IBM SPSS Modeler 16 CLEF Developer's Guide

Note
Before using this information and the product it supports, read the information in "Notices" on page 317.

Product Information

This edition applies to version 16, release 0, modification 0 of IBM(r) SPSS(r) Modeler and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Preface

IBM® SPSS® Modeler is the IBM Corp. enterprise-strength data mining workbench. SPSS Modeler helps organizations to improve customer and citizen relationships through an in-depth understanding of data. Organizations use the insight gained from SPSS Modeler to retain profitable customers, identify cross-selling opportunities, attract new customers, detect fraud, reduce risk, and improve government service delivery.

SPSS Modeler's visual interface invites users to apply their specific business expertise, which leads to more powerful predictive models and shortens time-to-solution. SPSS Modeler offers many modeling techniques, such as prediction, classification, segmentation, and association detection algorithms. Once models are created, IBM SPSS Modeler Solution Publisher enables their delivery enterprise-wide to decision makers or to a database.

About IBM Business Analytics

IBM Business Analytics software delivers complete, consistent and accurate information that decision-makers trust to improve business performance. A comprehensive portfolio of business intelligence, predictive analytics, financial performance and strategy management, and analytic applications provides clear, immediate and actionable insights into current performance and the ability to predict future outcomes. Combined with rich industry solutions, proven practices and professional services, organizations of every size can drive the highest productivity, confidently automate decisions and deliver better results.

As part of this portfolio, IBM SPSS Predictive Analytics software helps organizations predict future events and proactively act upon that insight to drive better business outcomes. Commercial, government and academic customers worldwide rely on IBM SPSS technology as a competitive advantage in attracting, retaining and growing customers, while reducing fraud and mitigating risk. By incorporating IBM SPSS software into their daily operations, organizations become predictive enterprises – able to direct and automate decisions to meet business goals and achieve measurable competitive advantage. For further information or to reach a representative visit http://www.ibm.com/spss.

Technical support

Technical support is available to maintenance customers. Customers may contact Technical Support for assistance in using IBM Corp. products or for installation help for one of the supported hardware environments. To reach Technical Support, see the IBM Corp. web site at http://www.ibm.com/support. Be prepared to identify yourself, your organization, and your support agreement when requesting assistance.

Chapter 1. Overview

Introduction to CLEF

Component-Level Extension Framework (CLEF) is a mechanism that allows the addition of user-provided extensions to the standard functionality of IBM SPSS Modeler. An extension typically includes a shared library—for example, a data-processing routine or a modeling algorithm—that is added to IBM SPSS Modeler and made available either from a new entry on a menu or as a new node on the nodes palette.

To do this, IBM SPSS Modeler requires details about the custom program, such as its name, the command parameters that should be passed to it, how IBM SPSS Modeler should present options to the program and results to the user, and so on. To provide this information, you supply a file in XML format known as a **specification file**. IBM SPSS Modeler translates the information in this file into a new menu entry or node definition.

Some of the benefits of using CLEF are that it:

- Provides an easy-to-use, extremely flexible and robust environment that allows engineers, consultants, and end users to integrate new features into IBM SPSS Modeler.
- Ensures that extension modules can look and behave the same as native IBM SPSS Modeler modules.
- Allows extension nodes to execute as close to the speed and efficiency of native IBM SPSS Modeler nodes as possible.

System Architecture

Like IBM SPSS Modeler itself, CLEF uses a two-tier client/server architecture, where the tiers can reside either on the same machine or on two different machines.

Client-Side Components

The components of the client tier are shown here.

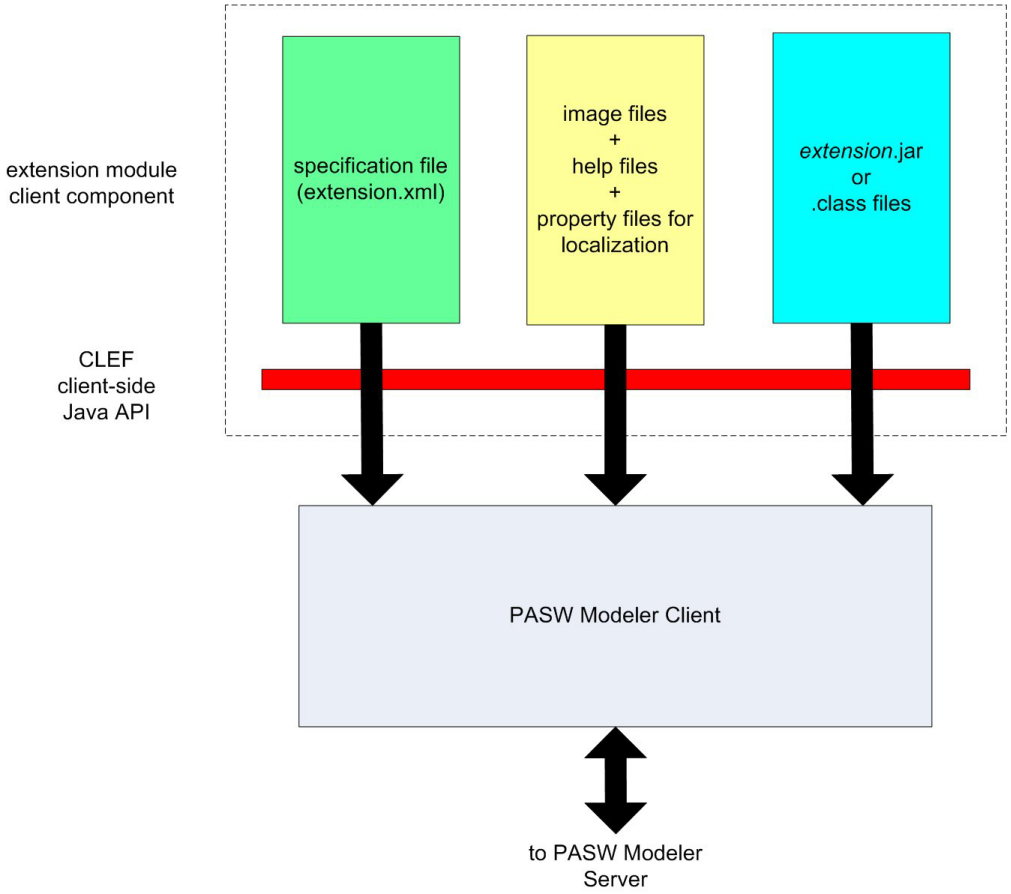

Figure 1. Client-side components

- **Specification file.** Lists properties, formats, data model changes, controls, and other characteristics that are defined by the extension.
- **Image files.** Contain the images used to identify a node in the extension.
- **Help files.** Used to display help information about the extension.
- **Property files.** Contain text strings comprising names, labels, and messages displayed on the screen by the extension.
- **Java .jar or .class files.** Contain any Java resources used by the extension.
- **Java application programming interface (API).** Can be used by extensions that require additional controls, user interface components, or interactivity not provided directly by the specification file.

Server-Side Components

The components of the server tier are shown here.

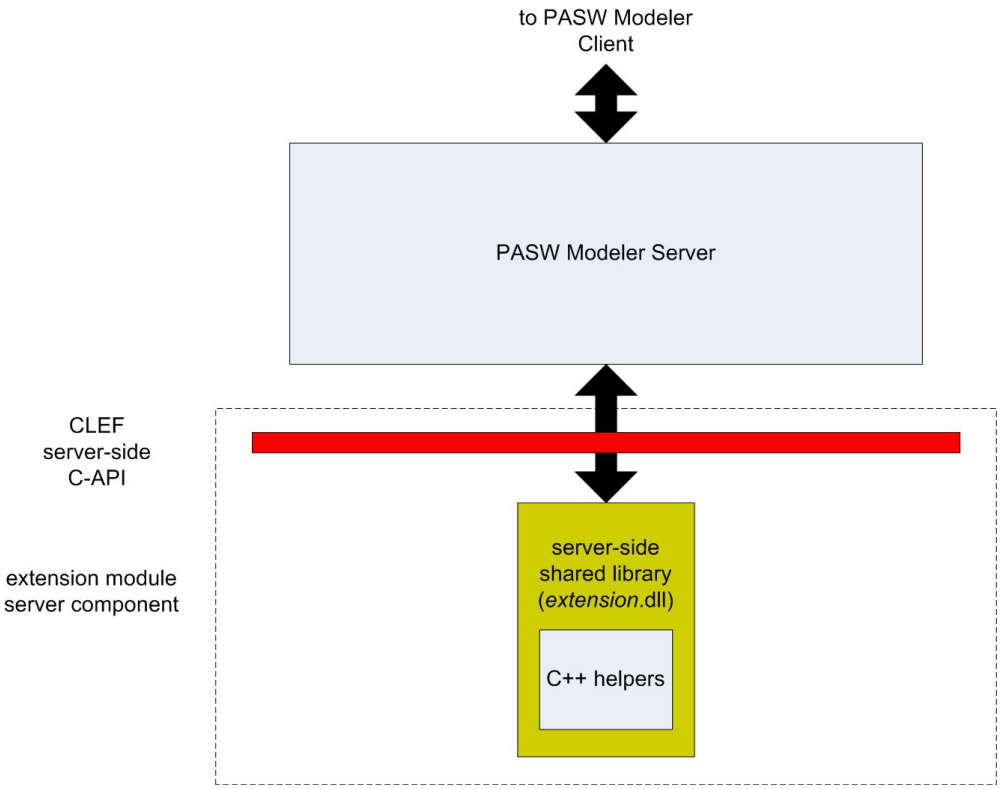

Figure 2. Server-side components

- **C-based API for shared libraries.** Covers aspects such as the setting and getting of execution settings, persistence of those settings, execution feedback, job control (for example, interrupting execution), SQL generation, and returned objects.
- **Server-side shared library.** A dynamic link library (DLL) that supports node execution. The C++ helpers are wrappers for some of the C-based APIs that are provided as source code and which can be easily compiled into a C++ CLEF module.

Features of CLEF

The following sections introduce a number of key features of CLEF:

- Specification file
- Nodes
- Data model
- Input and output files
- Application programming interfaces (APIs)

Specification File

The CLEF specification file is an XML file containing structured specifications describing the behavior of the new extension. A specification file describes:

- Shared resources needed by the extension (for example, localized text bundles and server-side shared libraries).
- Common definitions, such as file types or property types.
- New objects that the end user can utilize, such as nodes and output models.

When IBM SPSS Modeler starts up, specification files are loaded from the location where they reside, so that features defined in the files are immediately available.

For more information, see Chapter 4, “Specification File,” on page 29.

Nodes

When adding an extension to IBM SPSS Modeler that implements a new node, you first need to decide what type of node to create (for example, whether the node generates a model or simply transforms data). See the topic “Overview of Nodes” on page 9 for more information.

After creating the specification file and any necessary Java classes and shared libraries, you copy the files to specific locations from where IBM SPSS Modeler can read them. The next time you start IBM SPSS Modeler, the new node is added to the appropriate palette and is ready for use.

Data Model

The **data model** represents the structure of the data flowing through the IBM SPSS Modeler stream. Describing the data at that point in the stream, the model corresponds to the information displayed in the Type node. It lists the names of existing fields at a particular point in the stream and describes their type.

When adding any node to IBM SPSS Modeler, consider how the data model passed into the node affects the behavior of that node. For example, a Derive node takes an input data model, adds a new field to it, and produces an output data model that is passed to the next node in the IBM SPSS Modeler stream. In contrast, a Graph node takes an input data model and produces no output data model because the data are not passed to any subsequent nodes. IBM SPSS Modeler must know what will happen to the data model so that subsequent nodes can present the correct information about which fields are available. The data model information in the specification file gives IBM SPSS Modeler the information necessary to keep the data model consistent across the entire stream.

Depending on whether data flow into, out of, or through the node, the specification file must describe the data model for input, output, or both. A CLEF node can affect the data model either by adding new fields to whatever fields pass into the node, or by replacing the fields coming into the node with new fields generated by the program itself. The `OutputDataModel` element in the specification file describes the effects of the CLEF node on the data model. See the topic “Output Data Model” on page 55 for more information.

Input and Output Files

It is possible to specify one or more temporary files to be generated before a CLEF node is executed. These are known as **input files**, as they are input to node execution on the server. For example, a model builder node might have a model container from which the contents are transferred to the specified input file on execution of the node. See the topic “Input Files” on page 52 for more information.

Other temporary files are generated during execution of the node on the server; for example, the results of executing a model builder or document builder node. These are known as **output files**, and are transferred back to the client following node execution. See the topic “Output Files” on page 53 for more information.

Application Programming Interfaces (APIs)

Depending on what you want the extension to do, you may need to make use of an application programming interface (API). For a simple data transformation, you may be able to define the necessary processing entirely in the specification file. However, for more advanced requirements, you will need to interface with one or more of the available APIs:

- CLEF client-side API
- CLEF server-side API
- Predictive Server API (PSAPI)

The CLEF **client-side API** is a Java API that can be used by extensions requiring additional controls, user interface components, or interactivity not provided directly by the specification file.

The CLEF **server-side API** is a C-based API that covers aspects such as the setting and getting of execution settings, persistence of those settings, execution feedback, job control (for example, interrupting execution), SQL generation, and returned objects.

The **Predictive Server API** is a Java API that exposes IBM SPSS Modeler functionality for use by applications that require data mining and predictive analytic capabilities.

For more information, see Chapter 9, "Programming," on page 167.

File Structure

A CLEF extension consists of two sets of components:

- Client-side components
- Server-side components

The **client-side components** comprise the extension specification file, Java classes and .jar files, properties bundles containing localizable resources, and image and help files.

The **server-side components** are the shared libraries and DLLs required when an extension node is executed.

Client-Side Components

Client-side components are installed under the `\ext\lib` folder in the IBM SPSS Modeler installation directory. The client-side components are:

- Specification file
- Java classes and `.jar` files
- Property files
- Image files
- Help files

Extension Folder

Each extension is located in its own **extension folder** immediately under `\ext\lib`.

The suggested naming convention for the extension folder is:

`providerTag.id`

where *providerTag* is the provider identifier from the `ExtensionDetails` element of the specification file, and *id* is the extension identifier from the same element.

Thus for example, if the `ExtensionDetails` element begins as follows:

```
<ExtensionDetails providerTag="myco" id="sorter" ... />
```

the extension folder name `myco.sorter` would be used.

Specification File

The specification file itself must be named `extension.xml`, and must reside at the top level of the extension sub-folder. Thus in the example just given, the path to the specification file would be as follows under the IBM SPSS Modeler installation directory:

```
\ext\lib\myco.sorter\extension.xml
```

Java Classes and .jar Files

Extensions that use the client-side Java API include compiled Java code. This code can be left as a set of `.class` files, or it can be packaged as a `.jar` file.

Java `.class` files are located relative to the top-level extension folder. For example, a class that implements the `ActionHandler` interface might have the path:

```
com.my_example.my_extension.MyActionHandler
```

In this case, the `.class` file should be in the following location under the IBM SPSS Modeler installation directory:

```
\extension_folder\com\my_example\my_extension\MyActionHandler.class
```

A `.jar` file can be located anywhere under the extension folder. You specify the actual location of a `.jar` file by means of the `JarFile` element in the specification file. For example, if an extension uses a `.jar` file with the following path:

```
\extension_folder\lib\common-utilities.jar
```

the specification file should include the following entry in the `Resources` element:

```
<Resources>
    <JarFile id="util" path="lib\common-utilities.jar"/>
     ...
</Resources>
```

See the topic “Jar Files” on page 33 for more information.

Property Files

Localized resources (for example, screen text and error messages, and their foreign-language translations) can be stored in files with the extension `.properties`, which can be located anywhere under the extension folder. See the topic “Property Files” on page 159 for more information.

Image and Help Files

Files containing the graphic images for icon display, and those containing help systems, can be located anywhere under the extension folder. You might find it useful to separate image and help files into their own sub-folders.

You declare the location of an image file by means of the `imagePath` attribute of an `Icon` element in the specification file. See the topic “Icons” on page 102 for more information.

In a similar way, you declare the location of a help system using the `path` attribute of a `HelpInfo` element in the specification file. See the topic “Defining the Help System Location and Type” on page 155 for more information.

Example

The client-side file structure based on these components might be something like this:

```
\ext\lib\myco.sorter
\ext\lib\myco.sorter\extension.xml
\ext\lib\myco.sorter\sorter_en.properties
\ext\lib\myco.sorter\sorter_fr.properties
\ext\lib\myco.sorter\sorter_it.properties
```

```
\ext\lib\myco.sorter\com\my_example\my_extension\MyActionHandler.class
\ext\lib\myco.sorter\help\sorter.chm
\ext\lib\myco.sorter\images\lg_sorter.gif
\ext\lib\myco.sorter\images\sm_sorter.gif
\ext\lib\myco.sorter\lib\common-utilities.jar
```

Server-Side Components

Shared libraries required for execution must be located in a folder under the `\ext\bin` folder in the IBM SPSS Modeler installation directory, for example:

installation_directory`\ext\bin\myco.sorter\my_lib.dll`

Note that shared libraries must not be placed directly in the `\ext\bin` folder.

For shared libraries that IBM SPSS Modeler invokes directly during execution, you declare the location in a `SharedLibrary` element in the specification file. See the topic "Shared Libraries" on page 33 for more information.

The main shared library may require the use of other libraries. You should also place any dependent shared libraries in the same location as the main shared library to enable the dependent libraries to be found by the main library.

Example

An example of a server-side file structure might be:

```
\ext\bin\myco.sorter\my_lib.dll
\ext\bin\myco.sorter\my_lib2.dll
```

Chapter 2. Nodes

Overview of Nodes

When creating an extension that implements a new node, you need to become familiar with the characteristics of IBM SPSS Modeler nodes. Doing so will help you to define them correctly in the specification file.

IBM SPSS Modeler nodes are classified into source, process, output, and modeling nodes, depending on their function. In CLEF, nodes are classified in a slightly different way. The mapping between the two systems is shown in the following table.

Table 1. CLEF node types.

IBM SPSS Modeler classification	Palette	CLEF node type
Source nodes	Sources	Data reader
Process nodes	Record Ops	Data transformer
	Field Ops	
Output nodes	Graphs	Document builder
	Output (Report node)	
	Export	Data writer
Modeling nodes	Modeling	Model builder

When you create a new CLEF node, you define it as being one of the CLEF node types. The node type you choose depends on the main function of the node.

Table 2. Node types and functions.

CLEF node type	Description	Corresponding node palette	Icon shape
Data reader	Imports data into IBM SPSS Modeler from a different format.	Sources	 *Figure 3. Source node shape (circle)*
Data transformer	Takes data from IBM SPSS Modeler, modifies the data in some way, and returns the modified data to the IBM SPSS Modeler stream.	Record Ops, Field Ops	 *Figure 4. Ops node shape (hexagon)*
Model builder	Generates models from data in IBM SPSS Modeler.	Modeling	 *Figure 5. Model builder node shape*

Table 2. Node types and functions (continued).

CLEF node type	Description	Corresponding node palette	Icon shape
Document builder	Generates a graph or report from data in IBM SPSS Modeler.	Graphs	 *Figure 6. Graphs node shape (triangle)*
		Output (Report node)	 *Figure 7. Output node shape (rectangle)*
Model applier (also known as a "model nugget")	Defines a container for a generated model that is brought back to the IBM SPSS Modeler canvas.	–	 *Figure 8. Model applier node shape (gold diamond)*
Data writer	Exports data from IBM SPSS Modeler format to a format suitable for use with another application.	Export	 *Figure 9. Export node shape (rectangle)*

You define the node type, together with other attributes, in a `Node` element in the specification file—for example:

```
<Node name="sort_process" type="dataTransformer"
      palette="recordOp" ... >
     -- node elements --
</Node>
```

The `palette` attribute defines the palette in the IBM SPSS Modeler main window from which users will be able to access the node--in this case, the Record Ops palette. If you omit this attribute, the node appears on the Field Ops palette.

A number of example nodes are supplied with IBM SPSS Modeler. See the topic "About the Examples" on page 23 for more information.

Data Reader Nodes

A data reader node allows data from an external source to be read into an IBM SPSS Modeler stream. The nodes on the Sources palette of IBM SPSS Modeler are the equivalent of data reader nodes and are identified by having a circular icon shape.

In the specification for a data reader node, you include details of:

- The data source (such as a file or database)
- Any preprocessing of records (such as handling of leading and trailing spaces or the character to use as the record delimiter)
- Whether to filter out any record fields
- The data type (for example, range, set, flag) and storage type (string, integer, real) to associate with each field
- Whether the input data model is changed

The data reader node can include logic to read the source data records. Alternatively, this can be done further downstream by means of a Type node in IBM SPSS Modeler.

An example data reader node is supplied with IBM SPSS Modeler. See the topic "About the Examples" on page 23 for more information.

Data Transformer Nodes

A data transformer node takes data from an IBM SPSS Modeler stream, modifies the data in some way, and returns the modified data to the stream. The nodes on the Record Ops and Field Ops palettes of IBM SPSS Modeler are data transformer nodes, identified by having a hexagonal icon shape.

In the specification for a data transformer node, you include details of:

- Which records or fields are being transformed
- How the data is to be modified

An example data transformer node is supplied with IBM SPSS Modeler. See the topic "About the Examples" on page 23 for more information.

Model Builder Nodes

For an overview of model building in IBM SPSS Modeler, see "Introduction to Modeling" in the *IBM SPSS Modeler 16 Applications Guide*.

Model builder nodes generate objects that appear on the Models or Outputs tab of the manager pane in the IBM SPSS Modeler main window.

The nodes on the Modeling palette of IBM SPSS Modeler are examples of model builder nodes and are identified by having a pentagonal icon shape.

When executed, a model builder node generates a **model output object** (also known as a "model nugget") on the Models tab.

When a generated model is added to the canvas, it takes the form of a model applier node.

In the specification for a model builder node, you include:

- Model build details, such as the algorithm used to generate the model and which input and output fields are to be used for scoring data with the model
- Properties used by the model
- Containers used to hold output objects
- The user interface for the node dialog box
- Properties and files used when the node is executed
- How the input data model is affected by execution of the node
- The identifier of the model output object, and any other objects, produced by executing the node
- The identifier of the model applier node (see "Model Applier Nodes" on page 12)

Note: When defining a model builder node, you include the definition of the actual model output object and model applier node elsewhere in the same specification file.

An example model builder node is supplied with IBM SPSS Modeler. See the topic "About the Examples" on page 23 for more information.

Document Builder Nodes

Document builder nodes generate objects that appear on the Outputs tab of the manager pane of the IBM SPSS Modeler main window. The nodes on the Graphs palette are examples of document builder nodes and are identified by having a triangular icon shape.

When executed, a document builder node generates a **document output object** on the Outputs tab of the manager pane.

In contrast to a model output object, a document output object cannot be added back to the IBM SPSS Modeler canvas.

In the specification for a document builder node, you include:

- Document build details, such as the node dialog box tab that is to contain the document generation controls
- Properties used by the document
- Containers used to hold output objects
- The user interface for the node dialog box
- Properties and files used when the node is executed
- The identifier of the document output object, and any other objects, produced by executing the node

Note: When defining a document builder node, you include the definition of the actual document output object elsewhere in the same specification file.

Model Applier Nodes

A model applier node defines a container for a generated model for use when the model is added to the IBM SPSS Modeler canvas from the Models tab of the manager pane.

In the specification for a model applier node, you include details of:

- The container for the model (or containers, if model output can be produced in more than one format—for example, text and HTML)
- The user interface details for the dialog box displayed when the user browses the applier node on the Models tab or opens it on the canvas
- The output data model
- What processing to perform when the stream containing the node is executed
- Constructors to handle objects produced when the stream containing the node is executed

Data Writer Nodes

A data writer node exports data from IBM SPSS Modeler format to a format suitable for use with another application. The nodes on the Export palette of IBM SPSS Modeler are data writer nodes, identified by having a rectangular icon shape.

In the specification for a data writer node, you include:

- The details of the file or database to which the stream data is to be written
- Optionally, whether the entire stream is to be published so that it can be embedded in an external application

Menus, Toolbars and Palettes

Users can access an extension from an IBM SPSS Modeler menu, the toolbar, or a palette. An extension can either implement a node or perform a specified action.

An extension (node or action) that is accessible from an explicitly specified menu can also be made accessible from the toolbar, and vice versa.

A node that is accessible from a palette is automatically accessible from a corresponding item on the Insert menu.

Menus and Submenus

Users can access the standard IBM SPSS Modeler nodes from the Insert menu. Each of the items in the last group on this menu (apart from Models) has a submenu that provides access to a set of related nodes.

These items correspond directly to entries on the node palettes. Adding a node to a palette automatically adds it to the corresponding group on the Insert menu.

If your extension defines an action that is not accessible through a node, you can make the extension available by adding one or more of the following:

- A new item to a system menu or submenu
- A new menu to IBM SPSS Modeler
- A new item to the toolbar (see "Toolbars")

A new menu or menu item can optionally display the icon associated with the extension, as, for example, on some of the Insert menu items.

For more information, see "Menus" on page 103 and "Menu Items" on page 104.

Toolbars

If your extension defines an action that is not accessible through a node, you can make the extension available by adding it to the main IBM SPSS Modeler toolbar.

In this case, it is advisable to hide the label for the action.

You can also add an item to the toolbar of a node dialog box or output window. You can choose to show the item label or hide it.

See the topic "Toolbar Items" on page 105 for more information.

Palettes and Subpalettes

If your extension defines a new node, you can place the node in any position on one of the standard IBM SPSS Modeler palettes or subpalettes.

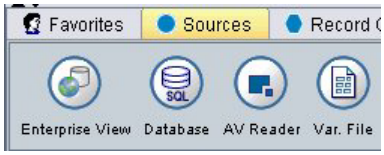

Figure 10. New node on a standard palette

You can add an entry to a standard subpalette and make the node accessible from there.

Figure 11. New node on custom addition to a standard subpalette

You can define a custom palette and place the new node there.

Figure 12. New node on a custom palette

A custom palette can have custom subpalettes.

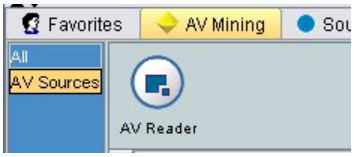

Figure 13. New node on a custom subpalette of a custom palette

For more information, see "Node" on page 45 and "User Interface (Palettes) Section" on page 40.

Designing Node Icons

For each new node that you create in CLEF, you can supply a central image for the icon that identifies the node on the screen.

Note: You do not have to supply an image--IBM SPSS Modeler provides a default, which is displayed if you do not specify one (this can be useful when starting to develop a node).

Figure 14. Default image for CLEF icons

Standard IBM SPSS Modeler icons are made up of three layers:

- Border
- Background
- Central image

For a new node, you need to supply only the central image (known as a **glyph**)--IBM SPSS Modeler handles the processing for the border and background. The glyph image needs to have a transparent

background so that it does not obscure the icon background layer. In this section, the representations of the glyph have a colored background to denote transparency.

This is how a typical IBM SPSS Modeler modeling icon is made up.

Table 3. Composition of node and generated model icons

	Node icon	Generated model icon
Border	 *Figure 15. Icon border*	None
Background	*Figure 16. Icon background*	 *Figure 17. Generated model background*
Glyph	 *Figure 18. Icon glyph*	 *Figure 19. Generated model glyph*
Image displayed as	 *Figure 20. Displayed icon*	 *Figure 21. Displayed generated model icon*

Borders

The function of the node is indicated by the shape of the icon border. See the topic "Overview of Nodes" on page 9 for more information.

If a node has caching enabled, the border shape has a miniature document symbol added to it. A white document icon on a node indicates that its cache is empty. When the cache is full, the document icon becomes solid green.

Table 4. Node borders and caching status

Caching status	Example
No caching	 *Figure 22. Node with no caching*

Table 4. Node borders and caching status (continued)

Caching status	Example
Caching enabled	 *Figure 23. Node with caching enabled*
Cache full	 *Figure 24. Node with cache full*

The different border symbols are supplied by the system, and IBM SPSS Modeler takes care of the necessary processing to display the right one at the right time.

