_rc = DELETE_MEMBER(pds member)
Exit del_rc

6.4.6.5  Creating a Resource List

Used if the creation of an Element List is required.

As a rule, the AWM will be designed so that an output parameter is expected from the Tool that will return the file name of the Resource List. The Resource List must have a structured design. This structure is defined by a corresponding File Descriptor in the AWM.

REXX example:
/* Create a PDS list */
Parse Arg PDS_prefix .
resource_dsn = CREATE_DSN()
resource_dd = ALLOC_DSN(resource_dsn)
Call GET_PDSLIST(PDS_prefix)
Call WRITE_PDS_INFOS(resource_dd)
Call FREE_DSN(resource_dd)
Exit 0 resource_dsn

6.4.6.6  Creation of a list of Property Values

Used if the values of a Property should be determined by an ISPF Tool in the AWM, for example, to offer a selection of the valid values in an input field.

The procedure for creating the ISPF Tool is similar to that for the creation of the Resource List. The Tool creates a file that contains one valid value per line. The generated file name will normally be returned to the Client application as an output parameter.

6.4.6.7  Tool-defined Resource Processing

If Resource Processing is specified as TOOL-DEFINED in the AWM, the tool will be responsible for delivering information about the type of resource processing to the Client application.

TOOL-DEFINED must be used in the following cases:

- When the type of resource processing (e.g. update or Insert) will only become clear during runtime of the Tool, and can therefore not be modelled.
- With generating Tools (e.g. Compile tools) that can create several outputs from a single input resource and the outputs are to be processed by the Client application.

The ISPF Interface already provides a temporary empty ISPF table, in which the Tools can store Resource Information.

The name of the temporary ISPF table is the value of the Shared Pool Variable TAUTOTBI.

The table is not sorted and contains no key columns.
### 6.4 Attaching ISPF Tools

<table>
<thead>
<tr>
<th>Column name</th>
<th>Meaning and valid values</th>
</tr>
</thead>
</table>
| TAUTOFNC    | Resource Operation in the form: operation(<refid>,<REFRESH>,<SELECT>)  
The term in brackets is optional.  
operation = INSERT-ELEMENT | REMOVE-ELEMENT | UPDATE-ELEMENT  
operation default: UPDATE-ELEMENT  
refid = unique reference identifier for subsequent tools within an action (for future use)  
REFRESH – refreshes all the Element Properties.  
SELECT – selects the Element in the Client table or tree view. |
| TAUTOPRM    | Output parameter for a resource specified in the AWM. |

<table>
<thead>
<tr>
<th>Resource Operation</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| INSERT-ELEMENT      | Checks whether the element is already known.  
If the element does not yet exist, it will be created and added to the list in which the Input Element is located.  
If the element already exists, the Element Properties will be updated in line with the Output Parameters of the tool and, where necessary, the Element will be added to the list in which the Input Element is located. |
| REMOVE-ELEMENT      | Removes the Element from all lists. |
| UPDATE-ELEMENT      | Updates an element that already exists with the values returned by the Tool.  
If the Element does not exist in the same list as the Input Element, the Element will not be added (compare INSERT-ELEMENT). |

| Table 6-6 ISPF Interface Table TAUTOTBI for Resource Operations |

Tools can generate rows with the ISPF Table Service TBADD in the ISPF table, which is empty at the time of the Tool call-up. Tools may not close this table with TBEND or TBCLOSE.
### 6.4.7 ISPF Exits

The ISPF interface delivered with the product consists of REXX procedures and ISPF skeletons.

The following describes the procedures and ISPF skeletons that represent Exits and that can or should be adapted to the respective environment.

<table>
<thead>
<tr>
<th>Exit</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAUTOXA1</td>
<td>REXX</td>
<td>Exit for the dynamic allocation of the ISPF environment. This exit procedure is always called directly before a modelled ISPF Tool is called. It receives one parameter, which is assigned to the modelled application (see 3.2). The Exit is typically used to carry out dynamic allocations depending on the modelled input parameter, e.g. in order to be able to test modified applications.</td>
</tr>
<tr>
<td>TAUTOXA2</td>
<td>REXX</td>
<td>Exit for the allocation of (temporary) files. The allocation of temporary MVS files takes place in this Exit.</td>
</tr>
<tr>
<td>TAUTOXCC</td>
<td>REXX</td>
<td>Exit for the integration of the COBOL compiler e.g. for Remote Syntax Check. This exit is used only in context with add-ons for the support of SCM products, in order to support Remote Syntax Check with Error Feedback. For these purposes, the Exit can also be integrated into home-grown Tool attachments as a REXX subroutine.</td>
</tr>
</tbody>
</table>
### 6.4 Attaching ISPF Tools

#### Table 6-8 ISPF Exits

<table>
<thead>
<tr>
<th>Exit</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAUTOXCP</td>
<td>REXX</td>
<td>Exit for the integration of the PLI compiler for Remote Syntax Check. This exit is used only in connection with add-ons for the support of SCM products, in order to support Remote Syntax Check with Error Feedback. For these purposes, the Exit can also be integrated into home-grown Tool attachments as a REXX subroutine.</td>
</tr>
<tr>
<td>TAUTOXCA</td>
<td>REXX</td>
<td>Exit for the integration of ASSEMBLER for Remote Syntax. This exit is used only in connection with add-ons for the support of SCM products, in order to support Remote Syntax Check with Error Feedback. For these purposes, the Exit can also be integrated into home-grown Tool attachments as a REXX subroutine.</td>
</tr>
<tr>
<td>TAUTOXC3</td>
<td>REXX</td>
<td>Exit for the integration of C compile for Remote Syntax. This exit is used only in connection with add-ons for the support of SCM products, in order to support Remote Syntax Check with Error Feedback. For these purposes, the Exit can also be integrated into home-grown Tool attachments as a REXX subroutine.</td>
</tr>
<tr>
<td>TAUTOXIS</td>
<td>Skeleton</td>
<td>This skeleton must be adapted for the use of the Batch Interface in order to build up a valid ISPF environment in batch, under which a modelled ISPF Tool can be started.</td>
</tr>
<tr>
<td>TAUTOXJC</td>
<td>Skeleton</td>
<td>This skeleton must be adapted for the use of the Batch Interface, in order to support a default Job card.</td>
</tr>
</tbody>
</table>

#### 6.4.8 REXX Debug

In all the REXX procedures supplied, the trace can be dynamically switched on and off.

Go to the Eclipse Remote Systems View and select “Debug”, using the context menu of the MVS entry, to open a Debug popup window. Enter valid trace parameters described in “Table 6.6 REXX Trace options”.