Backgrounds

For node icons other than those for generated models and model applier nodes, the background changes color to indicate the state.

Table 5. Node backgrounds

State	Color	Example
Unselected	Gray	*Figure 25. Icon background (gray)*
Selected	Blue	 *Figure 26. Icon background (blue)*
Error	Red	 *Figure 27. Icon background (red)*
Action being conducted in a database	Purple	 *Figure 28. Icon background (purple)*

Again, the background images are supplied by the system, and IBM SPSS Modeler performs the necessary processing to display the correct background in each situation.

Graphics Requirements

For each new CLEF node, create the following versions of the glyph layer image:

- Large size (49 x 49 pixels) for nodes on the stream canvas
- Small size (38 x 38 pixels) for nodes in the palette manager at the bottom of the screen

If you want to display the icon on a menu or toolbar or in the title bar of a browser or an output window, you will also need to create:

- Miniature size (16 x 16 pixels)

If the node generates a model, you will also need to create:

- Small size (38 x 38 pixels) with the design moved to the bottom left corner, for overlaying on the generated model icon (the gold nugget)

Note: Images larger than these sizes will be trimmed when displayed in IBM SPSS Modeler.

See the topic "Icons" on page 102 for more information.

Creating Custom Images

The image you create for a node should convey the main function of the node. For an international audience, take care to use images that are not specific to one country and that are not likely to be misunderstood by users in other countries.

To create a custom image for use with CLEF:

1. Using a graphics package that supports transparency, set the drawing canvas to the appropriate size and draw the image version.
2. Save each version (large, small, etc.) as a separate *.gif* file with these characteristics:
   - Transparent background
   - Color depth of 16 colors (4-bit) or higher

   The way you make the image background transparent depends on the graphics package you are using. For example, you may be able to set the background color to transparent directly, or you may need to nominate a transparency color and then "paint" the image background with this color.

   For image files, we recommend following the file naming conventions that IBM SPSS Modeler uses internally, shown in the following table.

Table 6. Image file naming conventions

Image type	**Filename**
Large	lg_*node*.gif
Small	sm_*node*.gif
Miniature	*node*16.gif
Generated model	sm_gm_*node*.gif

3. Test the appearance of the image by referencing the image files in the specification file (see "Adding the Image Files to the Node Specification") and adding the new node to IBM SPSS Modeler (see "Testing a CLEF Extension" on page 191).

Adding the Image Files to the Node Specification

When you have created the image files, copy them to a folder on the computer from which you will be running IBM SPSS Modeler. In the specification file, you will need to declare an image path relative to an \ext\lib*provider.nodename* folder in the IBM SPSS Modeler installation directory, so you should deploy the files to a folder that is easy to reach from there. See the topic "Icons" on page 102 for more information.

In the specification file, you associate the large and small icon graphics files with a custom node by means of the `Icons` element in the `UserInterface` section of the `Node` specification—for example:

```
<Icons>
     <Icon type="standardNode" imagePath="images/lg_mynode.gif" />
     <Icon type="smallNode" imagePath="images/sm_mynode.gif" />
</Icons>
```

For model builder or document builder nodes, you also reference the miniature (16 x 16 pixel) version in the `UserInterface` section of the `ModelOutput` specification (for a model builder node) or the `DocumentOutput` specification (for a document builder node)—for example:

```
<Icons>
     <Icon type="standardWindow" imagePath="images/mynode16.gif" />
</Icons>
```

For model applier nodes, you also reference the generated model version in the `UserInterface` section of the `Node` specification—for example:

```
<Icons>
     <Icon type="standardNode" imagePath="images/lg_gm_mynode.gif" />
     <Icon type="smallNode" imagePath="images/sm_gm_mynode.gif" />
</Icons>
```

Designing Dialog Boxes

This section describes the characteristics of the standard IBM SPSS Modeler node dialog boxes to help you design consistent dialog boxes in CLEF.

About Node Dialog Boxes

A node dialog box provides an interface that enables the end user to modify execution settings. The appearance of the dialog box is very important; it is where the node behavior is altered and modified. The interface must contain all of the necessary information and be easy to use.

Node behavior is changed through the use of various dialog-box-based **controls**, which are user interface elements with which a user can interact. A dialog box may include a number of controls, such as radio buttons, check boxes, text boxes, and menus. CLEF provides a wide variety of controls that you can design into your dialog boxes. See the topic “Property Control Specifications” on page 115 for more information.

The type of parameter that is modified by a control determines which control appears on the dialog box, with some types providing alternate controls. You can group options on new tabs by means of `Tab` elements in the specification file. See the topic “Tab Area” on page 21 for more information.

Note: You can test the look and feel of the user interface for an extension even if you have not specified the processing that the extension is to perform. See the topic “Testing CLEF Extensions” on page 191 for more information.

Dialog Box Design Guidelines

When defining the controls for a dialog box, consider the following guidelines:

- Carefully consider the text to use where the control has a display label. The text should be reasonably concise while conveying the correct information. If designing for an international market, bear in mind that the length of translated text may differ significantly from that of the original.
- Use the correct control for a parameter. For example, a check box is not always the best choice for a parameter that takes only two values. The IBM SPSS Modeler C5.0 node dialog box uses radio buttons to enable users to select the output type, which has one of two values—**Decision tree** or **Rule set**.

This setting could be represented as a check box labeled **Decision tree**. When selected, the output type is a decision tree; when deselected, the output is a rule set. Although the outcome would be the same, using radio buttons makes it easier for the user to understand the options in this case.

- Controls for filenames are generally positioned at the top of the dialog box.
- Controls that form the focus of the node are positioned high in the dialog box. For example, graph nodes display fields from the data. Selecting those fields is the main function of the dialog box, so field parameters are placed at the top.
- Check boxes or radio buttons often allow the user to select an option that needs further information. For example, selecting **Use boosting** in the C5.0 dialog box requires that the analysis include a number indicating **Number of trials**.

  The extra information is always placed after the option selection, either at the right side or directly beneath it.

CLEF dialog boxes use IBM SPSS Modeler's commit editing in the same way as standard IBM SPSS Modeler dialog boxes: the values displayed in the dialog boxes are not copied to the node until the user clicks **OK**, **Apply**, or in the case of terminal nodes, **Execute**. Similarly, the information displayed by the dialog box is not updated (for example, when the input fields to the node have changed as a result of operating upstream of the current node) until the user cancels and redisplays the dialog box or clicks the **Refresh** button.

Dialog Box Components

Dialog boxes have the following components:

- Title bar
- Icon area
- Toolbar and menu area incorporating:
  - File, Generate, View, Preview, Refresh and other buttons (depending on the node)
  - Maximize/Normal size button
  - Help button
- Status area
- Panel area
- Tab area
- Button area

Each custom node needs a dialog box that is displayed when the user opens the node. Provided that your specification file includes a `Node` element containing a `UserInterface` section with a `Tabs` element, you will see all of the dialog box components listed above when you open the node. Depending on the node type, the minimum contents of the tab area and button area are shown in the following table.

Table 7. Minimum tab area and button area contents for different node types

Node type	Tabs	Buttons
Data reader	Annotations (with Refresh button in toolbar area)	OK, Cancel, Apply, Reset
Data transformer	Annotations	OK, Cancel, Apply, Reset
Data writer	Publish, Annotations	OK, Cancel, Execute, Apply, Reset
Model builder	Annotations	OK, Cancel, Execute, Apply, Reset
Document builder	Annotations	OK, Cancel, Execute, Apply, Reset
Model applier	Summary, Annotations	OK, Cancel, Apply, Reset

Node dialog boxes are initially positioned so that when the user opens the node, the node icon is superimposed on the node that it represents. The user can move the dialog box, but the new position is not remembered the next time the node is opened. If the user has moved the dialog box and it has subsequently been partially or fully hidden by another dialog box, double-clicking the original node on the canvas brings the first dialog box to the front again. The dialog box is modeless (that is, the same user input always causes the same action) and resizable.

All editable fields in the dialog box support the keyboard shortcuts shown in the following table.

Table 8. Keyboard shortcuts for editable fields in dialog boxes

Shortcut	Effect
Ctrl-C	Copy
Ctrl-V	Paste
Ctrl-X	Cut

Title Bar

The title bar of a node dialog box includes a miniature version of the IBM SPSS Modeler nugget icon, followed by the model name. The text is taken from the setting of the model name controls. Also supplied by default is the Close button (X) in the upper right corner.

Icon Area

The node icon is displayed in the icon area near the top left of the dialog box. This is the small size (38 x 38 pixel) version of the icon that is also used on the node palette at the bottom of the main window, not the larger version that appears on the canvas.

Note: The miniature nugget icon at the left end of the title bar is hard-coded into all node dialog boxes.

Toolbar and Menu Area

The topmost area of the dialog box is reserved as a toolbar and menu area.

Data reader and data transformer node dialogs have a Preview button in this area, displaying a sample of the input data.

Data reader node dialogs also have a Refresh button, which updates the information displayed by the node (for example, when the input fields to the node have changed).

Model applier nodes have File, Generate and View menu buttons, which enable users to perform various operations such as exporting the model or generating new nodes. Model applier nodes also have a Preview button, which in this case displays a sample of the input data together with the additional columns created when the node is applied.

The right side of this area contains two buttons in every node dialog box:

- Maximize/Normal size button
- Help button

Maximize/Normal Size Button: This button resizes the dialog box to full screen size. Subsequent use shrinks the dialog box back to the size it was before maximization.

Help Button: This button opens context-sensitive help for the node. For a tabbed dialog box or output window, help for that tab is displayed. The F1 key may also be used to access the help.

Status Area

The rest of the area at the top of the dialog box is reserved for displaying information, warnings, or error text. Source nodes show the full path and filename of the source data file here. Individual nodes may have other node-specific information to display in this area. Any text specified for this area should be limited to two lines.

Panel Area

This is the main area of the dialog box and contains all of the controls and display areas for the node. Each tab has a different panel area. Each panel can be one of the following types:

- Text browser
- Extension object
- Properties

You can also specify subpanels, which are separate dialog boxes that open in a new window and are called from action buttons on the panel.

See the topic "Panel Specifications" on page 109 for more information.

Tab Area

Node dialog boxes have the following tabs:

- One or more user-supplied node-specific tabs
- A Summary tab (model output objects and model applier nodes only)
- An Annotations tab

The node-specific tabs are defined in the `Tabs` section of the CLEF specification file. See the topic "Tabs" on page 107 for more information.

Dialog boxes for model output objects and model applier nodes have a Summary tab supplied by the system. This displays summary information about a generated model, including the fields, build settings, and model estimation process used. Results are presented in a tree view that can be expanded or collapsed by clicking specific items.

The Annotations tab is supplied by the system for all node dialog boxes and enables the user to specify information about the node. This includes the node name, tooltip text, and a longer comment field.

Name. The default node name is specified in the `Label` attribute of the Node element in the specification file (see "Node" on page 45). The user can rename the node by selecting **Custom**, entering a name in the Custom edit field, and clicking **Apply** or **OK**. The new name is preserved across sessions, although the default name can be restored by selecting **Auto**. A custom name specified on the Annotations tab overrides a custom name specified on any other tab of the dialog box.

Tooltip text. Text specified here is displayed as the tooltip for the node on the canvas. If no tooltip text is specified here, no tooltip will be displayed when the user hovers the cursor over the node.

Keywords. The user can specify keywords to be used in project reports and when searching or tracking objects stored in the IBM SPSS Collaboration and Deployment Services Repository.

Comments panel. This area allows the user to enter comment text.

Creation and save information. This is a non-editable text area that shows the creation information and name and date/time that a file was saved (date/time format depends on locale). If a save has not taken place, this field will say "This item has not been saved."

Button Area

At the bottom of every dialog box, **Apply**, **Reset**, **OK**, and **Cancel** buttons are displayed. If the node is a terminal node (an executable node that processes stream data), an **Execute** button is also present.

OK. Applies all settings and closes the dialog box. When the dialog box is first opened from the node, this button has focus (indicated by a blue rectangle around the button), and pressing the Enter key also performs the OK operation.

Cancel. Closes the dialog box and leaves the settings as they were before opening the dialog box or since the last Apply operation. Pressing the Esc key when the entire dialog box has focus also performs the Cancel operation.

Execute. Applies all settings, closes the dialog box, and executes the terminal node.

Apply. Saves the settings of the dialog box so that downstream operations can use them.

Reset. Resets the entire dialog box to the values it contained upon opening the dialog box or since the last Apply operation.

Designing Output Windows

This section describes the characteristics of the standard IBM SPSS Modeler output windows to help you design consistent output windows in CLEF.

Output windows enable you to display the output from:

- A model—for example, from scoring (applying a model to) a set of data
- A document—for example, a graph or report

See the topic "About User Interfaces" on page 99 for more information.

Output windows are similar to node dialog boxes but with the following differences:

- Title bar has a node-specific miniature icon instead of the generic gold nugget icon
- Main node icon is omitted
- In the toolbar and menu area, the Maximize/Normal button is omitted (may be replaced by a Close and Delete button in a document output window), although the window is still resizable using the mouse
- Status area is omitted
- Tabs are typically:
  - A Model tab (for model output windows) to display the predictor importance data, if this option is selected on the model node.
  - A single tab for the output.
  - A Summary tab (for model output windows) to display summary details about the model.
  - An Annotations tab (the annotation values are taken from the node that generated the output).
- Button area has OK, Cancel, Apply and Reset buttons

CLEF provides default model output and document output windows similar to the one illustrated above. These are normally displayed when you use a `ModelOutput` or `DocumentOutput` element in the specification file. See the topic "Object Identifier" on page 45 for more information.

Alternatively, you can specify a `ModelOutput` or `DocumentOutput` element in such a way as to completely replace the default output window with a custom window of your own design. See the topic "Custom Output Windows" on page 152 for more information.

Chapter 3. CLEF Examples

About the Examples

To help you become familiar with CLEF, an IBM SPSS Modeler installation includes a set of example nodes together with their complete source code. These are basic nodes with limited functionality, designed to help you understand how CLEF works and how to use it. You can try these nodes out now or at any convenient time.

The examples are:

- Data reader node (named Apache Log Reader)
- Data transformer node (named URL Parser)
- Document builder node (named Web Status Report)
- Model builder node (named Interaction)

The examples need to be activated before they can be used.

Activating the Examples

The examples are installed to the *Demos* directory in compressed format as part of an IBM SPSS Modeler installation. You need to activate the examples by extracting the files to their correct locations as follows.

On the computer where IBM SPSS Modeler is installed:

1. Exit from IBM SPSS Modeler if it is running.
2. Locate the file *clef_examples_ext_lib.zip* in the *Demos* folder of the IBM SPSS Modeler installation.
3. Extract the contents of *clef_examples_ext_lib.zip* to the *\ext\lib* folder in the IBM SPSS Modeler installation directory.

For an IBM SPSS Modeler-only installation, extract the contents of *clef_examples_ext_bin.zip* to the *\ext\bin* folder in the IBM SPSS Modeler installation directory.

On the computer where IBM SPSS Modeler and/or IBM SPSS Modeler Server are installed:

1. Extract the contents of *clef_examples_ext_bin.zip* to the *\ext\bin* folder in both the IBM SPSS Modeler and IBM SPSS Modeler Server installation directories.
2. Use the makefile supplied in *clef_examples_ext_bin.zip* to compile the source code for the examples. See the topic "Examining the Source Code" on page 26 for more information.

Finally, in all cases, start IBM SPSS Modeler and ensure that the nodes shown in the following table are visible on the nodes palette.

Table 9. Nodes that are visible on the nodes palette.

Palette Tab	Node
Sources	Apache Log Reader
Field Ops	URL Parser
Modeling	Interaction
Output	Web Status Report

Data Reader Node (Apache Log Reader)

The data reader node example is a source node that reads data from the access log file of an Apache HTTP Web server. The access log contains details of all requests that the Web server has processed. The log records are in the format known as Combined Log Format—for example:

```
IP_address - - [09/Jul/2007:07:57:38 +0000] "GET /lsearch.php?county_id=3 HTTP/1.1" 200 16348
"http://www.google.co.uk/search?q=thunderbirds+cliveden&hl=en&start=10&sa=N"
"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322)"
```

You can use the example node to convert the log records to a tabular format that is easier to read.

To use the Apache Log Reader node:

1. If you have not yet activated the CLEF examples, do so now. See the topic "Activating the Examples" on page 23 for more information.
2. Open IBM SPSS Modeler.
3. On the Sources tab of the nodes palette, select **Apache Log Reader** and add the node to the canvas.
4. Edit the node. In the Apache Log File field on the Option tab, type:

   demos_folder\combined_log_format.txt

   where *demos_folder* is the location of the *Demos* folder in the IBM SPSS Modeler installation directory (do not use the *$CLEO_DEMOS* format).
5. Click **OK**.
6. Add a Type node to the stream.
7. Edit the Type node. Click **Read Values** to read the data, and then click **OK**.
8. Attach a Table node to the Type node and execute the stream. The log file contents are displayed in tabular format.
9. Save the stream for use with the next two examples.

Data Transformer Node (URL Parser)

The data transformer node example performs further processing on the data returned by the previous example. You select an ID field (which should contain a unique value for every row) and an input field containing a URL. The node generates output consisting of these two fields, together with the URL data additionally parsed into separate generated fields. For example, if a URL record contains a query string such as:

```
http://www.dummydomain.co.uk/resource.php?res_id=89
```

the record is parsed as shown in the following table.

Table 10. URL record parsing example.

Generated field	Contents
*URLfield*_server	*http://www.dummydomain.co.uk*
*URLfield*_path	*/resource.php*
*URLfield*_field	*res_id*
*URLfield*_value	*89*

To use the URL Parser node:

1. If the stream from the previous example was closed, open it now. The stream contains the Apache Log Reader and Type nodes.
2. From the Field Ops tab of the nodes palette, attach a URL Parser node to the Type node.

3. Edit the URL Parser node. From the ID Field drop-down list, select **ReturnedContentSize**. From the URL Field drop-down list, select **ReferralURL**. Click **OK**.
4. Attach a Table node to the URL Parser node and execute the stream. The **ReturnedContentSize** and **ReferralURL** fields are displayed, with **ReferralURL** additionally parsed into four separate generated fields: **ReferralURL_server**, **ReferralURL_path**, **ReferralURL_field**, and **ReferralURL_value**.

Document Builder Node (Web Status Report)

The document builder node example reads the data passed down from the Web server log and generates a report in the form of an HTML file. The report consists of a table showing the percentages of log records returning various HTTP status codes (for example, 200, 302, 404, etc.).

To use the Web Status Report node:

1. If the stream from the first example was closed, open it now. This is the stream that contains the Apache Log Reader and Type nodes. If your stream has a URL Parser node from the second example, that node is ignored in this example.
2. From the Output tab of the nodes palette, attach a Web Status Report node to the Type node.
3. Edit the Web Status Report node. From the Status Code Field drop-down list, select **StatusCode**. Click **Execute**. An output window is displayed with the contents of the report.

Model Builder Node (Interaction)

The model builder node example operates independently from the other examples and enables you to build a simple model in the standard (non-interactive) way or to interact with the model before it is generated. The model attempts to predict customer churn for a telecommunications company.

To use the Interaction node:

1. If you have not yet activated the CLEF examples, do so now. See the topic "Activating the Examples" on page 23 for more information.
2. Create a new stream in IBM SPSS Modeler.
3. Add a Statistics File source node that imports the file *telco.sav* from the *Demos* directory.
4. On the Types tab, click **Read Values**, and then click **OK** in the message box to confirm.
5. Set the role of the **churn** field (the last one in the list) to **Target**, and then click **OK**.
6. From the Modeling tab of the nodes palette, attach an Interaction node to the source node.

To test standard (non-interactive) model building:

1. Run the stream to create a model nugget in the stream, and in the Models palette at the top right of the screen.
2. Attach a Table node to the model nugget.
3. Execute the Table node. Scroll to the right of the table output window to view the churn predictions. Field `$I-churn` contains the predicted values, while `$IP-churn` shows the confidence values (from 0.0 to 1.0) for the predictions.

To test interactive model building:

1. On the Model tab of the Interaction model builder dialog box, select **Start an interactive session**.
2. Click **Execute** to display the Interaction Test dialog box.
3. In the Interaction Test dialog box, click **Start Build Task** to display the model build progress.
4. When the model build operation is complete, select the row that has been added to the Build Tasks table in the dialog box.

5. In the toolbar area at the top of the dialog box, click the button with the yellow diamond-shaped icon. This generates the model output object (named **model_1**) in the Models palette at the top right of the screen.

The interactively generated model is identical to the first model except that it has a different name. Repeating the process from **Start Build Task** generates a further identical model named **model_2**, and so on.

Examining the Specification Files

A good way to understand how CLEF works is to examine the specification files for the supplied examples. You can find these files in:

install_dir\ext\lib*extension_folder*\extension.xml

where *install_dir* is the IBM SPSS Modeler installation directory, and *extension_folder* is one of the following:

- *spss.apachelogreader*
- *spss.interaction*
- *spss.urlparser*
- *spss.webstatusreport*

You may see other extension folders listed under *\ext\lib*--these relate to system-supplied IBM SPSS Modeler nodes that are produced using CLEF. Whether these nodes appear in your installation depends on the IBM SPSS Modeler modules that you have licensed. You might find it illuminating to browse through their specification files, too, but **do not change these files in any way**. If you do, these nodes may not function correctly, in which case you will have to reinstall the relevant IBM SPSS Modeler product. Changes to system-supplied files will not be supported by IBM Corp.

Examining the Source Code

For reference purposes, the complete source code for the example nodes is also supplied. All the example nodes use C++ server-side libraries, but only the Interaction node uses client-side Java classes in addition.

The source code files are automatically extracted when you activate the examples, and are installed as shown in the following table.

Table 11. Source code file installation

Location	Contents
...\ext\lib\spss.interaction\src	Java source code for the `.class` files in the `ui.jar` file in the parent folder
...\ext\bin\spss.apachelogreader\src *...\ext\bin\spss.interaction\src* *...\ext\bin\spss.urlparser\src* *...\ext\bin\spss.webstatusreport\src*	C++ source and project files for the DLLs in the parent folder

Removing the Examples

If you no longer want to see the example nodes in IBM SPSS Modeler, you can remove them as follows:

1. Exit from IBM SPSS Modeler.
2. Delete the example folders from both the *\ext\bin* and the *\ext\lib* directories in your IBM SPSS Modeler installation. Do not delete any of the standard IBM SPSS Modeler folders by mistake. If you do, you will have to reinstall the relevant IBM SPSS Modeler product. The folders to be deleted are:

- *spss.apachelogreader*

- *spss.urlparser*
- *spss.webstatusreport*
- *spss.interaction*

The changes take effect the next time you start IBM SPSS Modeler.

Chapter 4. Specification File

Overview of Specification Files

Every CLEF extension must include an XML file in which you define all the extension characteristics. This file is known as the **specification file**, and is always named `extension.xml`. A specification file consists of the following sections:

- **XML declaration**. Optional declaration of XML version and other information.
- **Extension element**. Main part of the file; contains all the subsequent sections.
- **Extension Details Section**. Specifies basic information about the extension.
- **Resources Section**. Specifies external resources that are required for the extension to function, such as resource bundles, JAR files, and shared libraries.
- **Common Objects Section**. (optional) Defines items that may be used or referenced by other objects in the extension, such as models, documents, and property types.
- **User Interface (Palettes) Section**. (optional) Defines a custom palette or sub-palette on which a node is to appear.
- **Object Definition Section**. Identifies the object or objects defined by the extension, such as nodes, model outputs, and document outputs.

Each section may contain static declarations (such as components in an element), or simple dynamic processes (such as computation of a node's output data model), or both. The overall format of a CLEF specification file is as follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Extension ... >
   <ExtensionDetails ... />
   <Resources
      Resources section
   </Resources>
   <CommonObjects>
      Common Objects section
   </CommonObjects>
   <UserInterface>
      User Interface (Palettes) section
   </UserInterface>
   Object Definition section
     object definition
     object definition
     object definition
    ...
</Extension>
```

Comment Lines

At any point in the specification file, you can include a comment line of the following format:

```
<!- comment text ->
```

Mandatory or Optional?

In the element definitions in subsequent sections (normally identified by the heading **Format**), element attributes and child elements are optional unless indicated as "(required)". For complete element syntax, see “CLEF XML Schema,” on page 195.

Example of a Specification File

Here is a complete example of a CLEF specification file, in this case for a simple data transformer node.

```
<?xml version="1.0" encoding="UTF-8"?>
<Extension version="1.0" debug="true">
     <ExtensionDetails id="urlparser" providerTag="spss" label="URL CLEF Module" version="1.0"
provider="IBM Corp." copyright="(c) 2005-2011 IBM Corp." description="A Url Transform CLEF Extension"/>
     <Resources>
          <SharedLibrary id="urlparser_library" path="spss.urlparser/urlparser" />
     </Resources>
     <Node id="urlparser_node" type="dataTransformer" palette="fieldOp" label="URL Parser">
          <Properties>
               <Property name="id_fieldname" valueType="integer" label="ID field" />
               <Property name="url_fieldname" valueType="string" label="URL field" />
          </Properties>
          <UserInterface>
               <Icons />
               <Tabs>
                    <Tab label="Types" labelKey="optionsTab.LABEL">
                         <PropertiesPanel>
                              <SingleFieldChooserControl property="id_fieldname"  storage="integer" />
                              <SingleFieldChooserControl property="url_fieldname"  storage="string" />
                         </PropertiesPanel>
                    </Tab>
               </Tabs>
               <Controls />
          </UserInterface>
          <Execution>
               <Module libraryId="urlparser_library" name="">
                    <StatusCodes>
                         <StatusCode code="0" status="error" message="Cannot initialise a peer" />
                         <StatusCode code="1" status="error" message="error reading input data" />
                         <StatusCode code="2" status="error" message="Internal Error" />
                         <StatusCode code="3" status="error" message="Input Field Does Not Exist" />
                    </StatusCodes>
               </Module>
          </Execution>
          <OutputDataModel mode="replace">
               <AddField name="${id_fieldname}" fieldRef="${id_fieldname}"/>
               <AddField name="${url_fieldname}" fieldRef="${url_fieldname}"/>
               <AddField name="${url_fieldname}_server" storage="string" />
               <AddField name="${url_fieldname}_path" storage="string" />
               <AddField name="${url_fieldname}_field" storage="string" />
               <AddField name="${url_fieldname}_value" storage="string" />
          </OutputDataModel>
     </Node>
</Extension>
```

The `ExtensionDetails` element supplies basic information about the extension that is used internally by IBM SPSS Modeler.

The `Resources` element specifies the location of a server-side library that will be referenced later in the file. The path specification indicates that the library is located in the IBM SPSS Modeler installation directory under `\ext\bin\spss.urlparser\urlparser.dll`.

This particular specification file does not include a `CommonObjects` element.

The `Node` element specifies all the information about the node itself:

- Under `Properties`, two properties are initially declared for later use on a tab of the node dialog.
- The `UserInterface` element defines the appearance and layout of the node dialog tab that is specific to this extension (other tabs are supplied by IBM SPSS Modeler).
- The `Execution` element defines items that are used when the node is executed. In this case, these items are the server-side library that was declared earlier in the file, and a set of messages for display if the execution returns a particular status code.

- The `OutputDataModel` element defines the data transformation that this node performs. It specifies that the input data model (the set of fields input to this node) is to be replaced by the set of fields defined here, which constitute the output data model (the set of fields passed on to all nodes downstream from here, unless the model is subsequently modified further). In this particular example, the node passes on the two original fields (`id_fieldname` and `url_fieldname`) unchanged, but adds four more fields whose names are derived from `url_fieldname`.

This particular specification file is taken from one of the example nodes that are supplied as part of the IBM SPSS Modeler installation. See the topic "Data Transformer Node (URL Parser)" on page 24 for more information.

XML Declaration

The XML declaration is optional, and specifies the version of XML being used, together with details of the character encoding format.

Example

```
<?xml version="1.0" encoding="UTF-8" ?>
```

Extension Element

The Extension element constitutes the main part of the file, and contains all the other sections. The format is:

```
<Extension version="version_number" debug="true_false">
     Extension Details section
     Resources section
     Common Objects section
     User Interface (Palettes) section
     Object Definition section
</Extension>
```

where:

`version` is the version number of the extension.

`debug` is optional; if set to `true`, it adds a **Debug** tab to any dialog or frame associated with a CLEF node or output, and provides access to the properties and containers defined for that object. Default value is `false`. See the topic "Using the Debug Tab" on page 192 for more information.

Extension Details Section

The Extension Details section provides basic information about the extension.

Format

```
<ExtensionDetails providerTag="extension_provider_tag"
     id="extension_unique_identifier"
     label="display_name" version="extension_version_number"
     provider="extension_provider" copyright="copyright_notice"
     description="extension_description"/>
```

where:

`providerTag` (required) is a name that uniquely identifies the provider of this extension. Note that the value should not include the string `spss`, which is reserved for internal use.

`id` (required) is a name that uniquely identifies this extension, and is used in system messages about it. By convention, the extension file is placed in a folder named \ext\lib*providerTag.id* in the IBM SPSS Modeler installation directory.

`label` (required) is the display label of the extension. This text is displayed in the Name field of the Palette Manager when the node is added. See the topic "Testing a CLEF Extension" on page 191 for more information.

`version` is the version number of this extension.

`provider` is a string that identifies the provider of this extension. This text is displayed in the Provider field of the Palette Manager when the node is added. Default is the string `(unknown)`.

`copyright` is the copyright notice for this extension. This text is displayed in the Copyright field of the Palette Manager when the node is added.

`description` is a brief description of the purpose of the extension. This text is displayed in the Description field of the Palette Manager when the node is added.

Example

```
<ExtensionDetails providerTag="myco" id="sorter" name="Sort Data" version="1.2"
     provider="My Company Inc." copyright="(c) 2005-2006 My Company Inc."
     description="An example extension that sorts data using built-in OS commands."/>
```

Resources Section

This section defines what external resources are required for this extension to function.

Format

```
<Resources>
     <Bundle .../>
      ...
     <JarFile .../>
      ...
     <SharedLibrary .../>
      ...
     <HelpInfo .../>
</Resources>
```

where:

`Bundle` identifies a set of client-side localized resources. See the topic "Bundles" on page 33 for more information.

`JarFile` identifies a client-side Java jar file. See the topic "Jar Files" on page 33 for more information.

`SharedLibrary` identifies a server-side library or DLL. See the topic "Shared Libraries" on page 33 for more information.

`HelpInfo` specifies the type of help information, if any, for the extension. See the topic "Implementing a Help System" on page 155 for more information.

Example

```
<Resources>
     <SharedLibrary id="discriminantnode" path="spss.xd/Discriminant"/>
     <Bundle id="translations.discrim" type="properties" path="messages"/>
     <JarFile id="java" path="discriminant.jar"/>
     <HelpInfo id="help" type="native"/>
</Resources>
```

Bundles

The `Bundle` element specifies a client-side resource bundle (such as a set of message texts for localization) which may be implemented either as a `.properties` file or a Java `.class` file. See the topic "Localization" on page 159 for more information.

Format

```
<Bundle id="identifier" path="path"/>
```

where:

`id` (required) is a unique identifier for this bundle.

`path` (required) specifies the location of the bundle file relative to the parent folder of this specification file. Where the bundle refers to a `.properties` file, the path must not include language extensions or the `.properties` suffix.

Example

```
<Bundle id="translations.discrim" path="messages"/>
```

This indicates that a resource bundle exists in a file named `messages.properties` in the same folder as the specification file.

Jar Files

The `JarFile` element specifies a client-side Java archive (`.jar`) file which provides Java classes and other client-side resources for this extension.

Format

```
<JarFile id="identifier" path="path"/>
```

where:

`id` (required) is a unique identifier for this `.jar` file.

`path` (required) specifies the location of the `.jar` file relative to the parent folder of this specification file.

Example

```
<JarFile id="java" path="coxreg_model_terms.jar"/>
```

This indicates that a `.jar` file for this extension is located in the same folder as the specification file.

Shared Libraries

The `SharedLibrary` element specifies a server-side shared library or DLL. This is usually only required for supporting node execution. Where a library implements multiple modules, a `Module` element in the Execution section of the node specification identifies a specific module within the library.

Format

```
<SharedLibrary id="identifier" path="path"/>
```

where:

`id` (required) is a unique identifier for this shared library.

`path` (required) specifies the location of the shared library relative to the `\ext\bin` folder in the server-side installation directory. Note that the path must not include the shared library file extension (e.g. `.dll`).

Example

The following shared library declaration:

```
<SharedLibrary id="binning" path="spss.binning/Binning" />
```

specifies that the shared library is to be loaded from:

install_dir`\ext\bin\spss.binning\Binning.dll`

where *install_dir* is the directory to which the server-side CLEF components are installed. As this library implements more than one module, the particular module required (`supervisedBinning`) is identified by means of a `Module` element in the specification of the build node, referencing the library identifier as follows:

```
<Execution>
     <Module libraryId="binning" name="supervisedBinning" .../>
 ...
</Execution>
```

Help Information

The optional `HelpInfo` element indicates which of the possible types of help are to be provided for this extension. See the topic "Implementing a Help System" on page 155 for more information.

Common Objects Section

The optional Common Objects section defines objects that can be shared between elements defined elsewhere in the specification file. Some types of object in this section (such as property enumerations) may also be defined locally where they are required, while others (such as models and documents) can only be defined here.

Format

```
<CommonObjects>
     <PropertyTypes .../>
     <PropertySets .../>
     <ContainerTypes .../>
     <Actions .../>
     <Catalogs .../>
</CommonObjects>
```

where:

`PropertyTypes` allows common property definitions to be shared between objects. See the topic "Property Types" on page 35 for more information.

`PropertySets` is typically used when model builder nodes, model output objects and model applier nodes include the same set of properties. See the topic "Property Sets" on page 36 for more information.

`ContainerTypes` defines types of containers, which are objects that can wrap complex data structures. See the topic "Container Types" on page 37 for more information.

`Actions` defines basic information about user interactions, for example by means of menus or toolbars. See the topic "Actions" on page 38 for more information.

`Catalogs` implement a control that allows the choice of one or more options from a list of values that is dynamically generated by the server. See the topic "Catalogs" on page 39 for more information.

Example

```
<CommonObjects>
     <ContainerTypes>
          <ModelType id="discriminant_model" format="utf8" />
          <DocumentType id="html_output" />
          <DocumentType id="zip_outputType" format="binary"/>
     </ContainerTypes>
</CommonObjects>
```

Property Types

The optional Property Types section allows common property definitions to be shared between objects. This is partly for ease of maintenance--for example, the definition of a property can appear in a single place rather than being duplicated in several places. Sharing of definitions is also used to ensure compatibility between properties in different objects whose values are copied when a new instance of an object is created.

Property types can be defined only in the Common Objects section.

Format

```
<PropertyTypes>
     <PropertyType id="identifier" isKeyed="true_false" isList="true_false" max="max_value"
       min="min_value" valueType="value_type">
          <Enumeration ... />
          <Structure ... />
          <DefaultValue ... />
     </PropertyType>
     <PropertyType ... />
      ...
</PropertyTypes>
```

The `PropertyType` attributes are as follows.

`id` (required) is a unique identifier for the property type.

`isKeyed`, if set to `true`, indicates that the property type is keyed. A keyed property associates a set of operations with a field by means of a user-defined control (see "Property Control" on page 131). If `isKeyed` is set to `true`, the `valueType` attribute must be set to `structure`. For more information on structured properties, see "Structured Properties" on page 58.

`isList` specifies whether the property is a list of values of the specified value type (`true`) or a single value (`false`).

`max` and `min` denote the maximum and minimum values for a range.

`valueType` can be one of the following:

- `string`
- `encryptedString`
- `fieldName`
- `integer`

- `double`
- `boolean`
- `date`
- `enum` (see "Enumerated Properties" on page 57)
- `structure` (see "Structured Properties" on page 58)
- `databaseConnection`

The `Enumeration` and `Structure` child elements are mutually exclusive. The `Enumeration`, `Structure` and `DefaultValue` child elements are used in specific cases--see "Enumerated Properties" on page 57, "Structured Properties" on page 58, and "Default Values" on page 60.

Property Sets

Property sets are typically used when model builder nodes, model output objects and model applier nodes include the same set of properties. For example, a model builder node may define a default set of properties that can be set in the builder but are not actually used until model application. In order to be transferred automatically, they also have to be included in the model output.

Format

```
<PropertySets>
     <PropertySet id="identifier">
          <Property ... />
          <Property ... />
           ...
     </PropertySet>
       ...
</PropertySets>
```

where `id` is a unique identifier for this property set.

For a description of the Property element, see "Properties" on page 48.

Example

This example demonstrates the definition of a set of two properties: the number of predictions to produce, and whether to include probabilities. In the Common Objects section, you would define:

```
<PropertySets>
     <PropertySet id="common_model_properties">
          <Property name="prediction_count" valueType="integer" min="1" max="10"/>
          <Property name="include_probabilities" valueType="boolean" defaultValue="false"/>
     </PropertySet>
       ...
</PropertySets>
```

Then, in each of the definitions for the model builder node, model output object and model applier node, you would have an `includePropertySets` attribute such as the following (this illustrates the definition for the model builder node only):

```
<Node id="my_builder" type="modelBuilder" ... >
     <Properties includePropertySets="[common_model_properties]">
      ...
     </Properties>
      ...
</Node>
```

Container Types

Containers are objects that act as placeholders for complex data structures such as models and documents. A container is defined as being of a particular type, and the container types are defined here. The following container types can be defined:

- model types
- document types

Container types may be transferred between client and server, cloned, and saved to a file or content repository. A model is cloned when a model applier node is generated from a model output object.

Each type of container has a predefined set of properties, although custom properties can be added. Container types can be defined only in the Common Objects section.

Format

The format of the Container Types section is:

```
<ContainerTypes>
     <ModelType ... />
      ...
     <DocumentType ... />
      ...
</ContainerTypes>
```

where:

`ModelType` specifies the format for a particular type of model. See the topic "Model Types" for more information.

`DocumentType` specifies the format for a particular type of document. See the topic "Document Types" on page 38 for more information.

Example

```
<ContainerTypes>
     <ModelType id="discriminant_model" format="utf8">
     <DocumentType id="html_output" />
     <DocumentType id="zip_outputType" format="binary"/>
</ContainerTypes>
```

Model Types

A model must provide information such as algorithm name, model type, and input and output data models. A model type definition specifies the format for a particular type of model.

The model type information may be specified statically here in the specification file, or dynamically when the model is constructed by the model builder node.

Format

`<ModelType id="`*`identifier`*`" format="`*`model_type_format`*`" />`

where:

- `id` (required) is a unique identifier for the model type.
- `format` (required) is the format of the model type, and can be either `utf8` (text) or `binary`. The model format must be specified as part of the static information.

Example

```
<ModelType id="my_model" format="utf8" />
```

Document Types

A **document** is an output object such as a graph or report. A document type definition specifies the format for a particular type of document.

Format

```
<DocumentType id="identifier" format="document_type_format" />
```

where:

- `id` (required) is a unique identifier for the document type.
- `format` (required) is the format of the document type, and can be either `utf8` (text) or `binary`.

Examples

```
<DocumentType id="html_output" format="utf8" />
<DocumentType id="zip_outputType" format="binary"/>
```

Actions

Actions define basic information about user interactions, for example by means of menus or toolbars. Each action defines how it should be represented in the user interface, such as a label, tooltip, or icon. A collection of actions is handled by a client-side Java class which is defined for each group of actions. Actions may also be defined within specific objects.

Format

```
<Actions>
     <Action id="identifier" label="display_label" labelKey="label_key"
     description="action_description" descriptionKey="description_key" imagePath="image_path"
     imagePathKey="image_path_key" mnemonic="mnemonic_char" mnemonicKey="mnemonic_key"
     shortcut="shortcut_string" shortcutKey="shortcut_key" />
      ...
</Actions>
```

where:

`id` (required) is a unique identifier for the action.

`label` (required) is the display name for the action as it is to appear on the user interface.

`labelKey` identifies the label for localization purposes.

`description` is a description of the action--for example, for a custom menu item or an icon action button on a toolbar, this would be the text of the tooltip for that menu item or button.

`descriptionKey` identifies the description for localization purposes.

`imagePath` is the location of a graphic file, such as for an icon image. The location is given relative to the directory in which the specification file is installed.

`imagePathKey` identifies the image path for localization purposes.

`mnemonic` is the alphabetic character used in conjunction with the Alt key to activate this control (for example, if you give the value S, the user can activate this control by means of Alt-S).

mnemonicKey identifies the mnemonic for localization purposes. If neither mnemonic nor mnemonicKey is used, no mnemonic is available for this control. See the topic "Access Keys and Keyboard Shortcuts" on page 108 for more information.

shortcut is a string indicating a keyboard shortcut (for example, CTRL+SHIFT+A) that can be used to initiate this action.

shortcutKey identifies the shortcut for localization purposes. If neither shortcut nor shortcutKey is used, no shortcut is available for this action. See the topic "Access Keys and Keyboard Shortcuts" on page 108 for more information.

Example

```
<Actions>
     <Action id="generateSelect" label="Select Node..." labelKey="generate.selectNode.LABEL"
      imagePath="images/generate.gif" description="Generates a select node"
      descriptionKey="generate.selectNode.TOOLTIP"/>
     <Action id="generateDerive" label="Derive Node..." labelKey="generate.deriveNode.LABEL"
      imagePath="images/generate.gif" description="Generates a derive node"
      descriptionKey="generate.deriveNode.TOOLTIP"/>
</Actions>
```

Catalogs

Catalogs enable you to associate a property with a control that allows the user to choose one or more options from a list of values that is dynamically generated by the server.

The values are displayed in the control as a popup list when the user clicks on the **<Select>** entry.

When the user selects a row in the list, the row value from a column specified in the Catalog element is placed in the control.

Format

```
<CommonObjects>
     <Catalogs>
          <Catalog id="identifier" valueColumn="integer">
               <Attribute label="display_name" />
                ...
          </Catalog>
           ...
     </Catalogs>
</CommonObjects>
```

where:

id (required) is a unique identifier for the catalog.

valueColumn (required) is the number of the column whose value will be placed in the control when the user selects a row. Column numbering begins at 1.

Use one Attribute element per column, in column order--see the example below.

When the user activates a control associated with a catalog, the catalog containing the list of values is retrieved from the server by means of a call to the getCatalogInformation function. This function returns an XML document containing the list of values. See the topic "Peer Functions" on page 170 for more information.

Example

This example illustrates some of the code used to define catalog controls. Three catalogs are defined and associated with three different controls on a dialog tab.

First, the catalogs are defined in the Common Objects section:

```
<CommonObjects>
     <Catalogs>
          <Catalog id="cat1" valueColumn="1">
               <Attribute label="col1" />
               <Attribute label="col2" />
          </Catalog>
          <Catalog id="cat2" valueColumn="2">
               <Attribute label="col1" />
               <Attribute label="col2" />
               <Attribute label="col3" />
          </Catalog>
          <Catalog id="cat3" valueColumn="1">
               <Attribute label="col1" />
          </Catalog>
     </Catalogs>
</CommonObjects>
```

Next, the properties to be associated with the controls are defined in the Properties section of the node definition:

```
<Node id="catalognode" type="dataReader" palette="import" label="Catalog">
     <Properties>
          <Property name="sometext" valueType="string" label="Some Text" />
          <Property name="selection1" valueType="string" label="Selection 1" />
          <Property name="selection2" valueType="string" isList="true" label="Selection 2" />
          <Property name="selection3" valueType="string" label="Selection 3" />
     </Properties>
```

In the User Interface section of the node definition, the controls are defined and associated with the catalog definitions through references to the properties:

```
<UserInterface>
     <Tabs>
          <Tab label="Catalog Controls" labelKey="Catalog.LABEL" >
               <PropertiesPanel>
                    <TextBoxControl property="sometext" />
                    <SingleItemChooserControl property="selection1" catalog="cat1" />
                    <MultiItemChooserControl property="selection2" catalog="cat2" />
                    <SingleItemChooserControl property="selection3" catalog="cat3" />
               </PropertiesPanel>
          </Tab>
```

User Interface (Palettes) Section

This is an optional section, and is included only if you want this extension to define a custom palette or sub-palette on which a node is to appear.

Where an extension defines a custom palette or sub-palette, subsequently-loaded extensions that define nodes to be included on the same palette or sub-palette can omit this User Interface (Palettes) section--all they need is for the `Node` element to have a `customPalette` attribute that references the palette. Extensions are loaded in alphabetical order of *providerTag.id* value, where these are the values of the `providerTag` and `id` attributes of the `ExtensionDetails` element for this extension (see "Extension Details Section" on page 31). Thus for example, the extension `myco.abc` is loaded before the extension `myco.def`.

Note: The User Interface (Palettes) section is different from the main User Interface section, which appears as part of an individual object definition, and which is described in Chapter 6, "Building User Interfaces," on page 99.

Format

The format of the User Interface (Palettes) section is:

```
<UserInterface>
     <Palettes>
          <Palette id="name" systemPalette="palette_name" customPalette="palette_name"
             relativePosition="position" relativeTo="palette" label="display_label"
             labelKey="label_key" description="description" descriptionKey="description_key"
             imagePath="image_path" />
          <Palette ... />
      ...
     </Palettes>
</UserInterface>
```

Table 12. Palette attributes.

Attribute	Description
`id`	(required) A unique identifier for the palette or sub-palette that you are defining.
`systemPalette`	Used only when adding a sub-palette to a system palette, and identifies the system palette in which this sub-palette is to appear: `import` - Sources `recordOp` - Record Ops `fieldOp` - Field Ops `graph` - Graphs `modeling` - Modeling (see below) `dbModeling` - Database Modeling `output` - Output `export` - Export
`customPalette`	Used only when adding a sub-palette to a custom palette, and identifies the custom palette in which this sub-palette is to appear. This is the value of the `id` attribute of the `Palette` element that defines the custom palette.

`relativePosition`	Used only when defining a custom palette, and specifies its position on the palette strip at the bottom of the screen.
	
	Possible values are:
	
	`first`
	`last`
	`before`
	`after`
	
	If the value is `before` or `after`, the `relativeTo` attribute is also required (see below).
	
	If `relativePosition` is omitted, the palette is placed last on the strip.

`relativeTo`	If the value of `relativePosition` is `before` or `after`, then `relativeTo` is used to specify the identifier of the palette that this custom palette precedes or follows. Palette identifiers are listed as the values of the palette attribute of the Node element (see "Node" on page 45).

Table 12. Palette attributes (continued).

Attribute	Description
`label`	(required) The display name for the palette or sub-palette as it appears on the user interface.
`labelKey`	Identifies the label for localization purposes.
`description`	The text of the tooltip displayed when the cursor hovers over the palette tab (not used for sub-palettes). This value also acts as the long accessible description of the control. See the topic "Accessibility" on page 165 for more information.
`descriptionKey`	Identifies the description for localization purposes.
`imagePath`	Identifies the location of the image used on the palette tab (not used for sub-palettes). The location is specified relative to the directory in which the specification file is installed. If you omit this attribute, no image is used.

Example - Adding a Node to a System Palette

Assume that your organization has developed a new algorithm for mining audio and video data, and you want to integrate the algorithm into IBM SPSS Modeler. You begin by defining a custom data reader node that will read input from audio and video files.

Initially you decide to add your new data reader node to the Sources system palette. All you need to do is to identify the Sources palette by means of the `palette` attribute of the `Node` element. See the topic "Node" on page 45 for more information.

Thus to add the node after the Database node on the Sources palette, you would use:

```
<Node id="AVreader" type="dataReader" palette="import" relativePosition="after"
   relativeTo="database" label="AV Reader">
```

Example - Adding a Custom Palette

Using a standard IBM SPSS Modeler palette is fine, but you want to give your new node more prominence. You decide to define a custom palette for it, which you will place after the Favorites palette but before Sources. First you need to add a User Interface (Palettes) section to define the custom palette as follows:

```
<UserInterface>
     <Palettes>
          <Palette id="AV_mining" label="AV Mining" relativePosition="before"
      relativeTo="import" description="Audio video mining"  />
     </Palettes>
</UserInterface>
```

The `relativeTo` attribute has to use the internal identifier of the Sources palette, which is `import`.

You then alter the `Node` definition as follows:

```
<Node id="AVreader" type="dataReader" customPalette="AV_mining" label="AV Reader">
```

This places the **AV Mining** palette between the Favorites and Sources system palettes.

Example - Adding a Custom Sub-Palette to a Custom Palette

Following on from the previous example, you now decide that you would prefer the data reader node to go on its own **AV Sources** sub-palette of the **AV Mining** palette. To achieve this, you first need to specify the sub-palette by adding a second `Palette` element to the User Interface (Palettes) section:

```
<UserInterface>
     <Palettes>
          <Palette id="AV_mining" label="AV Mining" description="Audio video mining" />
          <Palette id="AV_mining.sources" customPalette="AV_mining" label="AV Sources" />
     </Palettes>
<UserInterface>
```

You then alter the Node element to refer to the sub-palette identifier:

```
<Node id="AVreader" type="dataReader" customPalette="AV_mining.sources" label="AV Reader">
```

Now when the user clicks on the **AV Mining** tab, they see two sub-palettes, one labeled **All** and one labeled **AV Sources**. The AV Reader node appears on both of these.

If you add another new node to another new sub-palette of **AV Mining**, the new node appears on both the **All** and the new sub-palette, but not on the **AV Sources** sub-palette.

Example - Adding a Node to a System Sub-Palette

To process the audio and video source data, you now define a model builder node. You decide to add it to the standard Modeling palette, which has a number of standard sub-palettes. You choose to add it to the Classification sub-palette, placing it just before the Neural Net node, so you specify:

```
<Node id="AVmodeler" type="modelBuilder" palette="modeling.classification"
   relativePosition="before" relativeTo="neuralnet" label="AV Modeler">
```

Note that the node is also added in the same relative position on the All sub-palette of the Modeling palette.

Example - Adding a Custom Sub-Palette to a System Palette

Looking again at the number of model builder nodes on the Classification sub-palette, you realize that users may not notice your new node easily. One way to give your node more prominence is by adding your own sub-palette to the Modeling palette and placing the node there.

First, then, you must define your custom sub-palette by adding a User Interface (Palettes) section to the file:

```
<UserInterface>
     <Palettes>
          <Palette id="modeling.av_modeling" systemPalette="modeling" label="AV Modeling"
             labelKey="av_modeling.LABEL" description="Contains AV mining-related modeling
             nodes" descriptionKey="av_modeling.TOOLTIP"/>
     </Palettes>
</UserInterface>
```

Note that you must explicitly specify systemPalette to identify the system palette that you are extending.

Then, in the main User Interface section for the node, you specify that it is to appear on this sub-palette:

```
<Node id="my.avmodeler" type="modelBuilder" customPalette="modeling.av_modeling"
   label="AV Modeler">
```

Custom sub-palettes are always placed after the system sub-palettes.

Note: If you wanted to add more nodes to the AV Modeling sub-palette, their specification files would **not** need a User Interface (Palettes) section provided that the AV Modeler extension was loaded first.

Hiding or Deleting a Custom Palette or Sub-Palette

If you no longer want a custom palette or sub-palette to be displayed, you can either hide it or delete by means of the IBM SPSS Modeler Palette Manager.

Note that a hide operation persists across IBM SPSS Modeler sessions, but is reversible as it is controlled by a check box. A delete operation is irreversible in the same session, but on restarting IBM SPSS Modeler the item reappears unless you either remove it from the specification file, or remove the entire extension. See the topic "Deinstalling CLEF Extensions" on page 194 for more information.

To hide or delete a palette:

1. From the main IBM SPSS Modeler menu, choose:
   Tools > **Manage Palettes**
2. Select a palette in the Palette Name field, then:
   - to hide the palette, uncheck the corresponding Shown? check box
   - to delete the palette, click the Delete selection button
3. Click OK.

To hide or delete a sub-palette:

1. From the main IBM SPSS Modeler menu, choose:
   Tools > **Manage Palettes**
2. Select a palette in the Palette Name field.
3. Click the Sub Palettes button.
4. Select the sub-palette in the Sub Palette Name field, then:
   - to hide the sub-palette, uncheck the corresponding Shown? check box
   - to delete the sub-palette, click the Delete selection button
5. Click OK.

Object Definition Section

Elements are the most visible parts of an extension. The Object Definition section constitutes the remainder of the CLEF specification file, and is used to define the various objects in the extension. The following types of object can be defined:

- nodes
- model output objects
- document output objects
- interactive output objects

Nodes are the objects that appear in a stream. **Model output objects** are generated by model builder nodes, and appear under the Models tab of the manager pane in the main window. In a similar way, **document output objects** are generated by document builder nodes, and appear under the Outputs tab of the same pane. **Interactive output objects** are generated by interactive model builder nodes, and appear under the Outputs tab of the manager pane.

The Object Definition section consists of one or more of these object definitions.

The elements that can be defined for the different types of object are described in the following sections. Some of these elements are common to all object types, while others are specific to node or model output definitions. Object-specific elements are denoted as such in the text.

- object identifier
- model builder

- document builder
- properties
- containers
- user interface
- execution
- output data model
- constructors

Object Identifier

The object identifier indicates the type of object, and is one of the following:

```
<Node .../>

<ModelOutput .../>

<DocumentOutput .../>

<InteractiveModelBuilder .../>
```

The object identifier also provides information about how the object should be exposed through scripting. The `scriptName` attribute represents a unique name for the object. Scripts are able to use this attribute to specify a particular object (for example, a node in a stream, or an output in the Outputs tab).

Node

A node definition describes an object that can appear in a stream.

Format

```
<Node id="identifier" type="node_type" palette="palette" customPalette="custom_palette"
     relativePosition="position" relativeTo="node" label="display_label" labelKey="label_key"
     scriptName="script_name" helpLink="topic_id" description="description"
     descriptionKey="description_key">
     <ModelBuilder ... >
      ...
     </ModelBuilder>
     <DocumentBuilder ... >
      ...
     </DocumentBuilder>
     <ModelProvider ... />
     <Properties>
      ...
     </Properties>
     <Containers>
      ...
     </Containers>
     <UserInterface>
      ...
     </UserInterface>
     <Execution>
      ...
     </Execution>
     <OutputDataModel ...>
      ...
     </OutputDataModel>
```