![Debug popup window](image)
Chapter 6  Attaching Tools

You will find the REXX trace in the executing User Server if you open the SYSTSPRT DDNAME.
The following trace parameters are supported:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Display debug status</td>
</tr>
<tr>
<td>OFF</td>
<td>Switch off any REXX tracing</td>
</tr>
<tr>
<td>ON</td>
<td>Activate the trace in all REXX procedures</td>
</tr>
<tr>
<td>procedure list</td>
<td>A blank separated list of REXX procedure names to be traced</td>
</tr>
<tr>
<td>+procedure list</td>
<td>Add additional procedure names to be traced</td>
</tr>
<tr>
<td>-procedure list</td>
<td>Remove procedure names from the procedure list</td>
</tr>
<tr>
<td>!dialogdebug</td>
<td>Activates tracing in the ISPF dialog interface procedure TAUZCDIF</td>
</tr>
<tr>
<td>!dialogdebug OFF</td>
<td>Stops tracing in the ISPF dialog interface procedure TAUZCDIF</td>
</tr>
</tbody>
</table>

Table 6-9 REXX Trace options

### 6.5 Attaching ISPF Dialog Tools

ISPF tools containing ISPF dialog functionality have to be modelled as Tool Descriptors using the Tool Type ISPFDIALOG. They are always called by the central tool adapter REXX procedure TAUZCIFD. The attribute Invoke Form in the Tool Descriptor's ISA_ISPF_Tool relation can only have the value CMD or PGM. The value REXX is not supported.

When attaching ISPF dialog tools, the ISPF Dialog Application runs on the remote system in an ISPF batch environment. ISPF panel fields are exchanged with Eclipse dialog fields. The Eclipse dialogs have to be defined when modelling the application.

The advantage of this approach is that the REXX or program which implements the dialog application does not need to be changed. Customization is only necessary in ISPF panels called in the application.

The technique used is ISPF Panel REXX processing. The panels of the ISPF dialog are extended by calls to the central REXX procedure TAUZCDIF, which handles the communication between the ISPF Dialog Application and the Client dialogs.

The panel REXX TAUZCDIF can be called in three ISPF Panel Sections:

- INIT section
- REINIT section
- PROC section

Calling TAUZCDIF in the INIT or REINIT section is used to pass the values of the panel fields and ISPF variables relevant for the dialog between the ISPF application and the Client dialog in both directions.

Calling TAUZCDIF in the PROC section is only used for ISPF table processing.

The extension of the ISPF panel in the INIT or REINIT section has the following format:

```vget(tautodia) shared
if (&tautodia = 'Y')```
Chapter 6  Attaching Tools

The extension of the ISPF panel starts with a VGET statement of the Shared Pool variable TAUTODIA which is set to ‘Y’ by the ISPF dialog tool adapter TAUZCIFD.

Some special variables are defined and explained in the table below.

The panel REXX procedure TAUZCDIF will be called only if called via the Client/Server tool adapter TAUZCIFD. This means that the panel will still work in an ISPF online session.

It is important that all ISPF variables which are not fields in the BODY section of the panel are explicitly passed to the panel REXX, this means particularly the special variables explained below but also all other ISPF variables which are not declared as input or output fields in the panel’s BODY section but whose values are to be passed to the Client dialog.

The * in the parameter list of the TAUZCDIF call ensures that all ISPF variables defined as input or output field in the panel’s BODY section are passed to the Client dialog.

Note:

If the panel contains a MODEL section, only the ISPF variables of the MODEL section are passed by the * parameter to the panel REXX. In this case, all other ISPF variables of the BODY section have to be explicitly passed to the panel REXX as well.

The special variable TAUERR is fetched from the ISPF shared pool before calling the panel rexx, so that dialog error conditions detected in a preceding dialog step can be passed to the panel REXX. The panel REXX will send the error message to the client and the user will be informed of the error by an error message dialog.

The call to the panel REXX TAUZCDIF is followed by a check that the special variable TAUERR has some content. If it does, the content is concatenated to the actual content of the ISPF variable TAUTOZCP which contains error messages from the tool execution and is evaluated by the tool adapter TAUZCIFD.

<table>
<thead>
<tr>
<th>Special variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAUPANEL</td>
<td>Name of the panel</td>
</tr>
<tr>
<td>TAUPS</td>
<td>Panel section (I, R or P)</td>
</tr>
<tr>
<td>TAURESP</td>
<td>ISPF variable in which the dialog response (enter or PF3) is returned</td>
</tr>
<tr>
<td>TAUCURS</td>
<td>Name of the field in which the cursor is positioned</td>
</tr>
</tbody>
</table>
### 6.5 Attaching ISPF Dialog Tools

<table>
<thead>
<tr>
<th>TAUTTL</th>
<th>Title of the panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAUERR</td>
<td>ISPF variable containing error messages from the dialog tool</td>
</tr>
<tr>
<td>TAUPARMS</td>
<td>List of names of all variables to be passed to the SWT dialog. The special variable TAUPARMS contains a list of variable names whose values are passed to the Client dialog in the sequence in which they are written in the TAUPARMS string.</td>
</tr>
</tbody>
</table>

**Table 6-7 ISPF dialog special variables**

On the client side, a standard input dialog corresponding to the ISPF panel must be defined in the AWM (see 6.2.1). In addition to this “ISA PLUG-IN” definition an ISA ISPFPANEL relationship has to be defined as well (see 0).

The parameters defined in the modelled dialog tool must exactly correspond to the variables defined under TAUPARMS. If the Tool Descriptor has no ISA_ISPF_Panel relation, the variables in TAUPARMS are mapped to the Tool Descriptor’s dialog input parameters.

The extension of the ISPF panel in the PROC section of a non TBDISPL panel has the following format:

```plaintext
IF (&tautodia = 'Y')
  IF (&TAURESPP = 'Exit')
    .RESP = END
```

#### 6.5.1 ISPF Table Display support

In order to support an ISPF Table Display dialog some additional extensions are necessary.

Some additional special variables are provided for the Table Display support as defined in the following table:

<table>
<thead>
<tr>
<th>Special variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAUTAB</td>
<td>Name of the ISPF table to display</td>
</tr>
<tr>
<td>TAUTABV</td>
<td>List of the table variable names</td>
</tr>
<tr>
<td>TAUSCAN</td>
<td>‘N’ → no search argument is defined for the ISPF table ‘Y’ → a search argument is defined for the ISPF table</td>
</tr>
<tr>
<td>TAUCRP</td>
<td>Variable containing the value of the current row pointer</td>
</tr>
<tr>
<td>TAUSELS</td>
<td>Variable containing the count of selected lines</td>
</tr>
</tbody>
</table>

**Table 6-7 Table Display dialog special variables**

TAUCRP and TAUSELS are output variables set by the client.

A typical extension of the ISPF Table Display panel in the INIT or REINIT section looks like this:

```plaintext
VGET(TAUTODIA) SHARED
IF (&tautodia = 'Y')
  &resp = ''
  &tausels = 0
  &taupanel = '<panel name>'
  &tautab = '<table name>'
  &tauscan = 'N'
  &taups = 'I'
  &tautabv = '<tabvar1> ... <tabvarn>'
```
Chapter 6  Attaching Tools