```
    <Constructors>
     ...
    </Constructors>
</Node>
```

The elements allowed within the node definition are described in the sections beginning at "Properties" on page 48.

Table 13. Node attributes.

Attribute	Description
`id`	(required) An identifier for this node, in text string format.
`type`	(required) The type of node: `dataReader` - node that reads data (e.g. Sources palette nodes) `dataWriter` - node that writes data (e.g. Export palette nodes) `dataTransformer` - node that transforms data (e.g. Record/Field Ops nodes) `modelBuilder` - model builder node (e.g. Modeling palette nodes) `documentBuilder` - node that creates a graph or report `modelApplier` - node that contains a generated model The node type determines the shape of the node icon on the palette and canvas. See the topic "Overview of Nodes" on page 9 for more information. If the node type is `modelBuilder`, the node definition must include a `ModelBuilder` element - see "Model Builder" on page 48. If the node type is `documentBuilder`, the node definition must include a `DocumentBuilder` element - see "Document Builder" on page 48.
`palette`	The identifier of one of the standard IBM SPSS Modeler palettes or sub-palettes on which the node is to appear, namely: `import` - Sources `recordOp` - Record Ops `fieldOp` - Field Ops `graph` - Graphs `modeling` - Modeling (see below) `dbModeling` - Database Modeling `output` - Output `export` - Export The Modeling palette has a number of standard sub-palettes: `modeling.classification` - Classification `modeling.association` - Association `modeling.segmentation` - Segmentation `modeling.auto` - Automated If you omit the `palette` attribute, the node appears on the Field Ops palette. Note: The `palette` attribute is used only for model builder nodes.
`customPalette`	The identifier of a custom palette or sub-palette on which the node is to appear. This is the value of the `id` attribute of a `Palette` element, which is specified in the User Interface (Palettes) section of the file. See the topic "User Interface (Palettes) Section" on page 40 for more information.

Table 13. Node attributes (continued).

Attribute	Description
relativePosition	Specifies the position of the node within the palette. Possible values are: first last before after If the value is before or after, the relativeTo attribute is also required (see below). If relativePosition is omitted, the node is placed last in the palette.
relativeTo	If the value of relativePosition is before or after, then relativeTo is used to specify the node in the palette that this node precedes or follows. The value of relativeTo is the script name of the node. For a standard IBM SPSS Modeler node, the script name can be found in the "Properties Reference" section of the *IBM SPSS Modeler Scripting and Automation Guide*, but without the ...node suffix (for example, for the Database node you would use database, not databasenode). For a CLEF node, this is the value of the scriptName attribute for that node.
label	(required) The display name for the node as it is to appear on the palette, canvas and dialogs.
labelKey	Identifies the label for localization purposes.
scriptName	Used to uniquely identify the node when referenced in a script. See the topic "Using CLEF Nodes in Scripts" on page 71 for more information.
helpLink	An optional identifier of a help topic to be displayed when the user invokes the help system, if any. The format of the identifier depends on the type of help system (see Chapter 7, "Adding a Help System," on page 155) : HTML Help - URL of the help topic JavaHelp - topic ID
description	A text description of the node.
descriptionKey	Identifies the description for localization purposes.

The elements that can be contained within the node definition are described in the sections beginning at "Model Builder" on page 48.

Example

For an example of a node definition, see "Example of a Specification File" on page 30.

Model Output

A model output definition describes a generated model--an object that will appear under the Models tab in the manager pane following execution of a stream.

For full details on coding this part of the file, see "Model Output" on page 83.

Document Output

A document output definition describes an object such as a generated table or graph that will appear under the Outputs tab of the manager pane following execution of a stream.

For full details on coding this part of the file, see “Document Output” on page 94.

Interactive Model Builder

For full details on coding this part of the file, see “Building Interactive Models” on page 84.

Model Builder

This element is used in Node element definitions only.

For full details on coding this part of the file, see Chapter 5, “Building Models and Documents,” on page 75.

Document Builder

This element is used in Node element definitions only.

For full details on coding this part of the file, see Chapter 5, “Building Models and Documents,” on page 75.

Model Provider

This element is used in Node element definitions only.

When defining a model output object and a model applier node, you can use the `ModelProvider` element to specify the container that is to hold the model. You can also specify whether the model is stored in PMML format. PMML models can be viewed either through a custom viewer, or through the standard IBM SPSS Modeler model output viewer, which is provided by the `ModelViewerPanel` element. See the topic “Model Viewer Panel” on page 114 for more information.

Format

```
<ModelProvider container="container_name" isPMML="true_false" />
```

where:

`container` is the name of the container that holds the model.

`isPMML` indicates whether the model is stored in PMML format.

Example

```
<ModelProvider container="model" isPMML="true" />
```

For an example of the use of `ModelProvider` in the context of a model applier node, see the example under “Model Viewer Panel” on page 114.

Properties

A property definition consists of a set of name-and-value pairs. Individual property definitions, of which there may be many, are contained within a single Properties section.

Note: If a property is defined within the Properties section, there is no need to define it for an individual property control as the Properties section definitions take precedence. For this reason, we recommend defining properties within the Properties section.

The one exception to this rule concerns the `label` attribute. If the `label` attribute is defined for a property control, then *any* property definition found within that property control declaration (not just the `label` definition) takes precedence over its corresponding definition in the Properties section. Note that this

exception applies only to property controls, not to other types of control such as menus, menu items and toolbar items. These must define a label explicitly, either directly (menus) or indirectly through an `Action` element (menu items and toolbar items).

Format

```
<Properties>
     <Property name="name" scriptName="script_name" valueType="value_type" isList="true_false"
          defaultValue="default_value" label="display_label" labelKey="label_key"
          description="description" descriptionKey="description_key" />
          <Enumeration ... />
          <Structure ... />
          <DefaultValue ... />
     ...
</Properties>
```

The `Enumeration`, `Structure` and `DefaultValue` elements are used in specific cases. See the topic "Value Types" on page 57 for more information.

The `Property` element attributes are shown in the following table.

Table 14. Property attributes.

Attribute	Description
`name`	(required) A unique name for the property.
`scriptName`	The name by which the property is referred to in a script. See the topic "Using CLEF Nodes in Scripts" on page 71 for more information.

`valueType`	Specifies the type of value that this property can take, and is one of:
	
	`string`
	`encryptedString`
	`fieldName`
	`integer`
	`double`
	`boolean`
	`date`
	`enum`
	`structure`
	`databaseConnection`
	
	See the topic "Value Types" on page 57 for more information.

`isList`	Specifies whether the property is a list of values of the specified value type (`true`) or a single value (`false`).
`defaultValue`	The default value for this property. This may be expressed as either a simple value attribute or a compound element, and must be consistent with the specified valid values.
`label`	The display name for the property value as it is to appear on the user interface.
`labelKey`	Identifies the label for localization purposes.
`description`	A description of the property.
`descriptionKey`	Identifies the description for localization purposes.

Properties may optionally declare how valid values may be determined:

- For numeric values, this will be the minimum and/or maximum value.
- For strings, this is typically a field selection (e.g. all fields, all numeric fields, all discrete fields etc.) but may be a file selection.

- For enumerations, this will be the set of valid values.

Keyed properties must also declare how the valid keys are determined. Note that the key type of a keyed property must either be a string or an enumeration. See the topic "Property Types" on page 35 for more information.

The optional default value associated with the property is evaluated when the associated object is created. For example, node property defaults are evaluated each time a new instance of the node is created; execution properties are evaluated each time the node is executed. Evaluation occurs in the order in which the properties were declared.

Note that a property definition may refer to a property type declared in the Common Objects section.

Containers

A container is a placeholder for an output object whose generation is defined in the Constructors section.

Format

```
<Containers>
     <Container name="container_name" />
      ...
</Containers>
```

where:

`name` corresponds to the value of the target attribute of a `CreateModel` or `CreateDocument` element (see under "Using Constructors" on page 95), and indirectly associates the container with one of the container types declared in the Common Objects section.

Example

First, the container types are declared in the Common Objects section. There is one container type for models, in text format, and two container types for document output objects, one in the default (text) format for HTML output, and one in binary format for zipped output.

```
<CommonObjects>
     <ContainerTypes>
          <ModelType id="my_model" format="utf8" />
          <DocumentType id="html_output" />
          <DocumentType id="zip_outputType" format="binary" />
     </ContainerTypes>
</CommonObjects>
```

In the Execution section of the node definition, the output files are defined as container files with container types corresponding to the identifiers specified in the Common Objects section:

```
<Node id="mynode" ... >
 ...
     <Execution>
      ...
          <OutputFiles>
               <ContainerFile id="pmml" path="${tempfile}.pmml" containerType="my_model" />
               <ContainerFile id="htmloutput" path="${tempfile}.html" containerType="html_
               output" />
               <ContainerFile id="zipoutput" path="${tempfile}.zip" containerType="zip_
               outputType" />
          </OutputFiles>
```

Following this, the Constructors section defines the output objects to be generated when the node is executed. Here, the `CreateModel` and `CreateDocument` elements have a `sourceFile` attribute corresponding to a container file, as specified in the Output Files section earlier:

```
          <Constructors>
               <CreateModelOutput type="myoutput">
                    <CreateModel target="model" sourceFile="pmml" />
                    <CreateDocument target="advanced_output" sourceFile="htmloutput" />
                    <CreateDocument target="zip_output" sourceFile="zipoutput" />
               </CreateModelOutput>
          </Constructors>
     </Execution>
</Node>
```

Finally the Model Output section associates a container with a model output or document output object. In the `Container` element, the `name` attribute corresponds to the `target` attribute in the `CreateModel` and `CreateDocument` elements just specified:

```
<ModelOutput id="myoutput" label="My Model">
     <Containers>
          <Container name="model" />
          <Container name="advanced_output" />
          <Container name="zip_output" />
     </Containers>
      ...
</ModelOutput>
```

User Interface

The specification file supports a range of user interface components to allow objects to be displayed, and controls and properties to be modified. Facilities are provided to specify the layout and resizing behavior of the component, and whether the component should be enabled or made visible if other controls are modified.

The User Interface section specifies the visible appearance of an object. The specification can be used to customize a basic user interface component, such as a node properties dialog or an output window.

The User Interface section is a required part of the `Node` element specification.

For full details on coding this part of the file, see Chapter 6, “Building User Interfaces,” on page 99.

Execution

This element is used in Node element definitions only.

The Execution section defines properties and files that are used when a node is executed.

Format

```
<Execution>
     <Properties>
      ...
     </Properties>
     <InputFiles>
          <ContainerFile ... />
           ...
     </InputFiles>
     <OutputFiles>
          <ContainerFile ... />
           ...
     </OutputFiles>
```

```
     <Module ... >
          <StatusCodes ... />
     </Module>
     <Constructors ... />
</Execution>
```

The Execution section includes the definition of a set of properties that are re-created each time the node is executed and which are available only while the node is being executed.

The execution information can also define the set of input files to be generated prior to execution of the node, and any output files generated during execution.

Any number of input and output files may be specified. Each input file is associated with a container defined by the node. Each output file is typically used to construct containers for generated objects. The format of an input or output file is determined by the declaration of the container in the Common Objects section.

Example

For an example of an Execution section, see "Example of a Specification File" on page 30.

Properties (Runtime)

This section defines the set of runtime properties that are available only while the node is being executed.

Format

The format is similar to that of the `Properties` section in the main part of the element definition. See the topic "Properties" on page 48 for more information.

During execution of a model builder or document builder node, a **server temporary file** is created to store the model output or document output object. The server accesses this file and brings the object into the client where it is wrapped in a container. You need to specify this file here.

Example

This example illustrates how you specify the server temporary file.

```
<Properties>
     <Property name="tempfile" valueType="string">
          <DefaultValue>
               <ServerTempFile basename="datatmp"/>
          </DefaultValue>
     </Property>
</Properties>
```

Input Files

This section defines the set of input files to be generated prior to execution of the node. Input files in this context are files that are input to node execution on the server. For example, a model applier node has a model container that is transferred to the specified input file on execution of the node.

Format

```
<InputFiles>
     <ContainerFile id="identifier" path="path" container="container">
      ...
</InputFiles>
```

In the `ContainerFile` element for an input file, the attributes are as shown in the following table.

Table 15. Container file attributes - input files.

Attribute	Description
`id`	A unique identifier for the container file.
`path`	The location on the server where you want the input file to be generated (for example, the location of a server temporary file - see "Properties (Runtime)" on page 52).
`container`	The identifier of the container holding the object being sent to the server as input.

Example

```
<InputFiles>
     <ContainerFile id="pmml" path="${tempfile}.pmml" container="model"/>
</InputFiles>
```

Output Files

This section specifies the output files that are generated during execution of the node on the server. Output files (for example, the results of execution of a model builder or document builder node) are transferred back to the client following execution.

Format

```
<OutputFiles>
     <ContainerFile id="identifier" path="path" containerType="container">
      ...
</OutputFiles>
```

In the `ContainerFile` element, the attributes are as shown in the following table.

Table 16. Container file attributes - output files.

Attribute	Description
`id`	A unique identifier for the container file.
`path`	The location on the server of the object to be transferred to the client (for example, the location of a server temporary file - see "Properties (Runtime)" on page 52).
`containerType`	The identifier of the container type for the object (i.e. the ID of the model type or document type), to enable the object to be transferred in the correct format. See the topic "Container Types" on page 37 for more information.

Example

```
<OutputFiles>
     <ContainerFile id="pmml" path="${tempfile}.pmml" containerType="mynode_model" />
     <ContainerFile id="htmloutput" path="${tempfile}.html" containerType="html_output" />
     <ContainerFile id="zipoutput" path="${tempfile}.zip" containerType="zip_outputType" />
</OutputFiles>
```

Modules

This section specifies a server-side shared library to be used during node execution (for example, a DLL to be loaded into memory).

Format

```
<Module libraryId="shared_library_identifier" name="node_name">
     <StatusCodes ... />
</Module>
```

where:

libraryId is the identifier of a library declared in a Shared Library element in the Resources section. See the topic "Shared Libraries" on page 33 for more information.

name is used if the library is shared by more than one node, and identifies the particular node being executed. If the library is used only by one node, the name can be left blank.

Example

```
<Module libraryId="mynode1" name="mynode">
     <StatusCodes>
          <StatusCode code="0" status="error" message="An exception occurred" />
          <StatusCode code="1" status="error" message="Error reading input data" />
           ...
     </StatusCodes>
</Module>
```

Status Codes

Most programs perform some sort of error checking and display any necessary messages to the user, typically returning integers to indicate successful completion or other status. The server-side API can return a status code following execution of a stream containing the node. See the topic "Status Detail Document" on page 184 for more information.

The Status Codes section enables you to associate a message with a particular status code, for display to the user.

Format

```
<StatusCodes>
     <StatusCode code="codenum" status="status" message="message_text"
       messageKey="message_key" />
      ...
</StatusCodes>
```

The status code attributes are shown in the following table.

Table 17. Status code attributes.

Attribute	Description
code	The status code (an integer value) with which the message will be associated.
status	The status classification: success - node executed successfully warning - node executed with warnings error - node execution unsuccessful
message	The message to be displayed when this status code is returned.
messageKey	Identifies the message for localization purposes.

Example

In this example, the text of the error message is included in the StatusCode element:

```
<StatusCodes>
     <StatusCode code="0" status="error" message="Cannot initialise a peer" />
     <StatusCode code="1" status="error" message="Error reading input data" />
     <StatusCode code="2" status="error" message="Internal Error" />
     <StatusCode code="3" status="error" message="Input Field Does Not Exist" />
</StatusCodes>
```

On execution, if the server-side API returns status code 3, the following message is displayed to the user.

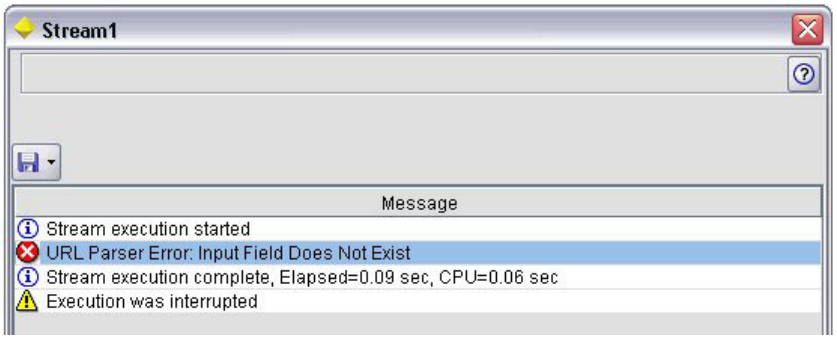

Figure 29. Error message display

In the next example, the error message texts are referenced by a `messageKey` attribute:

```
<StatusCodes>
     <StatusCode code="0" status="error" messageKey="initErrMsg.LABEL"/>
     <StatusCode code="1" status="error" messageKey="inputErrMsg.LABEL"/>
     <StatusCode code="2" status="error" messageKey="internalErrMsg.LABEL"/>
     <StatusCode code="3" status="error" messageKey="invalidMetadataErrMsg.LABEL"/>
      ...
</StatusCodes>
```

A property file (e.g. `messages.properties`), in the same folder as the specification file, contains the actual message texts as well as other display text:

```
 ...
initErrMsg.LABEL=Initialisation failed.
inputErrMsg.LABEL=Error when reading input data.
internalErrMsg.LABEL=Internal error.
invalidMetadataErrMsg.LABEL=Metadata (on input/output fields) not valid.
 ...
```

This method is useful when display text needs to be localized for overseas markets, for example, as all the text for localization can be found in one file. See the topic “Localization” on page 159 for more information.

Output Data Model

This element is used in Node element definitions only.

The Output Data Model section specifies how the data model is affected by certain properties.

The output data model can be determined in one of three ways:

- Using the field set definition features in the specification file. See the topic “Field Sets” on page 65 for more information.
- Using a client-side Java class that implements a data model provider interface that receives a set of properties and the input data model, and returns a data model instance.
- Using a server-side shared library component that receives a set of properties and the input data model and returns a metadata document.

The Output Data Model section defines how the properties in a node affect the fields flowing through the node. The output data model can:

- leave the input data model unchanged
- modify the input data model
- replace the input data model with a different data model

For example, a Sort node does not affect the properties themselves but reorders them, a Derive node modifies the data model by adding a new field, and an Aggregate node completely replaces the data model.

In the case where the input data model is modified, the definition can add new fields, or modify or remove existing fields. When replacing the data model only new fields can be added. The specification file supports these basic operations (including the ability to create new fields whose type is based on an input field) as well as the ability to iterate through the input field set or a key or list property representing a group of fields in the input field set.

Format

The general format of the Output Data Model section is as follows, but see the sections "Multi-Field Chooser Control" on page 128 and "Single-Field Chooser Control" on page 135 for specific formats for those cases.

```
<OutputDataModel mode="mode" libraryId="container_name">
     -- data model operations --
</OutputDataModel>
```

The output data model attributes are listed in the following table.

Table 18. Output data model attributes.

Attribute	Description
`mode`	Effect on the data model: `extend` - adds a new field to the existing model `fixed` - unchanged `modify` - change existing fields (e.g. remove or rename) `replace` - replace existing model
`libraryId`	The name of a server-side container from which to obtain the data model.

Data model operations are those that add new fields, or modify or remove existing fields. See the topic "Data Model Operations" on page 61 for more information.

Example

An `OutputDataModel` element is included in the example of a specification file. See the topic "Example of a Specification File" on page 30 for more information.

Constructors

Constructors define the objects that are produced as a result of either executing a node in a stream, or generating an object back to the stream.

For full details on coding this part of the file, see the sections beginning at "Using Constructors" on page 95.

Common Features

Some features can be used in more than one section of the specification file, namely:

- value types
- evaluated strings
- operations
- fields and field metadata

- field sets
- roles
- logical operators
- conditions

Value Types

Value type declarations specify the type of value that a column, property or property type specification can take.

Strings and Encrypted Strings

The `valueType="string"` format specifies that the value is a text string. A `valueType="encryptedString"` declaration is used for a property relating to a field whose contents need to be hidden when entered by the user, such as a password field.

Field Names

Where a value takes the form of a field name, use the `valueType="fieldName"` format.

Mathematical, Logical and Date Expressions

If the value is a mathematical (integer or double-precision), logical (true/false) or date expression, set `valueType` to `integer`, `double`, `boolean` or `date` accordingly.

Enumerated Properties

Enumerated properties are contained within an Enumeration section that immediately follows a `valueType="enum"` declaration. See the topic "Enumerated Properties" for more information.

Structure Declarations

A `valueType="structure"` declaration specifies a composite value containing other named attributes. Attributes are similar to properties but cannot be structured or keyed. See the topic "Structured Properties" on page 58 for more information.

- **Keyed indicator.** Specifies whether the property is a single value or is a hash table where each value in the table is of the specified value type.
- **Value set.** Specifies how the set of available values is determined.
- **Key set.** For keyed properties, this is used to specify how the set of available keys is determined. This information is also used to supply hints to the user interface about the most appropriate type of controller to use.

Database Connections

These are connection strings that enable users to log on to a database, for example `user1@testdb`. The logon details must have already been defined for the database. See the topic "Database Connection Chooser Control" on page 127 for more information.

Enumerated Properties

An enumerated property is one that can take a value from a predefined list of values.

Format

The format for enumerated properties uses an `Enumeration` section in which the list of values is defined, as follows:

```
<PropertyTypes>
     <PropertyType id="identifier" valueType="enum">
          <Enumeration>
               <Enum value="value" label="display_label" labelKey="label_key"
                    description="description" descriptionKey="description_key" />
           ...
          </Enumeration>
     </PropertyType>
<PropertyTypes>
```

where the PropertyType attributes are:

- id is a unique identifier for the property type.
- valueType indicates that the property type is enumerated.

and the Enum attributes are:

- value (required) is the property value that is to appear in the list of values.
- label (required) is the display name for the property value as it is to appear on the user interface.
- labelKey identifies the label for localization purposes.
- description is a description of the enumerated value.
- descriptionKey identifies the description for localization purposes.

Example

```
<PropertyTypes>
     <PropertyType id="shared_enum1" valueType="enum">
          <Enumeration>
               <Enum value="value1" label="Value 5.1" labelKey="enum5.value1.LABEL" />
               <Enum value="value2" label="Value 5.2" labelKey="enum5.value2.LABEL" />
               <Enum value="value3" label="Value 5.3" labelKey="enum5.value3.LABEL" />
          </Enumeration>
     </PropertyType>
<PropertyTypes>
```

Structured Properties

A structured property is one that is used in a grid-like structure such as a table control on a dialog.

Format

The format for structured properties uses a Structure section in which the structure is defined, and consisting of a number of Attribute elements, as follows:

```
<PropertyTypes>
     <PropertyType id="identifier" valueType="structure" isList="true_false">
          <Structure>
               <Attribute name="column_ID" valueType="value_type" isList="true_false"
                    label="column_label" labelKey="label_key" defaultValue="value"
                    description="description" descriptionKey="description_key" />
           ...
          </Structure>
     </PropertyType>
</PropertyTypes>
```

where the PropertyType element attributes are:

- id is a unique identifier for the property type.
- valueType indicates that the property type is structured.
- isList indicates whether the property is a list of values of the specified value type (true) or a single value (false).

and the Attribute element attributes are:

- name (required) is the identifier of the column.
- valueType specifies the type of value that the contents of this column can take, and is one of:

  string

  encryptedString

  integer

  double

  boolean

  date

  enum
- isList indicates whether the attribute is a list of values of the specified value type (true) or a single value (false). In this way, a keyed property can be associated with either a fixed set of known attributes (for example, Boolean attributes representing the different aggregation operations to be performed on a particular field) or a list of values (for example, associating a list of field names with some other field names).
- label (required) is the display name for the column as it is to appear on the user interface.
- labelKey identifies the label for localization purposes.
- defaultValue is a value that is to appear in the column when it is displayed.
- description is a description of the column.
- descriptionKey identifies the description for localization purposes.

Example - Table Control

For an example of how structured properties are used in a table control, see "Table Control" on page 138.

Examples - Keyed Property Types

The first of these examples illustrates the use of a keyed property type where each associated value is a structure representing which aggregation operation to apply to a field from a fixed set of operations:

```
<PropertyType id="aggregateOps" isKeyed="true" valueType="structure">
  <Structure>
    <Attribute name="MIN" valueType="boolean" label="Min" />
    <Attribute name="MAX" valueType="boolean" label="Max" defaultValue="true"/>
    <Attribute name="SUM" valueType="boolean" label="Sum" defaultValue="false"/>
    <Attribute name="MEAN" valueType="boolean" label="Mean" defaultValue="false"/>
    <Attribute name="SDEV" valueType="boolean" label="SDev" defaultValue="false"/>
  </Structure>
</PropertyType>
```

Thus a property declared to use the aggregateOps property type might be:

```
<Property name="aggregationSettings" scriptName="aggregation_settings" type="aggregateOps"/>
```

Here the property consists of multiple values, each value having a different key. For example, the key name is the name of a field (MIN, MAX and so on).

In the next example of a keyed property type, each associated value is a structure containing a single attribute. In this case the attribute is a list of double-precision expressions representing multipliers to be applied to a field:

```
<PropertyType id="multiplierOps" isKeyed="true" valueType="structure">
  <Structure>
    <Attribute name="multipliers" valueType="double" isList="true"/>
  </Structure>
</PropertyType>
```

A property declared to use the `multiplierOps` property type might be:

```
<Property name="multiplierSettings" scriptName="multiplier_settings" type="multiplierOps"/>
```

Default Values

The `DefaultValue` element is used to specify a server temporary directory, file, or both. These are created to store a model output or document output object.

Format

```
<DefaultValue>
     <ServerTempDir basename="name"/>
     <ServerTempFile basename="name"/>
</DefaultValue>
```

where `basename` (required) is the name of the temporary directory or file.

Example

```
<DefaultValue>
     <ServerTempFile basename="datatmp"/>
</DefaultValue>
```

Evaluated Strings

Some strings declared in the specification file may include references to property names. These strings are known as evaluated strings.

The syntax for a property reference is:

```
"${property_name}"
```

When an evaluated string is accessed, any property references are replaced by the value of the referenced property. If the property does not exist, an error occurs. For example, when adding a new field, you may have a property named `my_new_field` in the node definition and a control in the User Interface section that enables the user to edit the value of this property.

Example

```
<AddField name="${my_new_field}" ... >
```

Operations

Certain sections of the specification file support operations such as adding fields, creating components, and initializing properties. Sections that support operations are:

- output data model (source and process nodes)
- input and output data model (components)
- output object creation (model- and document builder nodes)
- model applier creation (model outputs)

Operations are divided into the following types:

- data model operations: `AddField`, `ChangeField`, `RemoveField`
- iteration: `ForEach`

Data Model Operations

The operations that you can perform on a data model are:

- add a new field to an existing data model
- modify an existing field in a data model
- remove a field from a data model

Add Field: The `AddField` element enables you to add a new field to an existing data model.

Format

```
<AddField prefix="prefix" name="name" direction="field_role" directionRef="field_role_ref"
     fieldRef="field_ref" group="group_id" label="label" missingValuesRef="mval_ref"
     storage="storage_type" storageRef="storage_ref" targetField="target_field"
     type="data_type" typeRef="type_ref" role="role" tag="propensity_type" value="value" >
     <Range min="min_value" max="max_value" />
</AddField>
```

The attributes for `AddField` are as follows.

Table 19. AddField attributes.

Attribute	Description
`prefix`	A prefix to be added to the field name, for example to denote a model output field.
`name`	(required) The name of the field to be added. This can be either a hard-coded string (e.g. `field8`) or an evaluated string (e.g. `${target}`) that references a field. See the topic "Evaluated Strings" on page 60 for more information.
`direction`	The field role, e.g. whether the field is an input or a target. One of `in`, `out`, `both`, `partition` or `none`.
`directionRef`	Specifies that the field direction is to be derived from the direction of the field identified by an evaluated string (e.g. `${field1}`) that references a field. See the topic "Evaluated Strings" on page 60 for more information.
`fieldRef`	Specifies that all the referenced values (`directionRef`, `missingValuesRef`, `storageRef` and `typeRef`) are to be derived from the corresponding values of the field identified by an evaluated string (e.g. `${field1}`) that references a field. See the topic "Evaluated Strings" on page 60 for more information.
`group`	Specifies that the field is a member of a field group. See the topic "Model Fields" on page 81 for more information.
`label`	A label for the field to be added.
`missingValuesRef`	Specifies that the way missing values are handled is to be derived from the missing values specification of the field identified by an evaluated string (e.g. `${field1}`) that references a field. See the topic "Evaluated Strings" on page 60 for more information.
`storage`	The data storage type for the field value - one of `integer`, `real`, `string`, `date`, `time`, `timestamp` or `unknown`.
`storageRef`	Specifies that the storage type is to be derived from the storage type of the field identified by an evaluated string (e.g. `${field1}`) that references a field. See the topic "Evaluated Strings" on page 60 for more information.

Table 19. AddField attributes (continued).

Attribute	Description
targetField	For a model output field, specifies the target field from which data for this new field is derived. This can be either a hard-coded string (e.g. field8) or an evaluated string (e.g. ${target}) that references a field. See the topic "Evaluated Strings" on page 60 for more information.
type	The data type of the field - one of auto, range, discrete, set, orderedSet, flag or typeless.
typeRef	Specifies that the data type is to be derived from the data type of the field identified by an evaluated string (e.g. ${field1}) that references a field. See the topic "Evaluated Strings" on page 60 for more information.
role	The kind of data held in a model output field - one of unknown, predictedValue, predictedDisplayValue, probability, residual, standardError, entityId, entityAffinity, upperConfidenceLimit, lowerConfidenceLimit, propensity, value or supplementary. See the topic "Roles" on page 66 for more information.
tag	Only used if role has the value propensity; indicates the propensity type, and is either RAW or ADJUSTED.
value	Specifies that the values that the new field is to contain are to be derived from the field identified by an evaluated string (e.g. ${field1}) that references a field. See the topic "Evaluated Strings" on page 60 for more information.

The attributes for Range are as follows.

Table 20. Range attributes

Attribute	Description
min	The minimum value that the field can accept.
max	The maximum value that the field can accept.

Examples

The following adds a string field named field8:

```
<AddField name="field8" storage="string" />
```

The next example shows how you can use a reference to a property name when adding a field. Here the field is added with a name that matches the value of the previously-defined property prop1:

```
<AddField name="${prop1}" ... />
```

In the following example, if the target field is named field1, the model creates an output field named $S-field1 to hold the predicted value for field1:

```
<AddField prefix="$S" name="${target}" role="predictedValue" targetField="${target}"/>
```

The next example adds a model output field that holds a probability score between 0.0 and 1.0:

```
<AddField prefix="$SC" name="${target}" storage="real" role="probability" targetField=
"${target}">
     <Range min="0.0" max="1.0"/>
</AddField>
```

In the final example, for every model output field, an output field is added that holds a probability score between 0.0 and 1.0 and which derives its value from that of the variable `fieldValue`:

```
<ForEach var="fieldValue" inFieldValues="${field}">
     <AddField prefix="$SP" name="${fieldValue}" storage="real" role="probability" targetField=
     "${field}" value="${fieldValue}">
          <Range min="0.0" max="1.0"/>
     </AddField>
</ForEach>
```

See the topic "Evaluated Strings" on page 60 for more information.

Change Field: The `ChangeField` element enables you to modify an existing field in a data model.

Format

```
<ChangeField name="name" fieldRef="field_reference" direction="field_role" storage="storage_
type" type="data_type" >
     <Range min="min_value" max="max_value" />
</ChangeField>
```

The attributes for `ChangeField` are as follows.

Table 21. ChangeField attributes.

Attribute	Description
`name`	(required) The name of the field to be changed.
`fieldRef`	A reference value for the field.
`direction`	The field role, e.g. whether the field is an input or a target. One of `in`, `out`, `both`, `partition` or `none`.
`storage`	The data storage type for the field value - one of `integer`, `real`, `string`, `date`, `time`, `timestamp` or `unknown`.
`type`	The data type of the field - one of `auto`, `range`, `discrete`, `set`, `orderedSet`, `flag` or `typeless`.

The attributes for `Range` are as follows.

Table 22. Range attributes

Attribute	Description
`min`	The minimum value that the field can accept.
`max`	The maximum value that the field can accept.

Remove Field: The RemoveField element enables you to remove a field from a data model.

Format

```
<RemoveField fieldRef="field_reference" />
```

where `fieldRef` is a reference value for the field.

Iteration with the ForEach Element

In some places, it is useful to be able to perform the same operation repeatedly to process each one of a set of values. The specification file supports a simple `ForEach` iterator that binds a temporary property to each value in the supplied set in turn. The `ForEach` loop can be set up to iterate in one of the following ways:

- between two integer values with an optional step size
- over values in a list property
- over keys in a keyed property
- over fields in a field group

Format

```
<ForEach var="field_name" from="integer_exp" to="integer_exp" step="integer_exp"
inFields="fields" inFieldValues="field_name" inProperty="property_name" >
   -- data model operation --
</ForEach>
```

where:

`var` (required) specifies the field containing the values to which the iteration will apply.

`from` and `to` specify integers (or expressions that evaluate to integers) that denote the lower and upper limits of the iteration, with the optional attribute `step` indicating an integer step size.

`inFields`, `inFieldValues` and `inProperty` are alternatives to the `from/to/step` format:

- `inFields` specifies a field set over which to perform the iteration, and is one of:
  `inputs` - the input fields for the node
  `outputs` - the output fields from the node
  `modelInput` - the input fields specified in the model signature
  `modelOutput` - the output fields specified in the model signature
- `inFieldValues` specifies a field name (or a property that represents a field name) and iterates through the values in the metadata for that field
- `inProperty` specifies the name of a property on which to perform the iteration

The data model operation that can be specified within a `ForEach` element is any of the `AddField`, `ChangeField` or `RemoveField` elements. See the topic "Data Model Operations" on page 61 for more information. `ForEach` elements can also be nested.

Examples

The following performs an operation ten times:

```
<ForEach var="val" from="1" to="10">
 ...
</ForEach>
```

The following performs an operation the number of times specified by an integer property:

```
<ForEach var="val" from="1" to="${history_count}">
 ...
</ForEach>
```

In the next example, processing iterates through the values in the output fields for the node:

```
<ForEach var="field" inFields="outputs">
 ...
</ForEach>
```

The following example iterates through the values in the metadata for the field identified by ${field}:

```
<ForEach var="fieldValue" inFieldValues="${field}">
 ...
</ForEach>
```

The next example iterates through the values in a list property:

```
<ForEach var="val" inProperty="my_list_property">
 ...
</ForEach>
```

The following iterates through the key values in a keyed property:

```
<ForEach var="key" inProperty="my_keyed_property">
 ...
</ForEach>
```

Fields and Field Metadata

Nodes, models, and data sources act as **data model providers**--they are able to define field metadata that is accessible from other objects.

Data model providers have an input data model and an output data model. An output data model can be defined in terms of the input data model, for example when extending an input model by adding a field, or when changing an existing model.

Each of these objects has slightly different requirements.

Nodes. The input data model can be referenced but is not modifiable. The output data model can be based on the input data model, or can replace it. The output data model is recomputed whenever the node properties or the input data model change. The output data model of a model applier node may also reference the output data model of the model component.

Models. By default, the input- and output data models (the model signature) are based on the input and output field settings used when the model was created. Ideally the model build process will return a metadata file that defines the required input fields and the generated output fields. Once defined, the model signature cannot be modified. However, properties in a model applier node may modify the data model output from the applier node. For example, these properties may define whether a cluster ID is returned as a string or an integer, or how many sequence IDs to generate. In addition, the model signature typically specifies outputs as having a field role (`direction`) set to `"out"`, whereas the node is likely to generate them with the field role `"in"`.

Data sources. Data sources used in data reader nodes can specify an output data model. The input data model is always empty.

Field Sets

A field set can be used in numerous places to select a subset of fields from a data model provider. The data model provider can be either the enclosing object or a container of the enclosing object. The initial state of a field filter can be either to include all available fields and then exclude specific types of fields, or to start with an empty set of fields and include the required fields or add new fields.

The following example shows how an extension node can specify the output data model. The key fields are specified by a list property called `keys` , and this is followed by an optional record count field that can be generated, whose name is also specified by a property.

```
<OutputDataModel mode="replace">
     <ForEach var="field" inProperty="keys">
          <AddField name="${field}" fieldRef="${field}"/>
     </ForEach>
```

```
     <AddField name="${record_count_name}" storage="integer">
          <Condition property="include_record_count" op="equals" value="true"/>
     </AddField>
</OutputDataModel>
```

Field Sets and Model Building

The following example shows how a model applier can use the information in the previously created model component to generate its output fields:

```
<OutputDataModel mode="modify">
     <AddField provider="model" dataModel="output">
</OutputDataModel>
```

Both `AddField` and `ForEach` specify a data model provider, as well as specifying which of the input or output data models should be used. They provide a mechanism for specifying a set (or subset) of fields from the data model provider. The default provider is `this` representing the enclosing element (i.e. not the object being created), with the input field set being used by default. If no field set is specified, all available fields are used.

Field sets can be based on storage, type, field role or names. Where they are based on names, this requires a reference to a list property. The field set can be either full (the default) or empty; the former allows fields to be excluded, while the latter allows fields to be included. Multiple values can be specified for each of the individual filters, and these values act as "intersection" or "and" operators, for example:

```
<FieldSet include="none">
     <Include direction="in" storage="string"/>
</FieldSet>
```

This starts with an empty field set (specified by `include="none"`) and will then include fields that have the field role (`direction`) set to `"in"`, and string storage.

Another example is:

```
<FieldSet include="all">
     <Exclude type="typeless"/>
</FieldSet>
```

This includes all available fields (specified by the `include="all"` attribute, which is the default behaviour) and then excludes any which have a `typeless` type. This will include fields with `direction` set to `"in"` or `"both"`.

Multiple filters can also be specified, and these act as "union" or "or" operators, for example:

```
<FieldSet include="all">
     <Exclude type="discrete" storage="real"/>
     <Exclude type="discrete" storage="integer"/>
</FieldSet>
```

This excludes fields which are either discrete with real storage, or discrete with integer storage.

Note that when including fields into an initially empty field set, the order of the `include` statements does not usually affect the order in which fields are included. That is, the fields from the field set provider are evaluated in their natural order against each condition to determine whether they should be included in the field set.

Roles

A role describes the kind of data that is held in a data model output field. A role can be specified by an `AddField` element and tested by a `Condition` element.

The possible roles are as follows.

Table 23. Model output roles

Role	Meaning
`unknown`	No role specified (should not be encountered).
`predictedValue`	This field contains the predicted value of the target field.
`probability`	The probability or confidence of the prediction.
`residual`	The residual value.
`standardError`	The standard error of the prediction.
`entityId`	The entity ID--typically represents the cluster ID in a cluster model.
`entityAffinity`	The entity affinity--typically represents the distance from the cluster centre in a model.
`upperConfidenceLimit`	The upper confidence limit of the prediction.
`lowerConfidenceLimit`	The lower confidence limit of the prediction.
`propensity`	The propensity score. An additional `tag` attribute specifies whether this refers to a raw or adjusted propensity.
`value`	Used to represent a value for models that is associated with another output (see below).
`supplementary`	Information generated by the model that is not covered by other roles.

As an example of `value`, the Anomaly Detection model generates groups of fields where each group consists of two fields, one representing a field name and the other specifying a measure of how anomalous that field is. In this case, `value` would be the field name.

Logical Operators

A number of elements can use the logical operators `And`, `Or` and `Not` to specify various kinds of processing, for example when setting compound conditions (see "Compound Conditions" on page 71).

Format

The format of the `And` element is as follows. The format of the `Or` and `Not` elements is almost identical, the only differences being the enclosing tags `<Or>...</Or>` and `<Not>...</Not>` respectively.

```
<And>
     <Condition .../>
     <And ... />
     <Or ... />
     <Not ... />
</And>
```

The `Condition` element specifies a condition to be tested. See the topic "Conditions" for more information.

Note that the `And`, `Or` and `Not` child elements can be nested.

Conditions

The behavior of some objects may be modified using conditions, which are specified by `Condition` elements (the equivalent of IF statements). For example, a node that is executed by a command can add a

condition to the execution information, such as only including a particular option if a property has a specific value. Similarly, a property control in the user interface may be enabled or visible only if another control has a specific value.

Conditions can be either simple or compound. A **simple condition** consists of the following:

- a value source (either a property or a control)
- a test
- an optional test value

A **compound condition** allows other conditions to be combined to form complex logical conditions. Compound conditions involve the use of:

- `And`
- `Or`
- `Not`

Format

```
<Condition container="container_name" control="prop_name" property="name" op="operator"
     value="value" />
```

where:

`container` specifies the name of a particular container whose value is to be tested by the condition.

`control` specifies a property control whose value is to be tested by the condition. *prop_name* is the value of the `property` attribute of the element in which the control is defined (for example, in a properties panel on a dialog tab).

`property` specifies a property whose value is to be tested by the condition. *name* is the value of the `name` attribute of the `Property` element in which the property is defined.

`op` is the condition operator. See the topic "Condition Operators" for more information.

`value` is the specific value to be tested for by the condition.

Examples

For examples of condition setting, see "Simple Conditions" on page 70 and "Compound Conditions" on page 71.

Condition Operators

A set of operators is available that addresses most conditions.

Table 24. Tests supported on any value

Operator	Value	Description
`equals`	*value*	True if the property equals the supplied value (case-sensitive for string values).
`notEquals`	*value*	True if the property does not equal the supplied value (case-sensitive for string values).
`in`	*list of values*	True if the property is in the supplied list of values.

Table 25. Tests supported on numeric values

Operator	Value	Description
`lessThan`	*number*	True if the property is less than the supplied number.
`lessOrEquals`	*number*	True if the property is less than or equal to the supplied number.
`greaterThan`	*number*	True if the property is greater than the supplied number.
`greaterOrEquals`	*number*	True if the property is greater than or equal to the supplied number.

Table 26. Tests supported on string values

Operator	Value	Description
`isEmpty`	-	True if the property has a zero-length string.
`isNotEmpty`	-	True if the property has a non-zero length string.
`startsWith`	*string*	True if the property starts with the supplied string (case-sensitive).
`startsWithIgnoreCase`	*string*	True if the property starts with the supplied string, ignoring case.
`endsWith`	*string*	True if the property ends with the supplied string (case-sensitive).
`endsWithIgnoreCase`	*string*	True if the property ends with the supplied string, ignoring case.
`equalsIgnoreCase`	*string*	True if the property equals the supplied string, ignoring case.
`hasSubstring`	*string*	True if the property contains the supplied string (case-sensitive).
`hasSubstringIgnoreCase`	*string*	True if the property contains the supplied string, ignoring case.
`isSubstring`	*string*	True if the property is a substring of the supplied string (case-sensitive).
`isSubstringIgnoreCase`	*string*	True if the property is a substring of the supplied string, ignoring case.

Table 27. Tests supported on list properties

Operator	Value	Description
`isEmpty`	-	True if the number of items in the list is 0.
`isNotEmpty`	-	True if the number of items in the list is not 0.
`countEquals`	*number*	True if the number of items in the list equals the supplied value.
`countLessThan`	*number*	True if the number of items in the list is less than the supplied value.
`countLessOrEquals`	*number*	True if the number of items in the list is less than or equal to the supplied number.
`countGreaterThan`	*number*	True if the number of items in the list is greater than the supplied number.
`countGreaterOrEquals`	*number*	True if the number of items in the list is greater than or equal to the supplied number.

Table 27. Tests supported on list properties (continued)

Operator	Value	Description
contains	*value*	True if the supplied item is in the list.

Table 28. Tests supported on field properties

Operator	Description
storageEquals	True if the storage equals the supplied value.
typeEquals	True if the type equals the supplied value.
directionEquals	True if the field role (direction) equals the supplied value.
isMeasureDiscrete	True if the field data type is discrete (i.e. can only be one of set, flag or orderedSet).
isMeasureContinuous	True if the field data type is range.
isMeasureTypeless	True if the field data type is typeless.
isMeasureUnknown	True if the field data type is unknown.
isStorageString	True if the field storage type is string.
isStorageNumeric	True if the field storage type is numeric.
isStorageDatetime	True if the field storage type is datetime.
isStorageUnknown	True if the field storage type is unknown.
isModelOutput	True if the field is a model output field.
modelOutputRoleEquals	True if the role for the field is one of the valid roles shown in the next table.
modelOutputTargetFieldEquals	True if the target field equals the specified value (string).
modelOutputHasValue	True if the field is a model output field and has a value associated with it.
modelOutputTagEquals	True if the tag equals the specified value (string).

Condition operators that support keyed properties are:

- isEmpty
- isNotEmpty
- countEquals
- countLessThan
- countLessOrEquals
- countGreaterThan
- countGreaterOrEquals
- contains

Simple Conditions

A simple condition consists of the initial value source to be tested (either a property or controller name, or an evaluated expression), the test to be performed, and optionally a value against which to perform the test.

Examples

The following evaluates to true if a Boolean property called values_grouped is true:

```
<Condition property="values_grouped" op="equals" value="true"/>
```

The next example evaluates to true if a control called values_grouped that displays Boolean values has been checked:

```
<Condition control="values_grouped" op="equals" value="true"/>
```

The following example evaluates to true if a list property called plot_fields has at least one value in it:

```
<Condition property="plot_fields" op="countGreaterThan" value="0"/>
```

The next example evaluates to true if a list property called input_fields contains only values that are instantiated:

```
<Condition property="input_fields" op="instantiated" listMode="all"/>
```

The final example evaluates to true if a list property called input_fields has at least one value that represents an uninstantiated field:

```
<Condition property="input_fields" op="uninstantiated" listMode="any "/>
```

Compound Conditions

Groups of simple conditions can be combined using logical operators.

Examples

The following evaluates to true if the Boolean values_grouped property is true and group_fields contains at least one value:

```
<And>
     <Condition property="values_grouped" op="equals" value="true"/>
     <Condition property="group_fields" op="countGreaterThan" value="0"/>
</And>
```

The following evaluates to true if the Boolean values_grouped property is true or if group_fields contains at least one value:

```
<Or>
     <Condition property="values_grouped" op="equals" value="true"/>
     <Condition property="group_fields" op="countGreaterThan" value="0"/>
</Or>
```

The following evaluates to true if group_fields contains at least one value:

```
<Not>
     <Condition property="group_fields" op="equals" value="0"/>
</Not>
```

Compound conditions may be nested to provide any combination of conditions.

Using CLEF Nodes in Scripts

You can refer to a CLEF node in a script by using the scriptName attribute of the Node element. In the same way, you can refer to a property of the node in a script by using the scriptName attribute of the Property element.

The scriptName attribute is optional in both cases, though we recommend using the attribute to avoid name clashes between extensions or properties.

If you omit the script name in a node definition, a script can reference the node by means of the value of the id attribute prefixed by the extension name. For example, given an extension named myext and which defines a data reader node with the ID import, a script will be able to reference the node as myextimport.

If you omit the script name in a property definition, a script can reference the property through the value of its `name` attribute.

For more information, see the *IBM SPSS Modeler Scripting and Automation Guide.*

Example - Edit and Execute a Node

The following example shows how you can use a script to automate the tasks of editing and executing the example data reader node shown under "Data Reader Node (Apache Log Reader)" on page 24.

In the specification file for the Apache Log Reader node, the node specification begins as follows:

```
<Node id="apachelogreader" type="dataReader" palette="import" labelKey="apacheLogReader.LABEL">
     <Properties>
          <Property name="log_filename" valueType="string" labelKey="logfileName.LABEL" />
     </Properties>
```

In the script, you would reference the node and the property like this:

```
create apachelogreader
set :apachelogreader.log_filename='installation_directory\Demos\combined_log_format.txt'
create tablenode at 200 100
connect :apachelogreader to :tablenode
execute :tablenode
```

where *installation_directory* is the directory to which IBM SPSS Modeler is installed.

Running this script:

- creates the data reader node
- specifies *combined_log_format.txt* as the Apache log file to read
- creates a table node
- connects the data reader node to the table node
- executes the table node

Example - Keyed Properties

Keyed properties support standard scripting syntax. For example, the structure shown in the first example of keyed property types under "Structured Properties" on page 58. could be defined in a script as:

```
set :mynode.aggregation_settings.Age = {true true false false false}
```

A single attribute could be modified as follows:

```
set :mynode.aggregation_settings.Age.MIN = true
```

Maintaining Backward Compatibility

When planning updates to an existing extension, take care to maintain compatibility with a previously-distributed version of that extension. Some changes will have no adverse effects, some involve a significant risk, and others are known to break compatibility and should be avoided.

Changes With No Risk

The following changes will not affect backward compatibility:

- adding new `Node`, `ModelOutput`, `DocumentOutput` or `InteractiveModelBuilder` elements
- adding new `Property` definitions and their associated new controls to these elements

- adding new containers to these elements*
- adding new values to an existing enumeration property

* Note that any code using these new containers should allow for the fact that these containers will be empty for objects created using the previous version of the extension.

Changes With Significant Risk

Changes to existing declarations carry a significant risk of breaking compatibility. These should be tested carefully before distribution.

Changes to Avoid

The following changes are known to break compatibility and should be avoided:

- changing the value of the `id` or `providerTag` attributes in the `ExtensionDetail` element
- changing the value of the `id` attribute in a `Node`, `ModelOutput`, `DocumentOutput` or `InteractiveModelBuilder` element
- removing a `Node`, `ModelOutput`, `DocumentOutput` or `InteractiveModelBuilder` element from the extension
- changing the value of the `valueType` attribute of a `Property` or `PropertyType` element

Chapter 5. Building Models and Documents

Introduction to Model and Document Building

The standard IBM SPSS Modeler modules include nodes that enable users to generate (or "build") a wide variety of models and graphs. CLEF gives you the ability to define additional nodes to build other models and documents (graphs and reports) that are not provided as standard.

When defining model builder or document builder nodes, you also need to define the objects that are produced when these nodes are executed. You do this by means of items known as "constructors."

The following sections describe this process in detail.

Models

A **model** is a set of rules, a formula, or an equation that can be used to predict an outcome based on a set of input fields. The ability to predict an outcome is the central goal of predictive analytics. In IBM SPSS Modeler you achieve this by:

- Generating a model from existing data
- Applying the generated model to data to make the predictions

The process of generating a model is also known as "building" a model, and in IBM SPSS Modeler you do this by means of a modeling node. In CLEF, modeling nodes are referred to as **model builder nodes**, a name derived from the syntax of the XML statement used to define them. See the topic “Model Builder Nodes” on page 11 for more information.

The process of applying a model to data is known as "scoring the data." Doing so enables you to use the information gained from model building to make predictions for new records. In IBM SPSS Modeler, you do this by adding the icon of a generated model to the stream canvas. The icon takes the form of a gold nugget, so a generated model is referred to in IBM SPSS Modeler as a "model nugget." In CLEF, a model nugget on the Models tab of the manager pane is known as a **model output object**, and when added to the canvas it is known as a **model applier node**. See the topic “Model Applier Nodes” on page 12 for more information.

Documents

In some cases, you will want to generate an object other than a model, such as a graph or report output. In IBM SPSS Modeler, these objects are known as **documents**, and they are generated by means of a **document builder node**. See the topic “Document Builder Nodes” on page 12 for more information.

Constructors

Constructors define the objects that are produced as a result of either executing a node in a stream or generating an object back to the stream.

Constructors can be defined for any of the following:

- Model builder node
- Document builder node
- Model applier node
- Model output object

In the case of a **model builder** or **document builder** node, a constructor enables these nodes to define how the output object is generated when the node is executed. An output object definition may include multiple properties and components, and the Constructors section defines how these are initialized or created from the object generated by the execution.

For a **model applier** node, a constructor defines what kind of object the node can generate back to the stream or the Models tab.

In the case where constructors are defined for a **model output object**, they can:

- Specify what model applier node to create when the model output object is dropped onto the stream canvas
- Generate a model builder node with the settings that were used to create the model output object

See the topic "Using Constructors" on page 95 for more information.

Building Models

When specifying a node from which a model can be generated (that is, a model builder node), you need to define how the node communicates with the Model Builder component of IBM SPSS Modeler, which is the process that actually creates the model. You do so in the definition of a `Node` element in the specification file.

Unless otherwise specified, model building begins as soon as the end user clicks the **Execute** button in the model builder node dialog box. However, it is also possible to define an **interactive model**, whereby the end user can refine or modify data values after clicking **Execute** but before the model is actually built. Interactive model building additionally requires the inclusion of specific elements through which the interactivity is defined. See the topic "Building Interactive Models" on page 84 for more information.

When defining a model builder node, the `Node` element must include:

- A `type="modelBuilder"` attribute
- A `ModelBuilder` child element
- A `Constructors` child element containing a `CreateModelOutput` element (see "Using Constructors" on page 95)

For the format of a `Node` element specification, see "Node" on page 45.

Note: In the element definitions in subsequent sections (normally identified by the heading **Format**), element attributes and child elements are optional unless indicated as "(required)." For complete element syntax, see "CLEF XML Schema," on page 195.

For model building, the extension also needs a `ModelOutput` element to describe the generated model (see "Model Output" on page 83). The `ModelOutput` element needs to include a `Constructors` child element that contains a `CreateModelApplier` definition. See the topic "Create Model Applier" on page 97 for more information.

Model Builder

The `ModelBuilder` element defines the behavior of a model builder node. This is done by means of the element attributes and one or more child elements.

Format

```
<ModelBuilder allowNoInputs="true_false" allowNoOutputs="true_false" nullifyBlanks="true_false"
     miningFunctions="[function1 function2 ... ]" >
     <Algorithm ... />
     <ModelingFields ... />
```

```
    <ModelGeneration ... />
    <ModelFields ... />
    <AutoModeling ... />
</ModelBuilder>
```

where:

- `allowNoInputs` and `allowNoOutputs` must be used explicitly in a case where you want to build a model that has either no input fields or no output fields, respectively.
- `nullifyBlanks`, if set to `false`, turns off the feature whereby blank values are replaced with null values (represented by `$null$`) in the data that is passed to the Model Builder component of IBM SPSS Modeler. The default is to replace blanks with nulls, but you might want to deactivate this feature, for example, if your algorithm needs to treat blanks differently from nulls.
- `miningFunctions` (required) identifies the data mining function or functions that the model performs.

Table 29. Data mining functions.

Function	Description
`classification`	Predicts a discrete (that is, `set`, `flag`, or `orderedSet` data type) value of an unknown target attribute from records with known target values.
`approximation`	Predicts a continuous (that is, `range` data type) value of an unknown target attribute from records with known target values.
`clustering`	Identifies groups of similar records and labels them accordingly.
`association`	Identifies related events or attributes in data.
`sequence`	Searches for sequential patterns in time-structured data.
`reduction`	Reduces the complexity of data—for example, via derived fields that summarize the contents of the original data fields.
`conceptExtraction`	Used in text mining.
`categorize`	Used in text mining.
`timeSeries`	Forecasts future values from patterns in past data.
`anomalyDetection`	Searches for unusual cases based on deviations from the norms of their cluster groups.
`attributeImportance`	Identifies the attributes that have the greatest influence on a target attribute.
`supervisedMultiTarget`	Estimates the likelihood of a (*yes* or *no*) outcome for one of a number of possibilities.

If the model performs more than one function, the function names are separated by spaces within the square brackets, as in the following example:

```
<ModelBuilder miningFunctions="[classification approximation]">
 ...
</ModelBuilder>
```

Child Elements

The child elements of the `ModelBuilder` element are shown in the following table.

Table 30. Child elements of model builder declaration.

Child element	Description	See...
`Algorithm`	(required) Specifies the algorithm used to generate the model.	"Algorithm" on page 78

Table 30. Child elements of model builder declaration (continued).

Child element	Description	See...
`ModelingFields`	Specifies the identifier to be used subsequently in the User Interface section to define the location of the input and output field controls for the model. The controls themselves are defined in the `InputFields` and `OutputFields` child elements of `ModelingFields`.	"Modeling Fields"
`ModelGeneration`	Specifies the identifier to be used subsequently in the User Interface section to define the location of the model name controls for the generated model.	"Model Generation" on page 81
`ModelFields`	Specifies the set of input and output fields used for scoring data with this model.	"Model Fields" on page 81
`AutoModeling`	Enables this model to be used by an ensemble modeling node, such as Auto Classifier, Auto Cluster, or Auto Numeric.	"Automated Modeling" on page 88

Algorithm

The `Algorithm` element defines details of the algorithm used to generate the model.

```
<Algorithm value="model_output_id" label="display_label" labelKey="label_key"/>
```

where:

- `value` (required) is the internal name of the algorithm. This is referenced in a number of other places in the specification file. See the topic "Model Builder Example" on page 82 for more information.
- `label` (required) is a description of the algorithm.
- `labelKey` identifies the label for localization purposes.

Modeling Fields

The standard way in which the model input and output fields are specified is by means of the Type node. Users set the field role to **in** or **out** as desired. Optionally, for a model builder node you can give users the choice of overriding the settings in an upstream Type node and using custom settings.

This is done by means of the `ModelingFields` element. This element specifies an identifier that is used subsequently in the User Interface section of the model builder node declaration to define the location of the controls for the input and output fields for the model. The controls themselves are defined by means of `InputFields` and `OutputFields` child elements.

Format

```
<ModelingFields controlsId="control_identifier" ignoreBOTH="true_false" >
      <InputFields ... />
      <OutputFields ... />
</ModelingFields>
```

where:

- `controlsId` (required) is the identifier to be used subsequently in a `SystemControls` element in the User Interface section of the model builder node declaration. Doing so identifies which tab of the node dialog box is to contain the input and output field controls for the model.
- `ignoreBOTH`, if set to `true` (default), specifies that fields with a field role set to **both** are ignored by the model.

The `InputFields` and `OutputFields` elements are described in the sections beginning at "Input Fields" on page 79.

Example

This example illustrates the use of a set of modeling fields controls on a Fields tab of a model builder node dialog box. First, an identifier is specified for the set of controls:

```
<ModelBuilder miningFunctions="[classification]">
 ...
     <ModelingFields controlsId="modelingFields">
          <InputFields property="inputs" onlyNumeric="true" multiple="true" label="Inputs"
            labelKey="inputFields.LABEL"/>
          <OutputFields property="target" multiple="false" types="[set flag]" label="Target"
            labelKey="targetField.LABEL"/>
     </ModelingFields>
 ...
</ModelBuilder>
```

The `modelingFields` identifier is subsequently referenced in the User Interface section of the node dialog box, at the point where the Fields tab is defined:

```
<UserInterface ... >
     <Tabs defaultTab="1">
          <Tab label="Fields" labelKey="Fields.LABEL" helpLink="modeling_fieldstab.htm">
               <PropertiesPanel>
                    <SystemControls controlsId="modelingFields">
                    </SystemControls>
               </PropertiesPanel>
          </Tab>
 ...
</UserInterface>
```

Input Fields: The `InputFields` element defines the set of fields from which users will be able to select one or more input fields (that is, predictors) for the model.

The set consists of all the fields that are visible at this node. If fields have been filtered further upstream from this node, only the fields that have passed through the filter are visible. The list can also be further restricted by specifying that only fields with particular storage and data types are to be available for selection.

```
<InputFields storage="storage_types" onlyNumeric="true_false" onlySymbolic="true_false"
     onlyDatetime="true_false" types="data_types" onlyRanges="true_false"
     onlyDiscrete="true_false" property="property_name" multiple="true_false" label="label"
     labelKey="label_key"/>
```

You can restrict the list of fields to be used as input fields by specifying two attributes, one of which must be from the following list:

- `storage` is a list property that specifies the storage type of fields to be allowed in the list--for example, `storage="[integer real]"` means that only fields with these storage types will be listed. For the set of possible storage types, see the table under "Data and Storage Types" on page 175.
- `onlyNumeric`, if set to `true`, specifies that only fields with a numeric storage type are listed.
- `onlySymbolic`, if set to `true`, specifies that only fields with a symbolic (that is, string) storage type are listed.
- `onlyDatetime`, if set to `true`, specifies that only fields with a date-and-time storage type are listed.

The second attribute specified must be from this list:

- `types` is a list property that specifies the data type of fields to be allowed in the list--for example, `types="[range flag]"` means that only fields with these storage types will be listed. The set of possible data types is:

  `range`

flag

set

orderedSet

numeric

discrete

typeless

- onlyRanges, if set to true, specifies that only fields with a range data type are listed.
- onlyDiscrete, if set to true, specifies that only fields with a discrete (that is, flag, set, or typeless) data type are listed.

Thus, for example, a control that specifies storage="[integer]" and types="[flag]" ensures that only integer fields that are flags will appear in the list.

The remaining attributes are as follows:

- property is the identifier of the property to be used to store the field values.
- multiple specifies whether the field values are an enumerated list (true) or not (false).
- label is the display name of the control.
- labelKey identifies the label for localization purposes.

Output Fields: The OutputFields element defines the set of fields from which users will be able to select one or more output fields (that is, targets) for the model.

The set consists of all the fields that are visible at this node. If fields have been filtered further upstream from this node, only the fields that have passed through the filter are visible. The list can also be further restricted by specifying that only fields with particular storage and data types are to be available for selection.

```
<OutputFields storage="storage_types" onlyNumeric="true_false" onlySymbolic="true_false"
     onlyDatetime="true_false" types="data_types" onlyRanges="true_false"
     onlyDiscrete="true_false" property="property_name" multiple="true_false" label="label"
     labelKey="label_key"/>
```

You can restrict the list of fields to be used as output fields by specifying two attributes, one of which must be from the following list:

- storage is a list property that specifies the storage type of fields to be allowed in the list--for example, storage="[integer real]" means that only fields with these storage types will be listed. For the set of possible storage types, see the table under "Data and Storage Types" on page 175.
- onlyNumeric, if set to true, specifies that only fields with a numeric storage type are listed.
- onlySymbolic, if set to true, specifies that only fields with a symbolic (that is, string) storage type are listed.
- onlyDatetime, if set to true, specifies that only fields with a date-and-time storage type are listed.

The second attribute specified must be from this list:

- types is a list property that specifies the data type of fields to be allowed in the list--for example, types="[range flag]" means that only fields with these storage types will be listed. The set of possible data types is:

  range

  flag

  set

  orderedSet

  numeric

discrete

typeless

- onlyRanges, if set to true, specifies that only fields with a range data type are listed.
- onlyDiscrete, if set to true, specifies that only fields with a discrete (that is, flag, set, or typeless) data type are listed.

Thus, for example, a control that specifies storage="[integer]" and types="[flag]" ensures that only integer fields that are flags will appear in the list.

The remaining attributes are as follows:

- property is the identifier of the property to be used to store the field values.
- multiple specifies whether the field values are an enumerated list (true) or not (false).
- label is the display name of the control.
- labelKey identifies the label for localization purposes.

Model Generation

The ModelGeneration element specifies the identifier to be used elsewhere in the file to define which tab of the model builder node dialog box is to contain the model name controls for the generated model.

The format is:

```
<ModelGeneration controlsId="control_identifier" />
```

The controlsId attribute specifies the identifier to be used subsequently in a SystemControls element in the User Interface section of the model builder node specification. The tab whose specification includes this SystemControls element is the one that will contain the model name controls.

Model Fields

The ModelFields element is used to construct the **model signature**--the set of input and output fields used for scoring data with this model.

```
<ModelFields inputDirections="[in]" outputDirections="[out]">
     <AddField prefix="field_prefix" ... />
      ...
     <ForEach ... >
          <AddField prefix="field_prefix" ... />
     </ForEach>
      ...
</ModelFields>
```

where inputDirections and outputDirections specify how the model signature is to be built; values can be in, out or both.

The fields themselves are specified by one or more AddField elements. The prefix attribute specifies the prefix to be added to a field name to denote a field generated by the model. For example, if the field name is field1 and the prefix value is $S, the generated field is named $S-field1. See the topic "Add Field" on page 61 for more information.

Iteration is possible by means of ForEach elements. See the topic "Iteration with the ForEach Element" on page 64 for more information.

Field groups enable you to group together two or more output fields from the model for iteration purposes. Output field names are appended with a suffix indicating the iteration, for example $S-field1-1, $S-field1-2 and so on. One use of field groups is to cause the same set of fields to appear multiple times in the model output. See the topic "Field Group Example" on page 82 for more information.

Automodeling

The AutoModeling element enables the model to be used by an ensemble modeling node, such as Auto Classifier, Auto Cluster, or Auto Numeric. See the topic "Automated Modeling" on page 88 for more information.

Model Builder Example

The following shows the complete Model Builder section in the specification file for the Interaction node example (see "Model Builder Node (Interaction)" on page 25):

```
<Node id="interaction.builder" type="modelBuilder" palette="modeling" label="Interaction">
     <ModelBuilder miningFunctions="[classification]">
          <Algorithm value="robd" label="Robert's Algorithm" />
          <ModelingFields controlsId="modellingFields">
               <InputFields property="inputs" multiple="true" label="Inputs"
                onlyDiscrete="true" />
               <OutputFields property="target" multiple="false" label="Target"
                onlyDiscrete="true" />
          </ModelingFields>
          <ModelFields inputDirections="[in]" outputDirections="[out]">
               <ForEach var="field" inFields="outputs">
                    <AddField prefix="$I" name="${field}" fieldRef="${field}" role=
                     "predictedValue" targetField="${field}" />
                    <AddField prefix="$IP" name="${field}" storage="real" role="probability"
                     targetField="${field}">
                         <Range min="0.0" max="1.0"/>
                    </AddField>
               </ForEach>
          </ModelFields>
     </ModelBuilder>
 ...
</Node>
```

Field Group Example

This example is taken from the SLRM node, and adds a group of two new fields to the model signature in order to contain the data generated when the model is scored. For each input record, the data are scored for each of the new fields a user-specified number of times, determined by the value of the `max_predictions` property.

The two new fields are:

- $S-*target* – contains the predicted value of the target field
- $SC-*target* – contains the probability value for this prediction

In order to group these two fields together, they are assigned the same group identifier when they are declared in the `ModelFields` section. Group identifiers are assigned by means of the `group` attribute of the `AddField` element.

Thus the declaration for the model builder node contains:

```
<Node ... type="modelBuilder" ... >
     <ModelBuilder ... >
     ...
          <ModelFields inputDirections="[in]" outputDirections="[out]">
               <AddField prefix="$S" name="${target}" fieldRef="${target}" role=
                "predictedValue" targetField="${target}" group="[1]"/>
               <AddField prefix="$SC" name="${target}" storage="real" role="probability"
                  targetField="${target}" group="[1]">
                    <Range min="0.0" max="1.0"/>
```

```
            </AddField>
        </ModelFields>
    </ModelBuilder>
</Node>
```

The declaration for the model applier node contains:

```
<Node ... type="modelApplier" ... >
...
    <OutputDataModel mode="extend">
        <ForEach var="group" from="1" to="${max_predictions}">
            <ForEach var="field" inFields="modelOutputs" container="model">
                <AddField name="${field}" group="[${group}]" fieldRef="${field}" />
            </ForEach>
        </ForEach>
    </OutputDataModel>
</Node>
```

The target field is named **campaign**, and the user inputs 2 in the field corresponding to the `max_predictions` property. Executing the model builder node causes the following fields to be appended to the model:

- `$S-campaign-1`
- `$SC-campaign-1`
- `$S-campaign-2`
- `$SC-campaign-2`

Model Output

The `ModelOutput` element describes a model output object--an object that will appear under the Models tab in the manager pane following execution of a stream.

Format

```
<ModelOutput id="identifier" label="display_label" labelKey="label_key" >
    <ModelProvider ... />
    <Properties>
        <Property ... />
         ...
    </Properties>
    <Containers ... />
    <UserInterface ... />
    <Constructors ... />
</ModelOutput>
```

where:

- `id` (required) is a unique identifier for the generated model.
- `label` (required) is the display name for the generated model as it is to appear on the Models tab.
- `labelKey` identifies the label for localization purposes.

The child elements that can be contained within the `ModelOutput` element are shown in the following table.

Table 31. Child elements of model output declaration.

Child element	Defines	See...
`ModelProvider`	The container that is to hold the model output, and whether the output is in PMML format.	"Model Provider" on page 48
`Properties`	Properties to be used by the generated model.	"Properties" on page 48

Table 31. Child elements of model output declaration (continued).

Child element	Defines	See...
`Containers`	Containers in which generated model output will be placed.	"Containers" on page 50
`UserInterface`	User interface through which generated model output can be viewed—for example, Model and Settings tabs on a model output object.	"User Interface" on page 51
`Constructors`	Objects produced by the generated model.	"Using Constructors" on page 95

Example

```
<ModelOutput id="interaction.model" label="Interaction Model">
     <Properties>
     </Properties>
     <Containers>
          <Container name="model" />
     </Containers>
     <UserInterface>
          <Tabs>
               <Tab label="Model">
                    <TextBrowserPanel container="model" textFormat="plainText" />
               </Tab>
          </Tabs>
     </UserInterface>
     <Constructors>
          <CreateModelApplier type="interaction.applier">
               <SetContainer target="model" source="model" />
          </CreateModelApplier>
     </Constructors>
</ModelOutput>
```

Building Interactive Models

Interactive modeling is a feature that enables you to create an output object with which the end user can interact before generating a model. This interactive output object is placed on the Outputs tab of the manager pane and contains an interim dataset. The interim dataset can be used to refine or simplify the model before it is generated. Interactive modeling is achieved by adding some extra elements to the specification of a regular model builder node:

- The `Constructors` section of the `Node` definition includes a `CreateInteractiveModelBuilder` element.
- The extension includes a dedicated `InteractiveModelBuilder` element.

User interaction with the interim dataset takes place by means of a window known as the **interaction window**, which is displayed as soon as the output object is created.

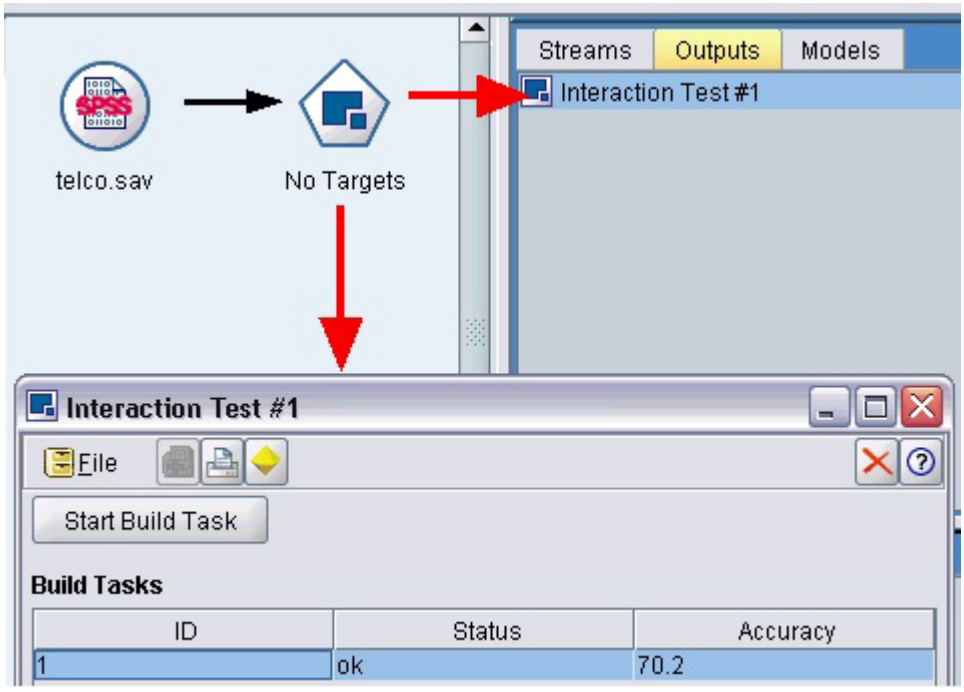

Figure 30. Interactive output object and interaction window

Interaction is specific to the algorithm used and is implemented by the extension. The interaction window is defined in the User Interface section of the `InteractiveModelBuilder` element. You can define an interaction window by specifying either of the following:

- A frame class (see "User Interface Section" on page 100), which defines the window completely
- A panel class, specified as an attribute of an extension object panel (see "Extension Object Panel" on page 111), for each tab of the window

Once closed, the interaction window can be redisplayed by double-clicking the object name on the Outputs tab.

The interaction window specification needs to include code to generate the model once the user has completed interaction. In the example illustrated, this is done by means of the toolbar button with the gold nugget icon, which is associated with an action to generate the model. The code for this is shown in the `InteractiveModelBuilder` section under "Example of Interactive Modeling" on page 87.

Create Interactive Model Builder

The `CreateInteractiveModelBuilder` element describes the output object with which the user will interact. It is effectively an interactive version of the `CreateModelOutput` element.

Format

This element is used in the Execution section of a model builder node definition:

```
<Node ... type="modelBuilder" ... >
 ...
    <Execution>
     ...
        <Constructors>
         ...
            <CreateInteractiveModelBuilder ... >
              ...
            </CreateInteractiveModelBuilder>
```

```
          </Constructors>
     </Execution>
 ...
</Node>
```

The format of the element itself is:

```
<CreateInteractiveModelBuilder type="output_object_id">
     <Condition ... ./>
     <And>
     <Or>
     <Not>
     <CreateModel type="model_id" target="container_id" sourceFile="container_file_id" />
     <CreateDocument type="model_id" target="container_id" sourceFile="container_file_id" />
</CreateInteractiveModelBuilder>
```

where `type` (required) is the identifier of the output object that is created by the `InteractiveModelBuilder` element.

The `Condition` section enables you to specify one or more conditions. See the topic "Conditions" on page 67 for more information.

You can also specify complex conditions involving the `And`, `Or` and `Not` operators. See the topic "Logical Operators" on page 67 for more information.

In the `CreateModel` and `CreateDocument` elements:

- `type` is the identifier of the model or document being defined.
- `target` (required) is the identifier of the container for the model; this container is defined in a Model Output section. See the topic "Model Output" on page 83 for more information.
- `sourceFile` (required) is the identifier of an output file that is generated during node execution; this file is defined in an Output Files section. See the topic "Output Files" on page 53 for more information.

Example

```
<CreateInteractiveModelBuilder type="my.interaction">
     <Condition property="interactive" op="equals" value="true" />
</CreateInteractiveModelBuilder>
```

This example specifies that an output object with the identifier `my.interaction` will be created on execution of the model builder node for which the specification contains this element. The output object itself is defined elsewhere in the specification file, by an `InteractiveModelBuilder` element referencing this identifier—for example:

```
<InteractiveModelBuilder id="my.interaction" label=...>
 ...
</InteractiveModelBuilder>
```

Interactive Model Builder

This element defines an interactive output object, which enables an end user to refine or simplify a model before generating it.

The `InteractiveModelBuilder` element follows the definition of a model builder node that contains the corresponding `CreateInteractiveModelBuilder` element. See the topic "Create Interactive Model Builder" on page 85 for more information.

Format

The format of the `InteractiveModelBuilder` element is:

```
<Node ... type="modelBuilder" ... >
 ...
     -- Create Interactive Model Builder section --
 ...
</Node>
 ...
<InteractiveModelBuilder id="identifier" label="display_label" labelKey="label_key">
     <Properties>
          <Property name=... />
           ...
     </Properties>
     <Containers>
          <Container name="container_name"/>
     </Containers>
     <UserInterface ... />
     <Constructors ... />
</InteractiveModelBuilder>
```

where:

- `id` (required) is a unique identifier for the generated model.
- `label` (required) is the display name for the generated model as it is to appear on the Models tab.
- `labelKey` identifies the label for localization purposes.

For more information about the `Properties`, `Containers`, `UserInterface` and `Constructors` elements, see "Properties" on page 48, "Containers" on page 50, "User Interface Section" on page 100, and "Using Constructors" on page 95.

Example of Interactive Modeling

This example illustrates how you can define a model builder node in such a way that users can choose via a simple check box whether or not to interact before generating the model.

To see this working in practice, use the Interaction example node provided with this release. See the topic "Model Builder Node (Interaction)" on page 25 for more information.

First, the model builder node specifies a Boolean property:

```
<Node id="interaction.builder" type="modelBuilder" ... >
 ...
     <Properties>
      <Property name="interactive" valueType="boolean" />
    </Properties>
```

In the User Interface section of the node specification, the section that defines the Model tab includes a reference to this property:

```
<Tab label="Model">
     <PropertiesPanel>
          <CheckBoxControl property="interactive" label="Start an interactive session" />
     </PropertiesPanel>
</Tab>
```

In the `CreateInteractiveModelBuilder` section for the same node, the setting of the property is tested and, if `true`, an interactive output object is created:

```
<CreateInteractiveModelBuilder type="my.interaction">
     <Condition property="interactive" op="equals" value="true" />
</CreateInteractiveModelBuilder>
```

The output object to which it refers is defined in the `InteractiveModelBuilder` section of the extension:

```
<InteractiveModelBuilder id="my.interaction" label="Interaction Test">
     <Properties>
     </Properties>
     <Containers>
     </Containers>
     <UserInterface actionHandler="ui.InteractionHandler">
          <Controls>
               <ToolbarItem action="generateModel" showLabel="false" />
          </Controls>
          <Tabs>
               <Tab label="Model">
                    <ExtensionObjectPanel id="model.panel" panelClass=
                     "ui.SampleInteractionPanel" />
               </Tab>
               <Tab label="Generic">
                    <ExtensionObjectPanel id="generic.panel" panelClass=
                     "ui.GenericInteractionPanel" />
               </Tab>
          </Tabs>
     </UserInterface>
</InteractiveModelBuilder>
```

The action to generate the model is controlled by the toolbar button defined by the `ToolbarItem` element.

Note the use of the `panelClass` attribute of the `ExtensionObjectPanel` element to specify a Java class to control the user interface for each tab of the interaction window.

Automated Modeling

IBM SPSS Modeler provides as standard a group of ensemble modeling nodes, such as the Auto Classifier, Auto Cluster, and Auto Numeric nodes. These nodes automate the building of a number of different models concurrently, allowing the end user to compare the results and choose the best model for their data. CLEF provides the `AutoModeling` element to enable a model specified by the `ModelBuilder` element to be used by any of these ensemble nodes.

The format of the `AutoModeling` element is:

```
<AutoModeling enabledByDefault="true_false">
     <SimpleSettings ... />
     <ExpertSettings ... />
     <Constraint ... />
     <Constraint ... />
      ...
</AutoModeling>
```

where `enabledByDefault` specifies whether the model is enabled for use by default in the ensemble modeling node (that is, the **Use?** column is checked by default for that particular model). If this attribute is omitted, the value `true` is assumed.

In an ensemble modeling node dialog box, the Expert tab lists the models that users can choose to build.

Clicking **Specify** in the **Model parameters** field for a particular model displays an Algorithm Settings dialog box, where the user can choose options for that model type.

The Algorithm Settings dialog box contains Simple and Expert tabs, corresponding to the Simple and Expert execution modes of the modeling node. The contents of the Simple and Expert tabs are controlled by the `SimpleSettings` and `ExpertSettings` elements, which are described in the sections that follow.

In addition, Constraint elements enable you to specify conditions that allow the end user to edit, or in some way constrain, parameters in the Algorithm Settings dialog box. See the topic "Constraints" on page 91 for more information.

Some parameters in the Algorithm Settings dialog box can take multiple values. Where multiple values are specified, the ensemble node will attempt to build models for all possible combinations of parameter values. For example, with a Generalized Linear model, if the user specifies two distributions (normal and gamma) and three link functions (identity, log, and power), the Auto Numeric node attempts to build six Generalized Linear models, one for each possible combination of those parameters.

Simple Settings

The SimpleSettings element determines which parameters are to appear on the Simple tab of the Algorithm Settings dialog box for this model in an ensemble modeling node. See the topic "Automated Modeling" on page 88 for more information.

Format

```
<SimpleSettings>
     <PropertyGroup label="group_name" labelKey="resource_key" properties="[prop_name1
      prop_name2 ...]"/>
     <PropertyGroup ... />
</SimpleSettings>
```

In a PropertyGroup element (at least one is required):

label is a display label for the property group, inserted as a subheading into the dialog box ahead of the first parameter in the group.

labelKey identifies the label for localization purposes. If neither label nor labelKey is used, no subheading is inserted.

properties (required) is a list of one or more properties that are to appear on the tab. The value of *prop_name1*, *prop_name2*, etc., is the value of the name attribute of the Property element in which this property is defined. See the topic "Properties" on page 48 for more information.

Example

```
<SimpleSettings>
     <PropertyGroup properties="[method]"/>
</SimpleSettings>
```

This example from the Discriminant node specifies that only the **Method** parameter will appear on the Simple tab of the Algorithm Settings dialog box for that model in the relevant ensemble modeling node (in this case, the Auto Classifier node). Because no label or labelKey attribute is specified, no subheading for the parameter is displayed in the dialog box.

Expert Settings

The ExpertSettings element determines which parameters are to appear on the Expert tab of the Algorithm Settings dialog box for this model in an ensemble modeling node. See the topic "Automated Modeling" on page 88 for more information.

Format

```
<ExpertSettings>
     <Condition ... />
     <PropertyGroup label="group_name" labelKey="resource_key"
```

```
      properties="[property1 property2 ...]"/>
     <PropertyGroup ... />
      ...
</ExpertSettings>
```

The `Condition` element specifies a condition which, if `true`, causes the parameters identified by the subsequent `PropertyGroup` element or elements to be enabled. See the topic "Conditions" on page 67 for more information.

In a `PropertyGroup` element (at least one is required):

`label` is a display label for the property group, inserted as a subheading into the dialog box ahead of the first parameter in the group.

`labelKey` identifies the label for localization purposes. If neither `label` nor `labelKey` is used, no subheading is inserted.

`properties` (required) is a list of one or more properties that are to appear on the tab. The value of *prop_name1*, *prop_name2*, etc., is the value of the `name` attribute of the `Property` element in which this property is defined. See the topic "Properties" on page 48 for more information.

Examples

In the following example, the Expert tab of the Algorithm Settings dialog box has the **Mode** parameter initially set to **Simple**.

Figure 31. Expert settings disabled

The following specifies that the other Expert tab parameters are enabled only if the user changes the **Mode** parameter setting to **Expert**:

```
<ExpertSettings>
     <Condition property="mode" op="equals" value="Expert"/>
     <PropertyGroup properties="[mode prior_probabilities covariance_matrix]"/>
      ...
</ExpertSettings>
```

Changing the **Mode** setting to **Expert** enables the two parameters in the property group.

Figure 32. Expert settings enabled

The next example illustrates the use of labels for the property group:

```
<ExpertSettings>
      ...
     <PropertyGroup labelKey="automodel.stepping_criteria_options"
      properties="[stepwise_method V_to_enter criteria]"/>
      ...
</ExpertSettings>
```

Here, the PropertyGroup element controls the parameters highlighted in the following illustration.

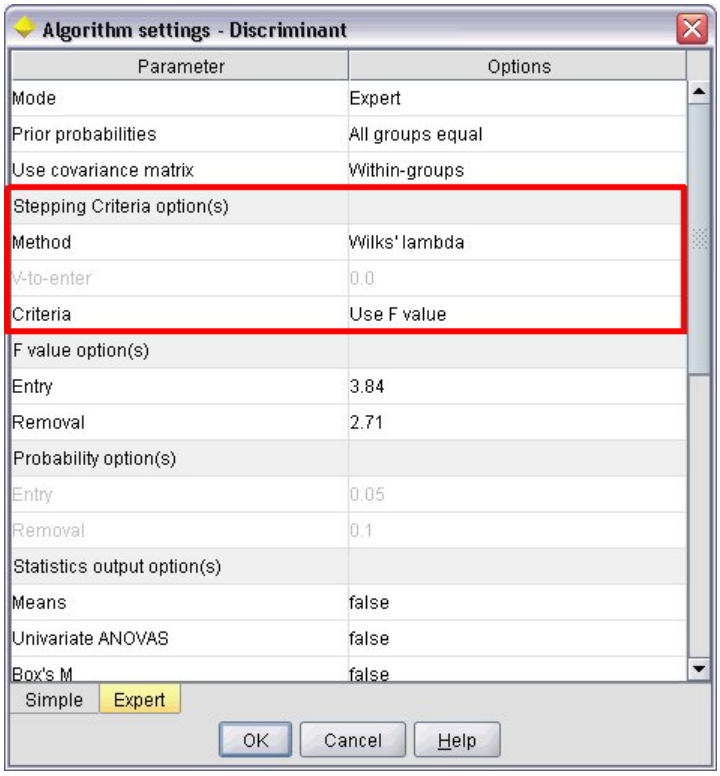

Figure 33. Expert settings disabled

The labelKey attribute enables CLEF to retrieve the display text for the property group subheading from the corresponding entry in the property file for the extension:

```
automodel.stepping_criteria_options=Stepping Criteria option(s)
```

See the topic "Property Files" on page 159 for more information.

Constraints

The Constraint element specifies the conditions that permit editing, or possible constraint in some way, of the parameters listed in the Algorithm Settings dialog box for a model in an ensemble modeling node. For example, certain parameters can be disabled if the end user is not currently permitted to modify them.

Format

```
<Constraint property="prop_name" singleSelection="true_false">
     <Condition property="prop_name" op="operator" value="value"/>
     <And ... />
     <Or ... />
     <Not ... />
</Constraint>
```

where:

- `property` (required) identifies a parameter to be edited or constrained. *prop_name* is the value of the `name` attribute of the `Property` element in which the property corresponding to this parameter is defined. See the topic "Properties" on page 48 for more information.
- `singleSelection` controls whether the end user can select more than one of the available values for a parameter. If set to `true`, only one value can be selected even if more than one value is listed in the Options field for that parameter in the Algorithm Settings dialog box. If set to `false` (default), the user can choose one or more of the available values, as illustrated in the example that follows.

The `Condition` element specifies the actual constraint. See the topic "Conditions" on page 67 for more information.

The `And`, `Or`, and `Not` elements can be used to specify compound conditions. See the topic "Logical Operators" on page 67 for more information.

Example

This example is taken from the specification file for the Generalized Linear node. Even in Expert mode, some parameters are not enabled by default.

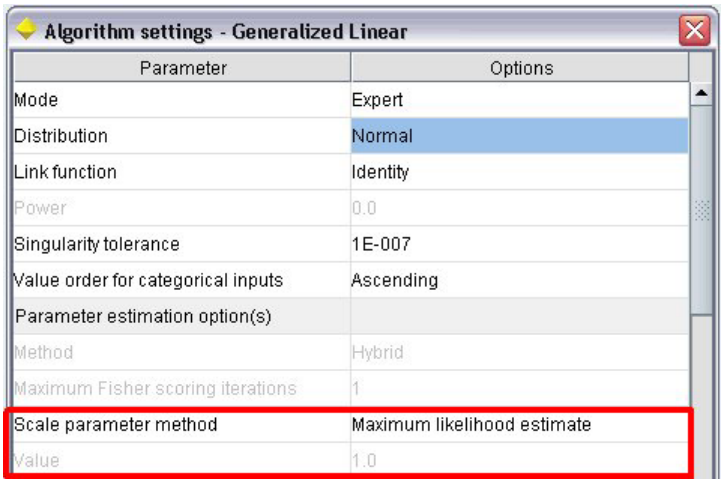

Figure 34. Effect of a constraint—parameter disabled

The constraint specifies the conditions under which the Value parameter is enabled:

```
<Constraint property="scale_value">
     <And>
          <Condition property="scale_method" op="equals" value="FixedValue"/>
          <Condition property="distribution" op="in" value="[IGAUSS GAMMA NORMAL]"/>
     </And>
</Constraint>
```

The **Scale parameter method** parameter (identified by the `scale_method` property) must be set to **Fixed value**, and the **Distribution** must be **Normal**, **Inverse Gaussian**, or **Gamma** before the **Value** parameter can be enabled.

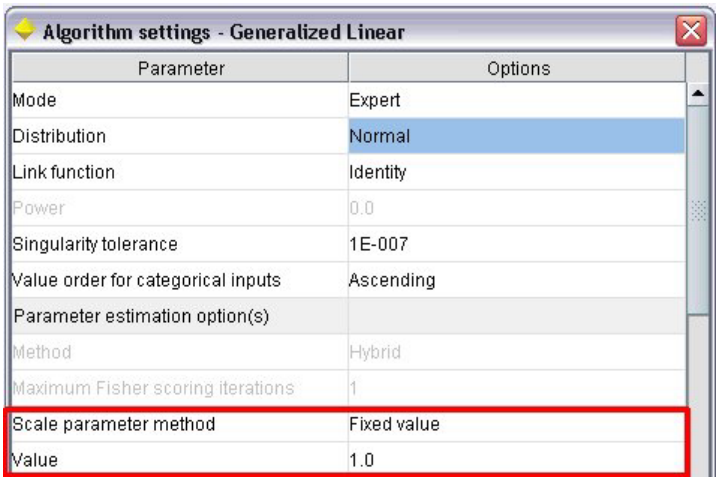

Figure 35. Effect of a constraint—parameter enabled

Applying Models

Applying a model means using a generated model to score data, that is, to use information gained from model building to create predictions for new records. In IBM SPSS Modeler, you achieve this by means of a model applier node. See the topic "Model Applier Nodes" on page 12 for more information.

Defining a model applier node in the specification file creates the framework for applying a generated model. In IBM SPSS Modeler, you create an instance of a model applier node by dragging the icon that represents the model output object from the Models tab of the manager pane onto the stream canvas. Without a model applier node definition, executing the model builder node would just generate an unrefined model, which cannot be added to the stream canvas.

When defining a model applier node, the `Node` element must include:

- A `type="modelApplier"` attribute
- A `Constructors` child element containing a `CreateModelOutput` element (see "Using Constructors" on page 95)

For the format of a `Node` element specification, see "Node" on page 45.

Building Documents

When defining a document builder node, the `Node` element must include:

- A `type="documentBuilder"` attribute
- A `DocumentBuilder` child element

For document building, the extension also needs a `DocumentOutput` element to describe the generated document. See the topic "Model Output" on page 83 for more information.

For the format of a `Node` element specification, see "Node" on page 45.

Document Builder

The `DocumentBuilder` element defines the behavior of a document builder node. The definition must include a `DocumentGeneration` child element to specify which tab on the document builder node dialog

box will contain the document generation controls. The controls themselves are defined in the User Interface section (see Chapter 6, "Building User Interfaces," on page 99) .

Format

```
<DocumentBuilder>
     <DocumentGeneration controlsId="control_identifier />
</DocumentBuilder>
```

where `controlsId` (required) is the identifier used by the system controls to specify where the document generation controls should appear.

Example

```
<DocumentBuilder>
     <DocumentGeneration controlsId="1"/>
</DocumentBuilder>
```

Document Output

The `DocumentOutput` element describes a document output object--an object that will appear under the Outputs tab in the manager pane following execution of a stream.

Format

```
<DocumentOutput id="identifier" label="display_label" labelKey="label_key" >
     <Properties>
          <Property ... />
           ...
     </Properties>
     <Containers>
          <Container ... />
           ...
     </Containers>
     <UserInterface ... />
     <Constructors ... />
</DocumentOutput>
```

where:

- `id` (required) is a unique identifier for the generated document.
- `label` (required) is the display name for the generated document as it is to appear on the Outputs tab.
- `labelKey` identifies the label for localization purposes.

The child elements that can be contained within the `DocumentOutput` element are shown in the following table.

Table 32. Child elements of document output declaration.

Child element	Defines	See...
`Properties`	Properties to be used by the generated document.	"Properties" on page 48
`Containers`	Containers in which generated document output will be placed.	"Containers" on page 50
`UserInterface`	User interface through which generated document output can be viewed.	"User Interface" on page 51
`Constructors`	Objects produced by the generated document.	"Using Constructors" on page 95

Example

```
<DocumentOutput id="webstatusreport">
     <Containers>
          <Container name="webstatusreportdata" />
     </Containers>
     <UserInterface>
          <Tabs>
               <Tab label="Advanced" labelKey="advancedTab.LABEL" >
                    <TextBrowserPanel container="webstatusreportdata" textFormat="html" />
               </Tab>
          </Tabs>
     </UserInterface>
</DocumentOutput>
```

Using Constructors

A Constructors element can appear in either of two places in the specification file:

- In the Execution section of a node definition (for model or document output objects)
- In a Model Output definition (for a model applier node)

Only one output object can be generated by a node. This limitation is imposed for consistency with existing nodes and the scripting and client API interfaces to these types of nodes.

Format

The format of the Constructors element when used in an Execution section is:

```
<Constructors>
     <CreateModelOutput ... />
      ...
     <CreateDocumentOutput ... />
      ...
     <CreateInteractiveModelBuilder ... />
      ...
</Constructors>
```

In a Model Output definition the format is:

```
<Constructors>
     <CreateModelApplier ... />
</Constructors>
```

Create Model Output

This section defines how a model output object is created on the Models tab or how a document output object is created on the Outputs tab.

Format

```
<CreateModelOutput type="output_object_id">
     <Condition ... ./>
     <And>
     <Or>
     <Not>
     <CreateModel type="model_id" target="container_id" sourceFile="container_file_id" />
      ...
     <CreateDocument type="document_id" target="container_id" sourceFile="container_file_id" />
      ...
</CreateModelOutput>
```

In the CreateModelOutput element:

- type (required) is the identifier of a model output object that is defined in a Model Output section. See the topic "Model Output" on page 47 for more information.

The Condition section enables you to specify one or more conditions. See the topic "Conditions" on page 67 for more information.

You can also specify complex conditions involving the And, Or, and Not operators. See the topic "Logical Operators" on page 67 for more information.

In the CreateModel and CreateDocument elements:

- type is the identifier of the model or document being defined.
- target (required) is the identifier of the container for the model; this container is defined in a Model Output section. See the topic "Model Output" on page 83 for more information.
- sourceFile (required) is the identifier of an output file that is generated during node execution; this file is defined in an Output Files section. See the topic "Output Files" on page 53 for more information.

Example

```
<Constructors>
     <CreateModelOutput type="naivebayes">
          <CreateModel type="naivebayes_model" target="model" sourceFile="pmml"/>
          <CreateDocument type="html_output" target="advanced_output" sourceFile="htmloutput" />
          <CreateDocument type="zip_outputType" target="zip_output" sourceFile="zipoutput" />
     </CreateModelOutput>
</Constructors>
```

Create Document Output

This element is used in the Execution section of a document builder node definition and identifies the document output object that is being created.

Format

```
<CreateDocumentOutput type="output_object_id">
     <Condition ... ./>
     <And>
     <Or>
     <Not>
     <CreateModel type="model_id" target="container_id" sourceFile="container_file_id" />
      ...
     <CreateDocument type="document_id" target="container_id" sourceFile="container_file_id" />
      ...
</CreateModelOutput>
```

where type (required) is the identifier of a document output object that is defined in a Document Output section. See the topic "Document Output" on page 47 for more information.

The Condition section enables you to specify one or more conditions. See the topic "Conditions" on page 67 for more information.

You can also specify complex conditions involving the And, Or, and Not operators. See the topic "Logical Operators" on page 67 for more information.

In the CreateModel and CreateDocument elements:

- type is the identifier of the model or document being defined.
- target (required) is the identifier of the container for the model; this container is defined in a Model Output section. See the topic "Model Output" on page 83 for more information.

- `sourceFile` (required) is the identifier of an output file that is generated during node execution; this file is defined in an Output Files section. See the topic "Output Files" on page 53 for more information.

Example

```
<Constructors>
     <CreateDocumentOutput type="webstatusreport">
          <CreateDocument type="webstatusreport" target="webstatusreportdata"
               sourceFile="webstatusreport_output_file" />
     </CreateDocumentOutput>
</Constructors>
```

Create Interactive Model Builder

This element is used in the Execution section of an interactive model builder node definition and identifies the output object with which the user will interact. See the topic "Create Interactive Model Builder" on page 85 for more information.

Create Model Applier

This element is used within a `Constructors` element in the Model Output section of a model builder node definition (see "Model Output" on page 83). The `CreateModelApplier` element defines how a model applier node is created when a model output object generated by the model builder node is dropped onto the canvas.

Format

```
<CreateModelApplier type="apply_node_identifier">
     <Condition ... ./>
     <And>
     <Or>
     <Not>
     <CreateModel type="model_id" target="container_id" sourceFile="container_file_id" />
      ...
     <CreateDocument type="document_id" target="container_id" sourceFile="container_file_id" />
      ...
</CreateModelApplier>
```

In the `CreateModelApplier` element:

- `type` (required) is the identifier of the model applier node to be created; this node is actually defined in a `Node ... type="modelApplier"` element later in the file.

The `Condition` section enables you to specify one or more conditions. See the topic "Conditions" on page 67 for more information.

You can also specify complex conditions involving the `And`, `Or` and `Not` operators. See the topic "Logical Operators" on page 67 for more information.

In the `CreateModel` and `CreateDocument` elements:

- `type` is the identifier of the model or document being defined.
- `target` (required) is the identifier of the container for the model; this container is defined in a Model Output section. See the topic "Model Output" on page 83 for more information.
- `sourceFile` (required) is the identifier of an output file that is generated during node execution; this file is defined in an Output Files section. See the topic "Output Files" on page 53 for more information.

Example

In the following example, the `CreateModelApplier` element contains a forward reference to the model applier node named `myapplier`. This node is defined in the subsequent `Node` element.

```
<ModelOutput>
     <Constructors>
          <CreateModelApplier type="myapplier"></CreateModelApplier>
     </Constructors>
</ModelOutput>
<Node id="myapplier" type="modelApplier">
 ...
</Node>
```

Chapter 6. Building User Interfaces

About User Interfaces

When adding a new CLEF node, you need to define how the end user will interact with the node and any model output, document output, or apply nodes that the extension specifies. The user interface for each object is defined in a `UserInterface` section of the specification file for the extension. This chapter provides a detailed description of the `UserInterface` section.

Note: There can be more than one `UserInterface` section in a single specification file, depending on what types of objects are defined in the file.

The user interface to a CLEF object consists of one or more of the following:

- Menu entry
- Palette entry
- Icons
- One or more dialog windows
- One or more output windows

Menu entries and **palette entries** provide access to the object from the IBM SPSS Modeler menu system and node palettes, respectively. **Icons** identify objects on the menus, on the drawing canvas, and in the various node palettes. **Dialog windows** contain tabs and controls to enable users to specify various options before a stream is executed and, optionally, to specify output to be produced when execution completes. **Output windows** are used to display model output (such as the results of applying a model to a set of data) or document output (such as a graph).

CLEF enables you to add new types of objects to the standard ones provided by IBM SPSS Modeler and supports a consistent approach to defining the user interface for these new objects. "Designing Node Icons" on page 14 and "Designing Dialog Boxes" on page 18 provide guidelines for creating icons and dialogs in CLEF.

Icons take the form of graphics that identify a particular node and appear inside the geometric shapes that identify the node type.

Dialogs and **output windows** have the following characteristics:

- Title bar containing miniature icon and object title
- Menu bar that can contain:
  - Menus
  - Object-specific action buttons
  - Common actions buttons (for example, Maximize or Help)
- Main content area
- Multiple tabs to organize the UI components into logical groups
- Resizing capability

A tab changes the main content area of a window in different ways. For a dialog window, different tabs display different sets of controls for the object properties. The controls can be modified, and the results are applied to the underlying object when the user clicks the **Apply** or **OK** button.

For an output window, tabs enable the user to perform different actions related to the output, such as displaying a summary of the results or adding annotations to them.

User Interface Section

The user interface for an object is declared in a `UserInterface` element within the object definition in the specification file. There can be more than one `UserInterface` element in the same specification file, depending on how many objects (for example, model builder node, model output object, model applier node) are defined in the file.

Within each User Interface section, you can define:

- Icons for display on the canvas or palettes
- Controls (custom menu and toolbar items) to appear on dialogs or output windows
- Tabs that define sets of property controls (for dialogs or output windows)

Note: In the element definitions that follow (normally identified by the heading **Format**), element attributes and child elements are optional unless indicated as "(required)." For the complete syntax of all elements, see "CLEF XML Schema," on page 195.

For each user interface, you specify the processing to be performed. You can do this by means of either an action handler or a frame class attribute, both of which are optional. Where neither an action handler nor a frame class attribute is specified, the processing to be performed is specified elsewhere in the file.

Format

The basic format of the `UserInterface` element is:

```
<UserInterface>
     <Icons>
          <Icon ... />
           ...
     </Icons>
     <Controls>
          <Menu ... />
          <MenuItem ... />
           ...
          <ToolbarItem ... />
           ...
     </Controls>
     <Tabs>
          <Tab ... />
           ...
     </Tabs>
</UserInterface>
```

An **action handler** is used where you are adding custom actions to a standard IBM SPSS Modeler window. The action handler specifies the Java class that is called when a user chooses a custom menu option or toolbar button on a node dialog, a model output window, or a document output window. It is an implementation of either an `ExtensionObjectFrame` class or an `ActionHandler` class. In either case, the standard window components are included automatically, such as the standard menus, tabs, and toolbar buttons. See the topic "Client-Side API Classes" on page 167 for more information.

The format for an action handler is the same as the basic format except for the first line:

```
<UserInterface actionHandler="Java_class" >
      ...
</UserInterface>
```

where `actionHandler` is the name of the action handler Java class.

A **frame class** is used for model output or document output objects where the extension is providing its own window rather than customizing a standard IBM SPSS Modeler window. A frame class is a Java class that completely specifies the entire window and its processing. No standard window components are included automatically--the class must specify these individually. Frame classes can be used only for model output or document output objects and not for nodes, which always use IBM SPSS Modeler dialogs. See the topic "Custom Output Windows" on page 152 for more information.

The format for a frame class is simply:

```
<UserInterface frameClass="Java_class" />
```

where `frameClass` is the name of the frame class for a model output or document output object. Any icons, controls, and tabs are specified by the frame class itself, so those elements are not used in this format.

The child elements of a `UserInterface` element are described in the subsequent sections.

Examples

The first example shows the user interface for a model builder node:

```
<UserInterface actionHandler="com.spss.myextension.MyActionHandler">
     <Icons>
          <Icon type="standardNode" imagePath="images/lg_discriminant.gif" />
          <Icon type="smallNode" imagePath="images/sm_discriminant.gif" />
     </Icons>
     <Tabs defaultTab="1">
      ...
     </Tabs>
</UserInterface>
```

The corresponding section for a model output object is:

```
<UserInterface>
     <Icons>
          <Icon type="standardWindow" imagePath="images/browser_discriminant.gif" />
     </Icons>
     <Tabs>
          <Tab label="Advanced" labelKey="advancedTab.LABEL"
           helpLink="discriminant_output_advancedtab.htm">
               <ExtensionObjectPanel id="DiscriminantPanel"
                panelClass="com.spss.clef.discriminant.DiscriminantPanel"/>
          </Tab>
     </Tabs>
</UserInterface>
```

The user interface section for a model applier node looks like this:

```
<UserInterface>
     <Icons>
          <Icon type="standardNode" imagePath="images/lg_gm_discriminant.gif" />
          <Icon type="smallNode" imagePath="images/sm_gm_discriminant.gif" />
     </Icons>
     <Tabs>
          <Tab label="Advanced" labelKey="advancedTab.LABEL"
           helpLink="discriminant_output_advancedtab.htm">
               <ExtensionObjectPanel id="DiscriminantPanel"
                panelClass="com.spss.clef.discriminant.DiscriminantPanel"/>
          </Tab>
     </Tabs>
</UserInterface>
```

Icons

This section defines the icons associated with the object.

For nodes, this section should define two `Icon` elements:

- A large version for display on the canvas
- A small version for display on the palette

For model outputs and document outputs, this defines the miniature icon that appears in the top left corner of the window frame.

"Designing Node Icons" on page 14 provides guidelines for creating icons in CLEF.

Format

```
<Icons>
     <Icon type="icon_type" imagePath="image_path" />
      ...
</Icons>
```

where:

`type` (required) is one of the icon types shown in the following table.

Table 33. Icon types.

Type	Example	Description
`standardNode`	 *Figure 36. Standard node icon*	The large size (49 x 49 pixels) icon displayed in the node shape on the canvas.
`smallNode`	 *Figure 37. Small node icon*	The smaller size (38 x 38 pixels) icon displayed in the node shape on the node palette.
`window`	 *Figure 38. Window icon*	The miniature (16 x 16 pixels) icon displayed in a browser or output window.

`imagePath` (required) identifies the location of the image used in this icon. The location is specified relative to the directory in which the specification file is installed.

Examples

```
<Icon type="standardNode" imagePath="images/lg_mynode.gif" />
<Icon type="smallNode" imagePath="images/sm_mynode.gif" />
<Icon type="window" imagePath="images/mynode16.gif" />
```

Controls

This section defines custom menu and toolbar items used to invoke actions declared in the Common Objects section of the specification file. See the topic "Actions" on page 38 for more information.

Format

```
<Controls>
     <Menu ... />
      ...
     <MenuItem ... />
      ...
     <ToolbarItem ... />
      ...
</Controls>
```

The `Menu`, `MenuItem`, and `ToolbarItem` elements are described in subsequent sections.

Example

The following example adds an item to the Generate menu and to the toolbar of the dialog in whose specification it is included. Both items implement a previously defined action named `generateDerive` that generates a Derive node.

```
<Controls>
     <MenuItem action="generateDerive" systemMenu="generate"/>
     <ToolbarItem action="generateDerive" showLabel="false"/>
      ...
</Controls>
```

Menus

You can define a custom menu to add to one of the standard menus.

Format

```
<Menu id="name" label="display_label" labelKey="label_key" systemMenu="menu_name"
showLabel="true_false" showIcon="true_false" separatorBefore="true_false" separatorAfter="true_
false" offset="integer" mnemonic="mnemonic_char" mnemonicKey="mnemonic_key"/>
```

where:

`id` (required) is a unique identifier for the menu that you are adding.

`label` (required) is the display name for the menu as it appears on the user interface (provided that `showLabel` is set to `true`).

`labelKey` identifies the label for localization purposes.

`systemMenu` (required) identifies the standard menu to which the custom menu is to be added. The value of *menu_name* is one of the following:

- `file`
- `edit`
- `insert*`
- `view*`
- `tools*`
- `window*`
- `generate`

- help*
- file.project
- file.outputs
- file.models
- edit.stream
- edit.node
- edit.outputs
- edit.models
- edit.project
- tools.repository
- tools.options
- tools.streamProperties

* only valid if adding to the main IBM SPSS Modeler window

showLabel specifies whether the display label of the item is to be shown on the user interface.

showIcon specifies whether the icon associated with the item is to be shown on the user interface.

separatorBefore specifies whether a separator (for example, a horizontal bar for menu items or a space for toolbar buttons) is to be added to the menu before this new item.

separatorAfter specifies whether a separator is to be added to the menu after this new item.

offset is a non-negative integer that defines the position of the new item; for example, an offset of 0 adds it as the first item (the default is to add it at the end).

mnemonic is the alphabetic character used in conjunction with the Alt key to activate this control (for example, if you give the value S, the user can activate this control by means of Alt-S).

mnemonicKey identifies the mnemonic for localization purposes. If neither mnemonic nor mnemonicKey is used, no mnemonic is available for this control. See the topic "Access Keys and Keyboard Shortcuts" on page 108 for more information.

Menu Items

You can define a custom menu item to add to one of the standard or custom menus.

Format

```
<MenuItem action="identifier" systemMenu="menu_name" customMenu="menu_name"
     showLabel="true_false" showIcon="true_false" separatorBefore="true_false"
     separatorAfter="true_false" offset="integer" />
```

where:

action (required) is the identifier of an action defined in the Common Objects section and specifies the action to be performed by this menu item.

systemMenu specifies that the item is to appear on the standard menu identified by *menu_name*, which is one of the following:

- file
- edit
- insert*

- view*
- tools*
- window*
- generate
- help*
- file.project
- file.outputs
- file.models
- edit.stream
- edit.node
- edit.outputs
- edit.models
- edit.project
- tools.repository
- tools.options
- tools.streamProperties

* only valid if adding to the main IBM SPSS Modeler window

customMenu is an identifier from a Menu element` and specifies that the menu item is to appear on this custom menu.

showLabel specifies whether the display label of the item is to be shown on the user interface.

showIcon specifies whether the icon associated with the item is to be shown on the user interface.

separatorBefore specifies whether a separator (for example, a horizontal bar for menu items or a space for toolbar buttons) is to be added to the menu before this new item.

separatorAfter specifies whether a separator is to be added to the menu after this new item.

offset is a non-negative integer that defines the position of the new item; for example, an offset of 0 adds it as the first item (the default is to add it at the end).

Example

```
<MenuItem action="generateSelect" systemMenu="generate" showIcon="true"/>
```

Toolbar Items

You can define a custom toolbar item for a dialog or output window.

Format

```
<ToolbarItem action="action" showLabel="true_false" showIcon="true_false"
     separatorBefore="true_false" separatorAfter="true_false" offset="integer" />
```

where:

action (required) is the identifier of an action defined in the Common Objects section and specifies the action to be performed by this toolbar item.

showLabel specifies whether the display label of the item is to be shown on the user interface.

showIcon specifies whether the icon associated with the item is to be shown on the user interface.

separatorBefore specifies whether a separator (for example, a horizontal bar for menu items or a space for toolbar buttons) is to be added to the menu before this new item.

separatorAfter specifies whether a separator is to be added to the menu after this new item.

offset is a non-negative integer that defines the position of the new item; for example, an offset of 0 adds it as the first item (the default is to add it at the end).

Example

```
<ToolbarItem action="generateDerive" showLabel="true"/>
```

Example: Adding to the Main Window

This is an example of a specification file that adds a new item to the Tools menu in the main window. It does not define any standard objects (for example, node, Model Output window, or Document Output window) but has a UserInterface element in the top-level Extension which means "change the main window." All other UserInterface sections would have to appear in one of the standard object definitions and would affect dialogs associated with those objects.