```plaintext
&tauparms = <var1> ... <varn>'
&taueerr = ''
*REXX(*,TAUPANEL,TAURES,TAUPS,TAUPARMS,TAUSELS,TAUCRP,TAUTAB,
  TAUSCAN,TAUTABV,TAUERR,(TAUZCDIF))
IF (&taueerr ^= &z)
  VGET(TAUTOZCP) SHARED
  &tautozcp = '&tautozcp &taueerr'
  VPUT(TAUTOZCP) SHARED
  .autosel = YES
  &tau001 = &tabvar1
  . . .
  &tau00n = &tabvarn
```

The table variables must be saved after the call to the Panel REXX TAUZCDIF to refresh the content in the PROC section.

Another special extension of Table Display panels is that the panel REXX TAUZCDIF is also called in the PROC section of the panel. This call sets the current row pointer of the table to the value returned by the client. Additionally, in the PROC section, the table variables are refreshed to the content returned by the client, and the ISPF table control variable ZTDSELS is set to the number of selected rows returned by the client.

A typical extension of the ISPF Table Display panel in the PROC section looks like this:

```plaintext
IF (&autodia = 'Y')
  IF (&TAURES = 'Ok')
    &taupanel = '<panel name>'
    &tautab = '<table name>'
    &ztdsels = &tausels
    &tabvar1 = &tau001
    . . .
    &tabvarn = &tau00n
    &taups = 'P'
    *REXX(*,TAUPANEL,TAUPS,TAUCRP,TAUTAB,(TAUZCDIF))
    IF (&TAURES = 'Exit')
      RESP = END
```

In the AWM, a Tool Descriptor representing an ISPF Table Display dialog is characterized by its relation to a File Descriptor. The File Descriptor has to be defined as a structure Table with the Type InputStream (see 3.7).

The Properties referenced by the File Descriptor have to map to the table variables defined in the special variable TAUTABV.

By defining relations from the File Descriptor to Action Descriptors, you can configure the panel and line commands of the ISPF table. The Action Descriptor must point to a Tool Descriptor of tool type ISPF_PANEL_COMMAND. This Tool Descriptor must have an ISA_ISPF_Panel_Command relation (see: 3.13.4.4)

6.5.2 Special processing for ISPF Edit or ISPF Browse

If the ISPF Application contains an ISPF Edit or ISPF Browse call, this needs special treatment. This is the only situation where the ISPF application itself has to be changed. The application must call Editor Interface TAUZCEIF.
The customization in the REXX dialog application looks like this:

```bash
Call ISPF "8 VGET (TAUTODIA) SHARED"
If rc = 0 & tautodia = 'Y' Then
    Call TAUZCEIF "Browse "dsn
Else
    Address ISPEXEC "BROWSE "dsn
```

The Editor Interface has the following three parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>funccode</td>
<td>Function code:</td>
</tr>
<tr>
<td></td>
<td>• EDIT</td>
</tr>
<tr>
<td></td>
<td>• BROWSE</td>
</tr>
<tr>
<td>dsn</td>
<td>Dataset name; can be in the form dsn(member)</td>
</tr>
<tr>
<td>WAIT [O]</td>
<td>'WAIT' means the client process is waiting for the editor part to be closed; this parameter is not interpreted by the client process if the function code is BROWSE.</td>
</tr>
</tbody>
</table>

### 6.5.3 ISPF Dialog Exits

The following exits are defined for ISPF dialog support and can or should be adapted.

<table>
<thead>
<tr>
<th>Exit</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAUTOXMS</td>
<td>REXX</td>
<td>Exit to format the layout of ISPF message for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This exit is called when the ISPF application generates an ISPF message. The message ID and short and long message separated by hex low value are passed to the exit. The exit is expected to format and return the ISPF message in the form in which it should be displayed in the Eclipse dialog message area.</td>
</tr>
</tbody>
</table>
This chapter provides a complete description of an example AWM called PDS Explorer. This example will be used to explain the modelling procedure and programming of the Tool Adapter.

The example is shipped with the AWM (PDSCONF) and all REXX procedures (all names starting with TAUPD), and is also included as an entry in the Master Configuration. Using this example in a production environment is not recommended, both from the viewpoint of performance and due to the limited functionality.

All the model definitions to be carried out are described in the illustrations and tables in the next chapters.

Attributes that are not listed can remain loaded with their standard values.

### 7.1 Concept

Before the implementation and modelling can be started, you must design a concept for how a PDS Explorer should be connected to Eclipse using the AWM. Consider the following questions:

1. What structure should the Container Tree View have (Container Element Types)?
   - How deep should the Container hierarchy go?
   - What Properties are relevant for the application per Container Element Type?

2. What other Element types are there?
   - How fine should the difference between the Element Types be?
   - What Properties are relevant for the application per Element Type?
   - What Properties make an Element unique (ID Definition)?

3. How does a user want to search for Elements (Filter Type)?

4. How should Elements be shown in the Element Table View (Element List Structure)?
   - Which Properties are to be defined as columns? The performance of the available tools for the determination of the table contents must be considered here.

5. What actions will be needed (Action, Tool Descriptor)?
These questions have been answered and implemented as follows for the example AWM:

1. The first level of the Container Tree View should show a “Project” that will be interpreted as the first Qualifier for PDS files. The Userid of the logged-in user will be used for this as standard.

   The second and last level contains all the PDS files for this project (1st Qualifier).

   Only the Container Element Types “Project” and “Data Set” need therefore be modelled.

   The following Properties are supported per Data Set: volume, organization, record format, record length, block size, creation date, last used date, is migrated.

2. A rough categorization with the types “EDITABLE”, “NON-EDITABLE” and “UNKNOWN” has been selected for the definition of the Element Types.

   A PDS Member is uniquely identified through the Data Set name and its Member names on the assigned system. “Data Set Name” and “Member name” are the only two Key Properties. The following additional Properties are assigned: creation date, change date and time, change user, file suffix.

3. The user should be able to use generic symbols (* %) in the search for files. The only Search criteria are “Data Set Name” and “Member name”.

4. The setup of the Element List Structure should correspond to the table structure of the ISPF Data Set List Utilities (Data Set name, Member name, date created, date and time modified, change user).

5. Remote Edit and Browse are the only Element Actions supported.

### 7.2 Creating the AWM

A AWM must be recreated under an Eclipse Project (see 4.2.1) and be opened in the Configuration Editor. As an alternative, to complete the example, the example AWM delivered can be used.

#### 7.2.1 Application Options

As for all following object types and relationships, the object for the Application Options will be created via the “New Child” function (or “New Sibling”) in the Editor. As the availability of the function depends on the selection, the third level of the tree view in the Editor must first be selected (Model configuration). A new object is created with “New Child → Application Options”, whose attributes can be modelled in the Properties View (see 4.2.1).
Chapter 7  Example AWM: PDS Explorer

The following attribute values must be adapted:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Generic Symbol</td>
<td>*</td>
<td>The standard generic icon is used as a default for generic input fields.</td>
</tr>
<tr>
<td></td>
<td>%</td>
<td>All permitted generic symbols (including Default Generic Symbol)</td>
</tr>
<tr>
<td>Version</td>
<td>0.1</td>
<td>The version of the AWM.</td>
</tr>
</tbody>
</table>

In principle, the decimal value is arbitrary. A new value must be set for every release of a changed AWM so that the changes can be activated.

Table 7.1 Configuration of the Application Options
The table indicates only attributes of the Application Option that have to be modified for the PDS Explorer.

Note: The Attribute **Root Container Action** must be set later because it refers to an Action Descriptor, which is not yet modelled (see 7.2.6.1).

7.2.2 Properties

As many object types and relationships refer to Properties, it makes sense to define them early.

All Properties from Figure 7.1 must be defined for the example AWM:

![Figure 7.1 Modelling Properties](image)

All values that start with “PROP_” are IDs of the respective Properties.
7.2 Creating the AWM

<table>
<thead>
<tr>
<th>Property</th>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROP_MemberName</td>
<td>Value max length = 8</td>
<td>The Member name. This cannot contain more than 8 characters</td>
</tr>
<tr>
<td></td>
<td>Value min length = 1</td>
<td></td>
</tr>
<tr>
<td>PROP_Project</td>
<td></td>
<td>Property for the first Container level. By default, it will contain the Userid of the logged-in user as a value during the runtime.</td>
</tr>
<tr>
<td>PROP_IsMigrated</td>
<td>Type = Boolean</td>
<td>The value of this Property is Boolean.</td>
</tr>
<tr>
<td>PROP_ElementType</td>
<td>ElementType = true</td>
<td>Creates the connection between the Element and the Element Type during the runtime via the Match Name.</td>
</tr>
</tbody>
</table>

Table 7.2 Configuration of the most important properties

The table contains only Properties whose attribute values are not listed in Figure 7.1 Modelling Properties.

We also recommend you define a Label for each Property, so that the Properties will be displayed in a readable manner in the Property View. The label “Data set name”, for example, is suitable for PROP_DatasetName, with “Member name” for PROP_MemberName, etc.

7.2.3 File Descriptors

Tools can output files that are to be interpreted as value tables. The contents of these value tables must therefore be described in the AWM. Files of this kind will be returned to the following places in the example AWM (see Figure 7.2 Modelling File Descriptors):

- Expansion of Container Elements (FILE_ProjList, FILE_DatasetList)
- Return of Element Lists (FILE_Memberlist)
- Return of valid Data Set names (FILE_DatasetListSimple)
### File Descriptors

<table>
<thead>
<tr>
<th>File Descriptor</th>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE_ProjList</td>
<td>Type = MVS_Seq&lt;br&gt;Structure = Table</td>
<td>A Project List is returned by the Tool that is responsible for determining the Container Elements of Type “CONT_Project”.</td>
</tr>
<tr>
<td></td>
<td>Prop Delimiter = &quot;,&quot;</td>
<td></td>
</tr>
<tr>
<td>FILE_DatasetList</td>
<td>Type = MVS_Seq&lt;br&gt;Structure = Table</td>
<td>A Dataset List including Properties will be returned by the Tool that is responsible for determining the Container Elements of Type “CONT_DataSets”. The separator is “,” (comma), as spaces (the standard separators) could occur among the returned values.</td>
</tr>
<tr>
<td>FILE_DatasetListSimple</td>
<td>Type = MVS_Seq&lt;br&gt;Structure = Table</td>
<td>A Dataset List without Properties will be returned by the Tool that returns valid values for the “PROP_DatasetName” Property.</td>
</tr>
</tbody>
</table>
## 7.2 Creating the AWM

### Table 7-3 Modelling File Descriptors

The relationships of the File Descriptors are shown in Figure 7.2 Modelling File Descriptors.

<table>
<thead>
<tr>
<th>File Descriptor</th>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| FILE_Memberlist | Type = MVS_Seq  
                    Structure = Table  
                    Prop Delimiter = ",,“ | A Member List with Properties is returned by the Tool that generates Element Lists. The separator is ",,“ (comma), as spaces (the standard separators) could occur among the returned values. |

7.2.4 Icons

In order to be able to assign actions and elements later on in the AWM, these must first be defined. Icons for the Element Type “ELE_EDITABLE” and the “ACT_Edit” action are used in the PDS Explorer.

<table>
<thead>
<tr>
<th>ID</th>
<th>Predefined</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICON_Edit</td>
<td>Action_Edit</td>
</tr>
<tr>
<td>ICON_EDITABLE</td>
<td>Element_Edit</td>
</tr>
</tbody>
</table>

7.2.5 Actions & Tools

The following actions are used in the example AWM:

<table>
<thead>
<tr>
<th>Action Descriptor ID</th>
<th>Description</th>
</tr>
</thead>
</table>
| ACT_FilterMembers    | Creates an Element List on the basis of a filter.  
                        Is assigned to the Filter Type and will be executed via the context menu of a filter or a double click (depending on the "Default Action" attribute in "FilterType has Action"). |
| ACT_GetAllProperties | Returns missing Properties of an Element and its correct Element Type if ACT_FilterMembers cannot determine it.  
                        Is assigned to each Element Type and is executed via the context menu entry of an Element (“Refresh Element Properties”). |
| ACT_GetDatasets      | Returns all the Datasets of a project.  
                        Is assigned to the Container Element of the second level (CONT_Dataset) and will be executed when expanding the Project Container Element. |
### Action Descriptor ID | Description
--- | ---
ACT_GetProjects | Returns all “Projects” (first level of the Container hierarchy). By default this is the user’s mainframe system identification (TSO userid).
| Is referenced by the **Root Containers Action** Attribute of the Application **Options**.
ACT_RemoteBrowse | Opens an Element in the Editor (read-only access).
| Is assigned to all Element Types and can be called up via their context menus.
ACT_RemoteEdit | Opens an Element in the Editor and permits changes.
| Is assigned to the Element Type ELE_EDITABLE and can be called up via its context menu.
ACT_RetrieveDatasets | Returns all the Data Sets on the basis of a generic input String.
| Example: USERID.* delivers: USERID.COBOL USERID.PLI USERID.JCL ...  
| Is assigned to the “PROP_Dataset” Property and can be executed via input dialogs (including filter dialog) (“Retrieve” button).

Table 7-5 All Action Descriptors of the PDS Explorer

The following Tool Descriptors’ actions are necessary to support the actions defined above. It is important that the **Tool Type** attribute of every ISPF Tool is set to **ISPFCOMMAND**.
### Tool Descriptors

<table>
<thead>
<tr>
<th>Tool Descriptor</th>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOOL_GetProjects</td>
<td>ISA ISPFTool</td>
<td>Name of a REXX procedure that returns all the projects (as standard only the userid) in one file.</td>
</tr>
<tr>
<td></td>
<td>Program ID = TAUPDGPR</td>
<td></td>
</tr>
<tr>
<td>TOOL_GetDatasets</td>
<td>ISA ISPFTool</td>
<td>Name of a REXX procedure that returns all matching datasets to the projects (property values inclusive) in one file.</td>
</tr>
<tr>
<td></td>
<td>Program ID = TAUPDGPD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Out Parm Separator = &quot;.&quot;</td>
<td></td>
</tr>
<tr>
<td>TOOL_RetrieveDatasets</td>
<td>ISA ISPFTool</td>
<td>Name of a REXX procedure that returns all matching datasets to a Search String in a file.</td>
</tr>
<tr>
<td></td>
<td>Program ID = TAUPDGDL</td>
<td></td>
</tr>
<tr>
<td>TOOL_FilterMembers</td>
<td>Input Parameter #2 → PROP_MemberName</td>
<td>Default Value = *</td>
</tr>
</tbody>
</table>

**Symbols:**
- Input parameter (String)
- Input file (File)
- Output parameter (String)
- Output file (File)
<table>
<thead>
<tr>
<th>Tool Descriptor</th>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOOL_FilterMembers</td>
<td>ISA ISPFTool</td>
<td>Name of a REXX procedure that returns an Element List, including Element Properties, in a file on the basis of two parameters (filter criteria).</td>
</tr>
<tr>
<td></td>
<td>Program ID = TAUPDELI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OutParmSeparator = “,”</td>
<td></td>
</tr>
<tr>
<td>TOOL_RemoteEdit</td>
<td>Integrated Tool = RemoteEdit</td>
<td>Opens the file in which the File Descriptor of the Input Parameter is described in the Editor and saves changes.</td>
</tr>
<tr>
<td>TOOL_RemoteBrowse</td>
<td>Integrated Tool = RemoveBrowse</td>
<td>Opens the file in which the File Descriptor of the Input Parameter is described in the Editor (read-only).</td>
</tr>
<tr>
<td>TOOL_GetAllProperties</td>
<td>Resource Processing = UPDATE</td>
<td>Updates the selected Element with the values that are returned by the Tool.</td>
</tr>
<tr>
<td>ISA ISPFTool</td>
<td>Program ID = TAUPDEPR</td>
<td>The REXX procedure TAUPDEPR delivers the suffix, the name of a Property Set and the Element Type (Match Name) to an Element if the information is available.</td>
</tr>
<tr>
<td></td>
<td>OutParmSeparator = “,”</td>
<td>As spaces can occur in the outputs of the Tools, commas are used for the separation of outputs.</td>
</tr>
</tbody>
</table>

Table 7-6 Modelling Tool Descriptors and Relationships

The table only shows Tool Descriptors and Relationships whose attribute values cannot be found in Figure 7.3.

Figure 7.4 Modelling Action Descriptors

The Action Descriptor “ACT_RemoteEdit” references the “ICON_Edit” icon.
The Action Descriptors must be modelled as follows:

### 7.2.6 Container Element Types

Two Container Element Types have been used in the example AWM (see Figure 7.5 Modelling two Container Element Types):

<table>
<thead>
<tr>
<th>Container Element Type / Relationship</th>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTELE_Project</td>
<td>Matchname = PROJ</td>
<td>The Match Name used to map an element to its type based on the Property (PROP_ElementType).</td>
</tr>
<tr>
<td></td>
<td>Get Children = ACT_GetDatasets</td>
<td>The action which will be executed when expanding the Container Element in the Tree View at runtime.</td>
</tr>
<tr>
<td>CONTELE_Dataset</td>
<td>Matchname = DATASET</td>
<td>See CONTELE_Project</td>
</tr>
<tr>
<td></td>
<td>Get Children = ACT_FilterMembers</td>
<td>See CONTELE_Project</td>
</tr>
<tr>
<td>CONTELE_Project has Property → PROP_P</td>
<td>Treview Label = true</td>
<td>The value of the PROP_Project Properties should be the Label of the Element in the Tree View.</td>
</tr>
</tbody>
</table>
Chapter 7  Example AWM: PDS Explorer

| CONTELE_Dataset has Property PROP_DatasetName | Treeview Label = true | Sort Criterion = true | The value of PROP_DatasetName should be both a label and the sorting criterion of the Elements of the CONT_Dataset types. |

Table 7-7 Modelling Container Element Types and Relationships

The table shows only Container Element Types and Relationships whose attribute values cannot be found in Figure 7.5 Modelling two Container Element Types.

7.2.6.1  Setting the Root Containers Action

Now that the ACT_GetProjects Action Descriptor is modelled, it must be referenced by the Root Containers Action in the Application Options. This will allow the user to see the first level of Container Elements in the Tree View.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root Containers Action</td>
<td>ACT_GetProjects</td>
<td>The Action will return Container Elements of the Element Type “CONTELE_Project” and populate the Tree View if the Application is expanded.</td>
</tr>
</tbody>
</table>

Table 7-8 Modelling Root Containers Action

7.2.7   Element Types

The PDS Explorer differentiates among three Element Types:

- EDITABLE
  Element Type for editable files, such as COBOL programs or Text files.

- NON-EDITABLE
  Element Type for files that should not be edited (e.g. Load modules)

- UNKNOWN
  Element Type for all files whose type could not be determined. The procedure of the Filter Action always assigns this Element Type if the correct types cannot be deduced from the Dataset or Member name. The “GetAllProperties” action reads the data content, however, so the correct Element Type can later be assigned to the Element.

Configuration notes:
As the Properties of the individual Element Types are identical, they need be defined only once and are duplicated via Drag & Drop or Copy & Paste.

### Table 7-9 Modelling the major Element Types and relationships

The table shows only Container Element Types and Relationships whose attribute values cannot be found in Figure 7.6.

<table>
<thead>
<tr>
<th>Element Type / Relationship</th>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ELE_EDITABLE</td>
<td>Matchname = EDITABLE</td>
<td>The Match Name, by which the Element Type of an Element can be deduced on the basis of a Property (PROP_ElementType).</td>
</tr>
<tr>
<td></td>
<td>Icon = ICON_Editable</td>
<td></td>
</tr>
<tr>
<td>ELE_EDITABLE has Action → ACT_RemoteEdit</td>
<td>Default Action = true</td>
<td>This action should be carried out by a double-click on an Element of this type.</td>
</tr>
<tr>
<td>ELE_NON-EDITABLE, ELE_UNKNOWN has Action → ACT_RemoteBrowse</td>
<td>Matchname = NON-EDITABLE</td>
<td>See ELE_EDITABLE</td>
</tr>
<tr>
<td>ELE_NON-EDITABLE</td>
<td>Matchname = UNKNOWN</td>
<td>See ELE_EDITABLE</td>
</tr>
<tr>
<td>ELE_UNKNOWN</td>
<td>Special Property = Element_Name</td>
<td>This property is the element name. This information is mainly used for internal processes.</td>
</tr>
<tr>
<td>ELE_EDITABLE, ELE_NON-EDITABLE, ELE_UNKNOWN has Property → PROP_MemberName</td>
<td>Special Property = Element_Suffix</td>
<td>This property is the suffix of the elements local file name, which is important in order to be able to display and edit the Element in a suitable editor.</td>
</tr>
<tr>
<td>ELE_EDITABLE, ELE_NON-EDITABLE, ELE_UNKNOWN has Property → PROP_Suffix</td>
<td>Special Property = PropertyGroup</td>
<td>This property will contain the name of the correct Property Set of the Elements at runtime.</td>
</tr>
</tbody>
</table>

#### 7.2.7.1 ID Definition

MVS Members in one remote system are identified by the dataset and member name so the ID Definition references the Properties for the Dataset and Member name.

The ID Definitions must be referenced by the corresponding Element Types. The ID Definition Member “ID_MVSMember” must be referenced by all three Element Types “ELE_EDITABLE”, “ELE_NON-EDITABLE” and “ELE_UNKNOWN”.

![Modelling ID Definitions](image)
7.2.8 Filter Type

Only one filter type is defined in this AWM. It consists of two filter criteria (Dataset Name, Member name) together with the action that can return an Element List.

![Filter Types](image)

*Figure 7.8 Modelling Filter Types*

<table>
<thead>
<tr>
<th>Filter Type</th>
<th>Attribute</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIL_Memberfilter</td>
<td>Default Filter Type = true</td>
<td>This Filter Type should be used as standard when the Filter dialog is called up.</td>
</tr>
<tr>
<td>Has Filtercriterion</td>
<td>PROP_MemberName</td>
<td>Generic = true</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Entries containing wildcard characters are permitted in this field. The valid wildcard characters are defined in the Application Options.</td>
</tr>
</tbody>
</table>

*Table 7-10 Modelling a Filter Type*

The table contains only relationships whose attribute values cannot be taken from Figure 7.8 or can only be found with difficulty.

7.2.9 Element List Structure

The PDS Explorer contains only one Element List Structure, as the Element Types that should be viewed in the Table View all have the same Properties.

![Element List Structure](image)

*Figure 7.9 Modelling a Element List Structure*
### 7.3 REXX Procedures

The REXX procedures of the example AWM are a part of the delivered Mainframe components. The names of these procedures all start with the prefix TAUPD. These example procedures can be executed without any adaptation. If any special rules exist for the allocation of User files, however, the REXX procedure TAUPDXA1 must be modified.

#### 7.3.1 Determining projects (TAUPDGPR)

The TAUPDGPR procedure delivers the projects for the Container Tree View (see 7.2.6). If the procedure has not been adapted, it will return the TSO/ISPF Userid to the Client, which will then be used later in the process as the 1st Qualifier to determine file names.

As an alternative, this procedure can be modified to determine another valid list of 1st Qualifiers for the example.
Chapter 7  Example AWM: PDS Explorer

REXX Example: Extract from TAUPDGPR

```plaintext
. . .
Parse Arg .
. . .
/* Create the container list dataset */
Call REXX '0 TAUPDXA1 CONFIG CONLIST'
Parse Value Result With . condsn .
/* Alloc container list */
Call ALLOC_FILE condd condsn
/* Create the container entries */
prjlist = SET_PROJECTS()
k = Words(prjlist)
Do i = 1 to k
    projects.i = Word(prjlist,i)',PROJ'
End
projects.0 = k
/* Write projects stem to output dataset */
Call TSO "0 EXECIO * DISKW" condd "(FINIS STEM projects.)"
outparm = condsn
Call EXIT 0 wmsg

SET_PROJECTS:
Call ISPF '0 VGET (ZUSER) SHARED'
projs = zuser
/* as an alternative, you may return a blank separated list */
/* of dataset 1st qualifiers, e.g.: */
/* projs = 'PROD TEST DEV1 DEV2 MAINT1 MAINT2' */
Return projs
. . .
EXIT:
Parse Arg exitrc exitmsg
. . .
Exit exitrc outparm
```

Note the following in the programming for determining the Project Container:

- No input parameters are foreseen for the REXX procedure in the AWM.
- There must be a file to which the Container information will be written. The REXX subroutine TAUPDXA1 creates this file.
- The Container information will be written to this file line by line (with EXECIO in the example).
- The name of the file (value of the variable “outparm”) will be transmitted to the Client via the Exit statement.
- This file name is the only modelled output parameter.

7.3.2 Determining Datasets with Properties (TAUPDGPD)

The TAUPDGPR procedure delivers the PDS files, including their Properties for the Properties view, for the Container Tree View (see 7.2.6).
REXX Example: Extract from TAUPDGPD (simplified)

Parse Arg project.

/* Create the container list dataset */
Call REXX '0 TAUPDXA1 CONFIG CONLIST'
Parse Value Result With . condsn.
/* Alloc container list */
Call ALLOC_FILE condd condsn
/* Get datasets from dataset list project.* */
Call GET_DATASETS
Call TSO "0 EXECIO * DISKW" condd "(FINIS STEM dataset.)"
outparm = condsn
Call EXIT 0 wmsg

GET_DATASETS:

sdsn = project'.**'
Call ISPF "0 LMDINIT LISTID(DSNDD) LEVEL("sdsn")"
lmdinit.dsndd = 1
dsn = ''
j = 0
Do Forever
  Call ISPF "8 LMDLIST LISTID("dsndd") OPTION(LIST) DATASET(DSN) STATS(YES)"
  If Result > 0 Then Leave
  If zdldsntp = '' Then Iterate
  lmdlist.dsndd = 1
  j = j + 1
  parm = dsn, 'Strip(zdlmigr)', 'Strip(zdlvol)', 'Strip(zdlorg)
  parm = parm, 'Strip(zdllrecl)', 'Strip(zdllblksz)
  parm = parm, 'zdlicdate', 'zd1rdate', 'DATASET'
  dataset.j = parm
  End
dataset.0 = j
Return

EXIT:
Parse Arg exitrc exitms

Call ISPF '* LMDLIST LISTID('dsndd') OPTION(FREE)'
Call ISPF '* LMDFREE LISTID('dsndd')'
Call FREE_FILE condd

Exit exitrc outparm

Note the following in the programming for determining the Dataset Container:

- The project (1st qualifier) is the only input parameter for the REXX procedure foreseen in the AWM.
- There must be a file to which the Container information will be written. The REXX subroutine TAUPDXA1 creates this file.
- The Datasets for the 1st Qualifier, including the Properties, will be determined from the ISPF Library Management Services.
- The Container information matching the model definitions of the corresponding File Descriptor is written to the previously allocated file line by line as a comma-separated list.
- The name of the file (value of the variable “outparm”) will be transmitted to the Client via the Exit statement.
- This file name is the only modelled output parameter.
7.3.3 Determining Datasets without Properties (TAUPDGPL)

The TAUPDGPR procedure delivers a selection list of files according to the generic Search criterion for modelled Filter types (see Figure 7.8). The process for developing this procedure is similar to the previously described TAUPDGPD procedure (see 7.3.2).

7.3.4 Exit for the creation of files (TAUPDXA1)

The TAUPDXA1 procedure will be used as the Exit within the PDS Explorer example, in order to store the output files generated by the Tools. The Exit may need to be adapted with regard to the Allocation guidelines. The Exit creates the files for the Container and Element Lists, and will be called up as a subroutine in the previously described Procedures.

The following example shows an extract from this procedure, in which the allocation of the files is carried out.

REXX Example: Extract from TAUPDXA1

```rexx
/* Create Container list dataset */
CONLIST:
  condsn = zuser'.TAU_TMP.CONLIST'
  If "SYSDSN"("'condsn'" ') = "DATASET NOT FOUND" Then Do
    Call TSO "0 ALLOC FI("alcdd") DA("condsn")
     , "NEW CATALOG REUSE SPACE(1,1) CYLINDERS"
     , "LRECL(255) RECFM(V,B) DSORG(PS)"
    Call TSO "* FREE FI("alcdd")"
  End
  Return condsn

/* Create Element list dataset */
ELELIST:
  eledsn = zuser'.TAU_TMP.ELELIST'
  If "SYSDSN"("'eledsn'" ') = "DATASET NOT FOUND" Then Do
    Call TSO "0 ALLOC FI("alcdd") DA("eledsn")
     , "NEW CATALOG REUSE SPACE(1,1) CYLINDERS"
     , "LRECL(255) RECFM(V,B) DSORG(PS)"
    Call TSO "* FREE FI("alcdd")"
  End
  Return eledsn
```

. . . .
7.3.5 Determining Member Lists (TAUPDELI)

The TAUPDELI procedure generates the Member List of a PDS according to the filter criterion (see 7.2.8).

REXX Example: Extract from TAUPDELI (simplified)

```rexx
... Parse Arg indsn filter .
/* Create the element list dataset */
Call REXX '0 TAUPDXA1 CONFIG ELELIST'
Parse Value Result With . eledsn .
Call ALLOC_FILE eledd eledsn
/* Get the Elementtype */
type = GET_TYPE(indsn)
/* Get members from PDS */
Call GET_MEMBERS
Call TSO "0 EXECIO * DISKW" eledd "(FINIS STEM member.)"
outparm = eledsn
Call EXIT 0 wmsg
...

GET_TYPE:
Arg tdsn .
Select
When Pos('SOURCE', tdsn) > 0 Then ttype = 'EDITABLE'
... Otherwise ttype = 'UNKNOWN'
End
Return ttype

GET_MEMBERS:
Call ISPF "0 LMINIT DATAID(DSNNDD) DATASET('"indsn"') ENQ(SHR)"
Call ISPF "0 LMOPEN DATAID("dsnnd")"
j = 0
Do Forever
  mbr = ' '
  parm = "DATAID("dsnnd") OPTION(LIST) STATS(YES) PATTERN("filter")"
  Call ISPF "8 LMMLIST" parm "MEMBER(MBR)"
  If Result > 0 Then Leave
  j = j + 1
  If Symbol('ZLC4DATE') = 'VAR' Then
    parm = zlc4date','zlm4date zlmtime','Strip(zlcnorc)'),'Strip(zluser)
  Else parm = ',,,
  member.j = indsn','Strip(mbr)','parm','type
End
member.0 = j
Return
...

EXIT:
Parse Arg exitrc exitmsg
...
Call ISPF "* LMMLIST DATAID("dsnnd") OPTION(FREE)"
Call ISPF "* LMCLOSE DATAID("dsnnd")"
Call ISPF "* LFREE DATAID('dsnnd')"
Call FREE_FILE eledd
...
Exit exitrc outparm
```
Chapter 7  Example AWM: PDS Explorer

Note the following in the programming for determining the Member List:

- Two input parameters (PDS and Memberfilter) are foreseen for the REXX procedure in the AWM.
- There must be a file to which the Member information can be written. The REXX subroutine TAUPDXA1 is used to create this file.
- The Member, including the Properties, will be determined from the ISPF Library Management Services.
- The Member information matching the model definition of the corresponding File Descriptor is written to the previously allocated file line by line as a comma-separated list.
- The name of the file (value of the variable “outparm”) will be transmitted to the Client via the Exit statement.
- This file name is the only modelled output parameter.

7.3.6  Determining Element Properties (TAUPDEPR)

The TAUPDEPR procedure determines the modelled Properties for a PDS Member (see 7.2.5). Most of the Properties were already determined during the generation of the Member list. Some Properties are only determined during the individual processing – mainly for reasons of performance.

```
REXX Example: Extract from TAUPDEPR (simplified)

    Parse Arg indsn member .
    ...
    Call ALLOC_FILE indd indsn'('member')'
    Call TSO "2 EXECIO 200 DISKR" indd "(FINIS STEM record.)"
    Do i = 1 To record.0
       record.i = Translate(record.i)
       ...
       If Pos('IDENTIFICATION DIVISION',record.i) > 0 Then Do
          suffix = 'CBL'
          type = 'EDITABLE'
          propgroup = 'COBOL'
          End
       End
    outparm = suffix','propgroup','type
    Call EXIT 0 wmsg
    ...
    EXIT:
    Parse Arg exitrc exitmsg
    ...
    Call FREE_FILE indd
    ...
    Exit exitrc outparm
```

Note the following in the programming for determining the Member Properties:

- Two input parameters separated by spaces (File name, Member name) are foreseen for the REXX procedure in the AWM.
- The two output parameter File-Suffix and the Element Type will be determined by reading through the Members.
- These two output parameters, which are separated by commas according to the modelling definitions (value of the variable “outparm”), will be transmitted to the Client via the Exit statement.
7.4 Editing the Master Configuration

In order to be able to test the PDS Explorer example, the Example application must be entered into the Master Configuration File (see also 4.1).
Chapter 8
Commissioning

8.1 Starting for the first time

The following requirements must be met before Client model interpreter can be used:

- The Mainframe Components (REXX, Skeletons ...) must be installed according to the z/Server installation instructions.
- The Master Configuration File must exist on the remote system according to the reference in the REXX procedure TAUZCAPP and the modelled applications must be correctly entered.
- The AWMs must exist according to the reference in the Master Configuration File.
- The Workflow Manager must be installed on the Client PC under one of the supported environments (see: installation manual, Prerequisites).
- The connection to the remote system must be defined in Eclipse.

Once Eclipse has started, you must change to the Team Developer Perspective (see Figure 8.1).

Figure 8.1 Changing to the Team Developer Perspective
A new remote system can now be defined via the context menu in the Tree View (upper left) (see Figure 8.2). A dialog appears in which you can select the remote connection over which the search for AWMs should be carried out. Double-click the desired connection, and the Client will attempt to reach the remote system. If the connection has not yet been activated, a popup menu will request the identification of the user. Once the connection has been made, the Rexx procedure TAUZCAPP is called and creates a corresponding System entry in the tree view. If this is expanded by clicking on ▶, all the applications assigned in the Master Configuration will appear. The applications can be selected so that their properties can be viewed in the Properties View.

**Note:** Normally the system entry is already available and there is no need to create it manually.

An AWM can be activated via the **Load Application** function in the context menu of an application. This causes the downloading and reading of the AWM to which the application has been assigned. If an AWM has been downloaded once, it remains stored locally and can also be used when there is no connection to the system. Actions that require a connection to be made to the remote system will always check whether a new version of the AWM is available. If yes, the user receives a message, and the update is carried out automatically.

Once the application has been loaded, the user can work with it. The Container of the first level is determined by expanding the application, and filters can now be generated via the context menu of the Filter View (on the right).

**8.2 Further development and maintenance**

Modelled applications can be flexibly maintained. It is possible to modify and test both the AWM and tools of an application without changing the production status.
8.2.1 Maintaining the AWM

A suitable way to maintain an AWM is to define a test application in the Master Configuration File.

One way to test model modifications is to specify a local file path for the AWM in the Master Configuration File. This should point to a project in the workspace of the administrator (see 4.2.1), so that the project can conveniently maintain the file with the Model Editor. It is also possible to make use of the file suffix "\.model", as the file can be directly opened with the Model Editor.

Example: Conf attribute of the Master Configuration for a local test AWM

- `local: c:\config\repository.model`

Note the following for the maintenance and test of this application:

- The AWM is copied when loading the test application and uses the copy at runtime. The user should therefore be familiar with the rules that cause the loading of an application (see in particular the remarks regarding the Version number under 3.2 and 4.1). Note in particular that a revised AWM will only be loaded if the Version Number of the application in the Master Configuration is different from the version number in the modelled Application Options.

- The user can initiate the loading of his application for testing by the targeted modification of the version number in the Master Configuration and in the Application Options.

- If ISPF Tools that should be verified in the test environment are also changed during maintenance of the AWM, the ISPF test environment can be dynamically allocated for this application with the Exit TAUTOXA1 (see 6.4.7, and also EXITPARM under 3.2).

Certain changes to a AWM that has already been used could lead to the existing cache no longer being compatible with the AWM. This includes in particular the removal of Properties and the modification of IDs. The results are error situations at runtime (as a rule when loading the application).

The parts of the caches that must be deleted can be specified in the Master Configuration through the entry of a new version (see 4.1). If the effects on consistency as a result of the changes carried out are not absolutely clear to the user, a test must be carried out to determine which parts of the cache will have to be deleted in the case of an update. In order to do this, the most important components of the cache must have first been generated in the existing AWM. This requires the following actions to have been carried out at least once:
Further development and maintenance

- Loading an application
- Identifying containers (expansion of the application sufficient)
- Creation of a filter
- Creation of an Element List (e.g. by carrying out the filter action)
- Opening an Element in the Editor
- Last of all: Restarting Eclipse

If errors then occur when loading a new version, a number of components will have to be deleted in the cache (entries in the Master Configuration).

If a situation arises when testing that makes the automatic deletion of the cache by the entry no longer possible in the Master, the cache can be deleted manually at any time.

Figure 8.4 shows the complete cache of an application in the Navigator View. The individual files have the following meanings:

<table>
<thead>
<tr>
<th>File name</th>
<th>Meaning</th>
<th>Critical changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>eb_elementsinlists.xml</td>
<td>Contain all the information stored about Elements and Element Lists.</td>
<td>Changes to ID\nDefinitions, Element List Structures, Elements and all their relationships</td>
</tr>
<tr>
<td>lm_Elements.xml</td>
<td>The deletion of the files removes all Element Lists and the cached information about Elements. They should always be deleted together; otherwise inconsistencies could arise.</td>
<td>None\nChanges to Filter Types and their relationships</td>
</tr>
<tr>
<td>et_elements.xml</td>
<td></td>
<td>\n</td>
</tr>
<tr>
<td>ft_filters.xml</td>
<td>Contains all the filters defined by the user.</td>
<td>Changes to Filter Types and their relationships</td>
</tr>
<tr>
<td>ft_openfiles.xml</td>
<td>Stores references to all files that the user had open when leaving the application in the editor.</td>
<td>None</td>
</tr>
<tr>
<td>jt_jobs.xml</td>
<td>Contains all submitted and finished jobs.</td>
<td>None</td>
</tr>
</tbody>
</table>

Table 8-1 Cache Files
Even if the Navigator View supports the deletion of files in principle, the cache must be directly deleted from the file system after closing Eclipse. Otherwise, the cache will be rewritten when leaving Eclipse (as the cache is exclusively created on Close). The Navigator View can nevertheless be used as Help, however, by simply copying the path of the cache from the Properties View (see Figure 8.5).

If all changes to a new AWM have been tested locally, they can be released for all users. The following steps must be followed:

1. Increase the version number of the new AWM.

2. Adapt the Master Configuration File:
   - Compare the version number with the new AWM.
   - Where necessary, supplement the INFO entries for the deletion of the cache.

3. Update the central AWM

8.2.1.1 Changes to ID Definitions

The modification of an ID Definition or the reference of Element Types to an ID Definition is a special modification case if the application supports persistent local Files to Elements (e.g. Check-out). Elements that should be edited locally will be created under a unique path in the workspace that is derived from the ID definition. If the ID Definition has been changed, any local file which exists at the time of the new AWM replacement can no longer be reached via Integrated Tools. Changes to ID definitions should therefore be avoided wherever possible.

In order to avoid data losses, a backup folder is generated at every update of the AWM. This contains all the locally stored content of the corresponding application before the update (including the cache). If a user has Elements in local processing at the time of an update in which ID Definitions
will change, he can restore these manually from the Backup path by modifying the new folder structure.

Example (see Figure 8.6):

![Diagram showing ID definition update]

**Old ID Definition:**
1. Project
2. Project Definition
3. Stage
4. Type
5. Membername

**New ID Definition:**
1. Project
2. Stage
3. Type

*Figure 8.6 Local Application Data after an ID Definition Update*

Here, you can see the file path of the local checked-out Element TESTC1 before the update (red) and after it (blue). The new folder structure (blue) has been created manually.

**Note:** Cached files are there as well but not shown in this picture.