```
<?xml version="1.0" encoding="UTF-8"?>
<Extension version="1.0">
     <ExtensionDetails providerTag="example" id="main_window" label="Main Window" version="1.0"
          provider="IBM Corp." copyright="(c) 2005-2006 IBM Corp."
          description="An example extension that plugs into the main window"/>
     <Resources/>
     <CommonObjects>
          <Actions>
               <Action id="customTool1" label="Custom Tool..." labelKey="customTool.LABEL"
                    imagePath="images/generate.gif" description="Invokes the custom tool"
                    descriptionKey="customTool.TOOLTIP"/>
               <Action id="customTool2" label="Custom Tool..." labelKey="customTool.LABEL"
                    imagePath="images/generate.gif" description="Invokes the custom tool"
                    descriptionKey="customTool.TOOLTIP"/>
          </Actions>
     </CommonObjects>
     <UserInterface actionHandler="com.spss.cleftest.MainWindowActionHandler">
          <Controls>
               <Menu systemMenu="tools" id ="toolsExtension" separatorBefore="true"
                    label="Extension Items" offset="3"/>
               <MenuItem action="customTool2" customMenu="toolsExtension" showIcon="true"/>
               <MenuItem action="customTool1" systemMenu="file.models" showIcon="true"/>
               <ToolbarItem action="customTool1" offset="0"/>
          </Controls>
     </UserInterface>
</Extension>
```

The effect of this is to add a new sub-menu named **Extension Items** to the Tools menu. This new sub-menu has a single entry named **Custom Tool**.

You can try out this example by saving the XML code in a file named *extension.xml*, and then adding the extension to CLEF. See the topic "Testing a CLEF Extension" on page 191 for more information.

Tabs

The Tabs section defines the tabs that can appear on:

- The dialog displayed when the user opens a node on the canvas
- A model output window
- A document output window

Each Tabs section can contain multiple Tab elements, each of which declares a custom tab to be displayed:

```
<Tabs defaultTab="integer">
     <Tab ... />
     <Tab ... />
      ...
</Tabs>
```

where defaultTab is a non-negative integer that specifies which tab is to be displayed when the node dialog or window is opened for the first time in a stream. If the user selects a different tab, on closing and reopening the dialog or window while the stream is active, the most recently viewed tab is displayed instead of the default. Tab numbering starts at 0.

Note that other tabs may be included on the dialog or window automatically--for example, all dialogs and output windows include an **Annotations** tab, and all data source node dialogs include **Filter** and **Types** tabs.

A Tab element must declare the tab label (the text that appears on the tab itself) and should also include a label key to be used as a lookup if the tab label is to be translated.

Within each Tab element is a panel specification, which defines how the main content area of the tab is laid out. The panel specification can be one of three types: text browser, extension object, or properties. See the topic "Panel Specifications" on page 109 for more information.

The format of an individual Tab element is:

```
<Tab id="identifier" label="display_label" labelKey="label_key" helpLink="help_ID"
          mnemonic="mnemonic_char" mnemonicKey="mnemonic_key" >
     <TextBrowserPanel ... />
     <ExtensionObjectPanel ... />
     <PropertiesPanel ... />
     <ModelViewerPanel ... />
</Tab>
```

where:

id is a unique identifier that can be used to reference the tab in Java code.

label (required) is the display name for the tab as it is to appear on the user interface.

labelKey identifies the label for localization purposes.

helpLink is the identifier of a help topic to be displayed when the user invokes the help system, if any. The format of the identifier depends on the type of help system (see "Defining the Help System Location and Type" on page 155):

HTML help: URL of the help topic

JavaHelp: topic ID

mnemonic is the alphabetic character used in conjunction with the Alt key to activate this control (for example, if you give the value S, the user can activate this control by means of Alt-S).

mnemonicKey identifies the mnemonic for localization purposes. If neither mnemonic nor mnemonicKey is used, no mnemonic is available for this control. See the topic "Access Keys and Keyboard Shortcuts" for more information.

Examples

For examples of Tab elements, see the sections on the different types of panel specification that they can contain: "Text Browser Panel" on page 110, "Extension Object Panel" on page 111, "Properties Panel" on page 112, and "Model Viewer Panel" on page 114.

Access Keys and Keyboard Shortcuts

As an alternative to mouse access to features on the user interface, you can specify various combinations of keys to enable users to access features by means of the keyboard.

Access keys

Access keys are keys that can be used in conjunction with the Alt key. For menu items, dialog tabs, and various other dialog controls, you can specify access keys through the use of the mnemonic attribute of the following elements.

Table 34. Features on the user interface.

Feature	Element	See...
Screen action (e.g. for a menu item)	action	"Actions" on page 38
Menu	menu	"Menus" on page 103
Dialog tab	tab	"Tabs" on page 107
Controllers	Various	"Controllers" on page 121

For example, consider the following menu.

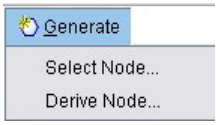

Figure 39. Menu items

To specify access keys for this menu you would use the following code:

```
<Actions>
     <Action id="generateSelect" label="Select Node..." labelKey="generate.selectNode.LABEL"
          imagePath="images/generate.gif" description="Generates a select node"
          descriptionKey="generate.selectNode.TOOLTIP" mnemonic="S" />
     <Action id="generateDerive" label="Derive Node..." labelKey="generate.deriveNode.LABEL"
          imagePath="images/generate.gif" description="Generates a derive node"
          descriptionKey="generate.deriveNode.TOOLTIP"  mnemonic="D" />
   ...
</Actions>
```

This gives you underscore characters on the menu items.

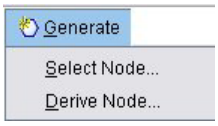

Figure 40. Using access keys with menu items

Users can now access the menu items by means of Alt-S and Alt-D, as well as clicking them with the mouse.

Shortcut keys

Shortcut keys are key combinations that enable end users to navigate around the user interface. IBM SPSS Modeler provides a number of keyboard shortcuts as standard. In CLEF, you can add shortcuts only for custom menu items that you have added.

For example, to specify shortcuts for items on the menu shown in the access keys example, you would use the following code:

```
<Actions>
     <Action id="generateSelect" label="Select Node..." labelKey="generate.selectNode.LABEL"
          imagePath="images/generate.gif" description="Generates a select node"
          descriptionKey="generate.selectNode.TOOLTIP" mnemonic="S" shortcut="CTRL+SHIFT+S" />
     <Action id="generateDerive" label="Derive Node..." labelKey="generate.deriveNode.LABEL"
          imagePath="images/generate.gif" description="Generates a derive node"
          descriptionKey="generate.deriveNode.TOOLTIP"  mnemonic="D"  shortcut="CTRL+SHIFT+D" />
     ...
</Actions>
```

The shortcut key combinations are now added to the menu items.

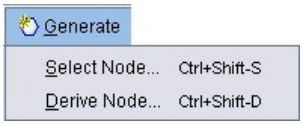

Figure 41. Using shortcut keys with menu items

Users can now access the menu items by means of the keyboard shortcuts, as well as clicking them with the mouse. You can specify shortcut keys in conjunction with access keys, as in this example.

Take care not to use shortcuts that have already been specified on the same dialog, or any of the standard IBM SPSS Modeler shortcuts.

Panel Specifications

Each `Tab` element can contain the specification of a single panel, which can be one of the types shown in the following table.

Table 35. Panel specification types

Panel	Displays...	See...
Text browser panel	Text content of a specified container.	"Text Browser Panel" on page 110
Extension object panel	Content defined by a specified Java class.	"Extension Object Panel" on page 111
Properties panel	Property controls (for example, buttons, check boxes, input fields).	"Properties Panel" on page 112

Table 35. Panel specification types (continued)

Panel	Displays...	See...
Model viewer panel	PMML-format model output from a specified container.	"Model Viewer Panel" on page 114

Text Browser Panel

A text browser panel displays the text content of a specified container in the extension. Supported formats (UTF-8 encoding) are plain text, HTML, and Rich Text Format (RTF).

The following is an example of a model output window containing a text browser panel in HTML format.

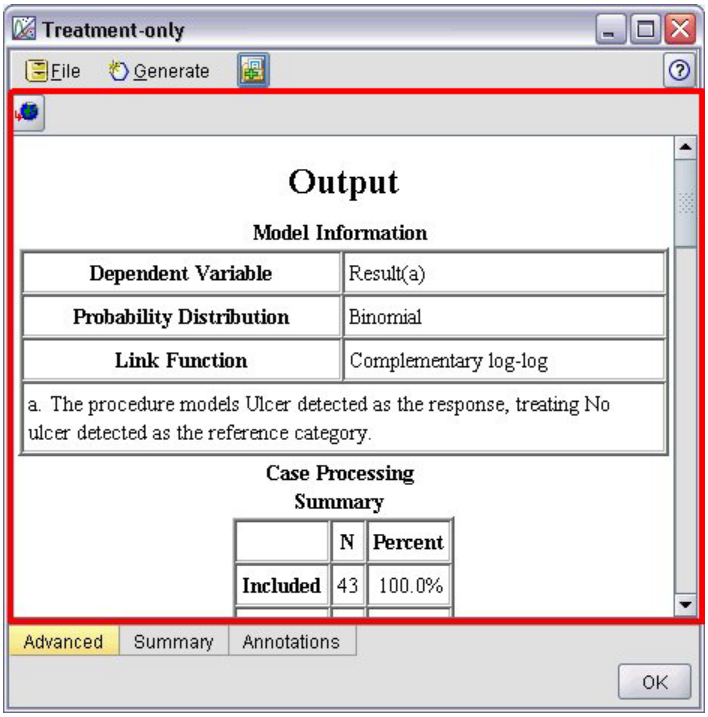

Figure 42. Model output window with text browser panel highlighted

Format

```
<TextBrowserPanel container="name" textFormat="text_format" rows="integer"
     columns="integer" wrapLines="true_false" >
     -- advanced custom layout options --
</TextBrowserPanel>
```

where:

`container` (required) is the name of the container from which to obtain the contents of the panel.

`textFormat` (required) specifies the format of the text displayed in the panel and is one of:

- `plainText`
- `html`
- `rtf`

rows and columns specify the number of text rows and columns that are visible when the window containing the panel is opened.

wrapLines specifies whether to use line wrap for long text lines (true) or to require horizontal scrolling to read long text lines (false). Default is false.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

The following example illustrates the Tab section that defines the text browser panel shown earlier:

```
<Tab label="Advanced" labelKey="advancedTab.LABEL" helpLink="genlin_output_advancedtab.htm">
     <TextBrowserPanel container="advanced_output" textFormat="html" />
</Tab>
```

The model output is sent to a container that is defined in the same ModelOutput section as the tab specification:

```
<ModelOutput id="generalizedlinear" label="Generalized Linear">
     <Containers>
      ...
          <Container name="advanced_output"/>
     </Containers>
     <UserInterface>
      ...
          <Tabs>
               <Tab label="Advanced" ... >
                    <TextBrowserPanel container="advanced_output" ... />
               </Tab>
          </Tabs>
     </UserInterface>
</ModelOutput>
```

The container is referenced in a CreateDocument element in the Execution section for the relevant build node:

```
<Execution>
 ...
     <Constructors>
          <CreateModelOutput type="generalizedlinear">
               <CreateModel type="generalizedlinear_model" target="model" sourceFile="pmml"/>
               <CreateDocument type="html_output" target="advanced_output" sourceFile=
                "htmloutput"/>
          </CreateModelOutput>
     </Constructors>
</Execution>
```

Extension Object Panel

An extension object panel works in a similar way to a text browser panel, but instead of displaying the text content of a container, it creates an instance of a specified Java class that implements the ExtensionObjectPanel interface defined by the CLEF Java API.

Here is an example of a model apply node dialog containing an extension object panel.

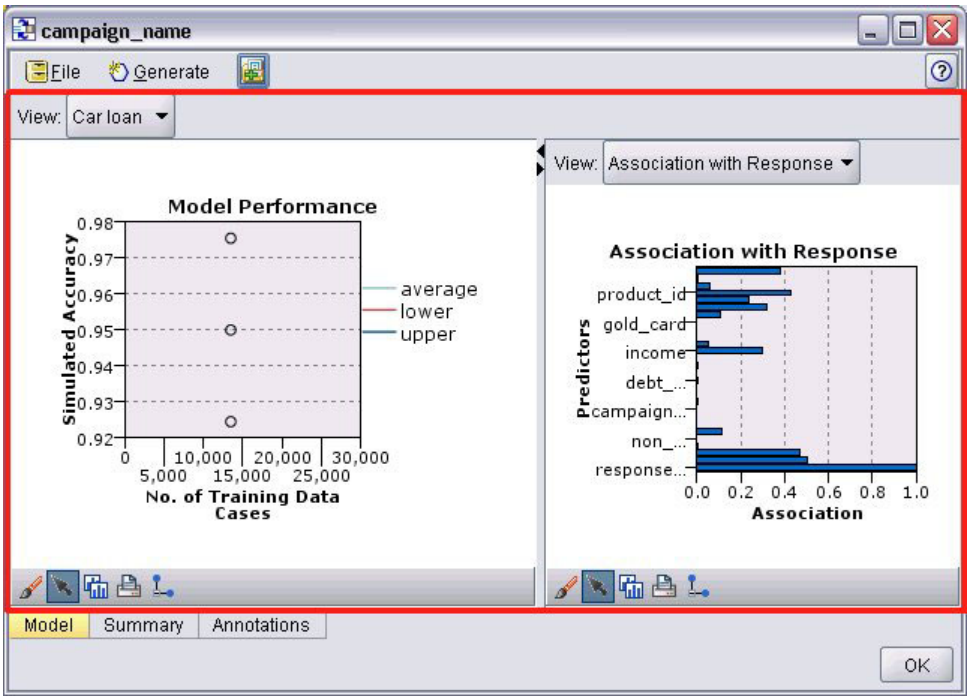

Figure 43. Model output window with extension object panel highlighted

Format

```
<ExtensionObjectPanel id="identifier" panelClass="Java_class" >
     -- advanced custom layout options --
</ExtensionObjectPanel>
```

where:

`id` is a unique identifier that can be used to reference the panel in Java code.

`panelClass` (required) is the name of the Java class that the panel implements.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

The following example illustrates the Tab section that defines the extension object panel shown earlier:

```
<Tab label="Model" labelKey="Model.LABEL" helpLink="selflearnnode_output.htm">
     <ExtensionObjectPanel id="SelfLearningPanel"
          panelClass="com.spss.clef.selflearning.SelfLearningPanel"/>
</Tab>
```

Properties Panel

A properties panel allows a tab or a properties sub-panel (see "Properties Sub-Panel" on page 119) to display **property controls**, which are screen components (such as buttons, check boxes, and input fields) that can be used to modify the properties of an object shown on the screen. The properties panel automatically applies the changes made with these controls when the user clicks **OK** or **Apply**. When the user clicks **Cancel** or **Reset**, the panel discards any changes made since the last apply operation.

The following is an example of a node dialog containing a properties panel.

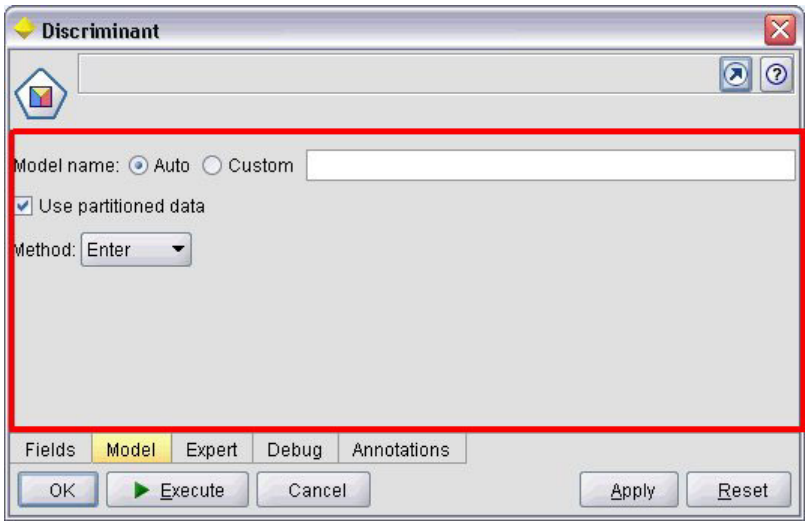

Figure 44. Node dialog with properties panel highlighted

The next example illustrates a properties sub-panel containing three properties panels.

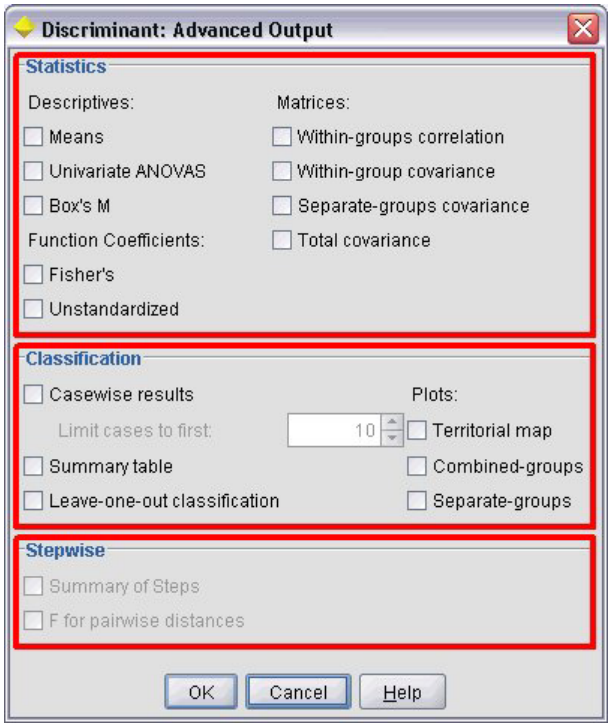

Figure 45. Properties sub-panel with properties panels highlighted

Format

```
<PropertiesPanel id="identifier" label="display_label" labelKey="label_key">
     -- advanced custom layout options --
     -- property control specifications --
</PropertiesPanel>
```

where:

`id` is a unique identifier that can be used to reference the panel in Java code.

`label` is the display heading for a group of property controls (for example, **Statistics**, **Classification** and **Stepwise** in the last example).

`labelKey` identifies the label for localization purposes.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

The individual property control specifications are described under "Property Control Specifications" on page 115.

Example

```
<Tab label="Model" labelKey="Model.LABEL" helpLink="discriminant_node_model.htm">
     <PropertiesPanel>
          <SystemControls controlsId="ModelGeneration" />
          <ComboBoxControl property="method">
               <Layout fill="none" />
          </ComboBoxControl>
     </PropertiesPanel>
</Tab>
```

Model Viewer Panel

A model viewer panel displays any PMML-format model output from a container specified in the extension.

Following is an example of a model applier node window containing a model viewer panel.

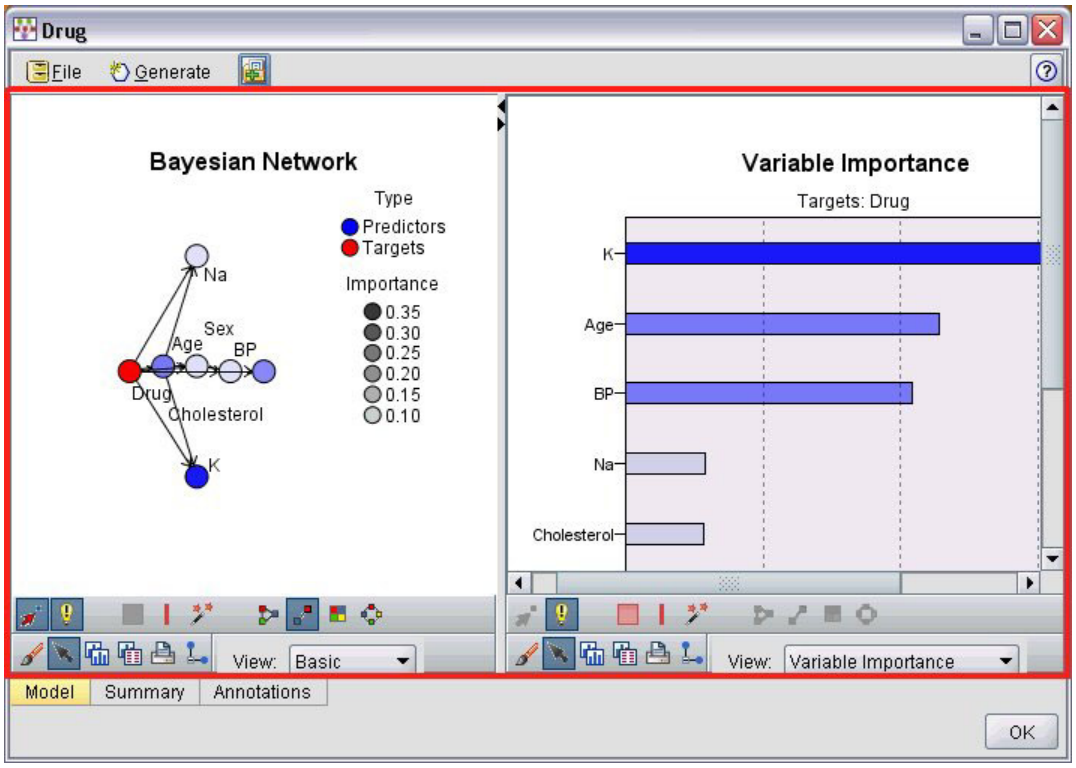

Figure 46. Output window with model viewer panel highlighted

Format

```
<ModelViewerPanel container="container_name">
     -- advanced custom layout options --
</ModelViewerPanel>
```

where `container` (required) is the name of the container to which the model output is assigned.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

This example illustrates the use of a model viewer panel in a model applier specification. The model output has previously been assigned to the container named `model`. Here the model applier specification picks up this container and associates it with the model viewer panel:

```
<Node id="applyBN" type="modelApplier">
     <ModelProvider container="model" isPMML="true" />
      ...
     <Containers>
          <Container name="model" />
     </Containers>
     <UserInterface>
      ...
     <Tabs>
          <Tab label="Model" labelKey="modelTab.LABEL" helpLink="BN_output_modeltab.htm">
               <ModelViewerPanel container="model"/>
          </Tab>
      ...
     </Tabs>
     </UserInterface>
      ...
</Node>
```

Property Control Specifications

Property controls are screen components such as buttons, check boxes, and input fields that can be used to modify the properties of an object shown on the screen. The format of a property control specification depends on the property control type, which can be one of the following:

- UI component
- Property panel
- Controller

UI component controls are action buttons, static text on the display, and system controls (a set of controls that handle properties common to all dialogs).

Property panel controls are individual panels within the properties panel specification.

Controllers form the largest group of property controls and include items such as check boxes, combo boxes, and spinner controls.

UI Component Controls

UI component controls are shown in the following table.

Table 36. UI component controls

Control	Description
`ActionButton`	A screen button that executes a predefined action when clicked.

Table 36. UI component controls (continued)

Control	Description
StaticText	A non-variable text string displayed on the screen.
SystemControls	Standard sets of controls that handle properties common to all models.

Action Button

Defines a dialog or toolbar button that executes an action specified in the Common Objects section. The action (for example, displaying a new screen) is performed when the user clicks this button.

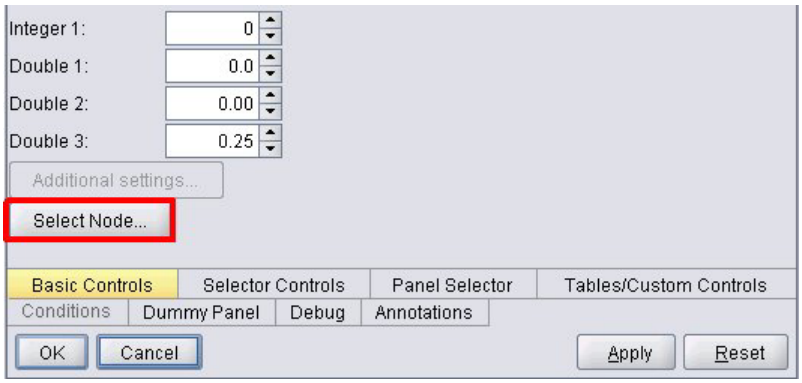

Figure 47. Dialog with action button highlighted

Format

```
<ActionButton action="action" showLabel="true_false" showIcon="true_false" >
     -- advanced custom layout options --
</ActionButton>
```

where:

`action` (required) is the identifier of the action to be performed.

`showLabel` specifies whether to show (`true`) or hide (`false`) the button label (for example, for a toolbar button, you might choose to show an icon and not a label). Default is `true`.

`showIcon` specifies whether to show (`true`) or hide (`false`) an icon associated with the button. Default is `false`.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

The code to create the action button shown earlier is:

```
<ActionButton action="generateSelect"/>
```

The action is defined in the Common Objects section as follows (note that the button label is defined here too):

```
<CommonObjects extensionListenerClass="com.spss.cleftest.TestExtensionListener">
    ...
   <Actions>
     <Action id="generateSelect" label="Select Node..." labelKey="generate.selectNode.LABEL"
          imagePath="images/generate.gif" description="Generates a select node"
```

```
            descriptionKey="generate.selectNode.TOOLTIP"/>
        ...
    </Actions>
</CommonObjects>
```

Static Text

This element enables you to include a non-variable text string on a dialog or output window. The following illustrates a properties panel that includes static text.

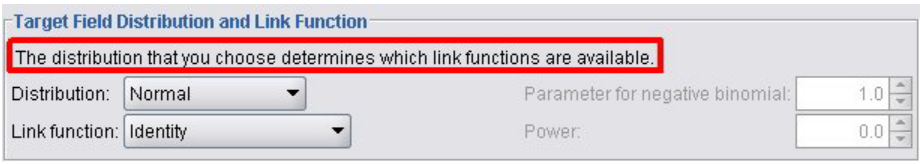

Figure 48. Properties panel with static text highlighted

Format

```
<StaticText text="static_text" textKey="text_key" >
     -- advanced custom layout options --
</StaticText>
```

where:

`text` is the text string you want to use.

`textKey` identifies the static text for localization purposes.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

The following example shows the declaration used for the static text shown earlier:

```
<StaticText text="The distribution that you choose determines which link functions are
     available." textKey="Genlin_staticText1"/>
```

System Controls

Some properties are common to all models. In a model builder node, the system controls are standard sets of controls that handle these properties.

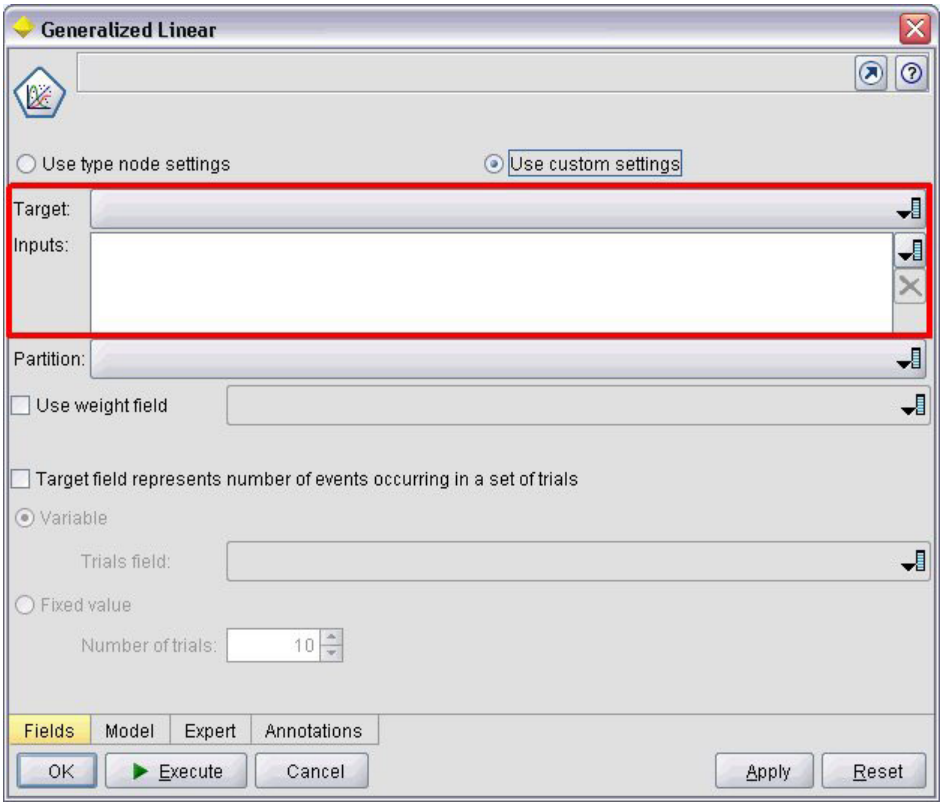

Figure 49. Example of dialog with system controls highlighted

Format

```
<SystemControls controlsID="identifier" >
     -- advanced custom layout options --
</SystemControls>
```

where `controlsID` is the identifier of the set of controls. This identifier must be the same as one specified in the `controlsID` attribute of a `ModelingFields` element in a Model Builder declaration (see "Model Builder" on page 48).

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

This example shows the declaration used for the system controls in the preceding illustration.

In the Model Builder section of the node specification, the following defines a set of system controls, in this case, comprising the field pickers for input and output fields for the model:

```
<ModelBuilder ... >
     <ModelingFields controlsId="modelingFields">
          <InputFields property="inputs" multiple="true" label="Inputs" types="[range set
orderedSet flag]" labelKey="inputFields.LABEL"/>
          <OutputFields property="target" multiple="false" types="[range flag]"
label="Target" labelKey="targetField.LABEL"/>
     </ModelingFields>
      ...
</ModelBuilder>
```

Subsequently in the file, this set of controls is referenced in the definition for the tab of the model builder node dialog on which they appear:

```
<Tab label="Fields" labelKey="Fields.LABEL" helpLink="genlin_node_fieldstab.htm">
     <PropertiesPanel>
          <SystemControls controlsId="modelingFields">
          </SystemControls>
      ...
     </PropertiesPanel>
</Tab>
```

Property Panel Controls

Property panel controls are shown in the following table.

Table 37. Property panel controls

Control	Description
`PropertiesSubPanel`	Separate dialog that is displayed when the user clicks a button on a properties panel.
`PropertiesPanel`	Properties panel nested in a properties sub-panel declaration or nested in a top-level properties panel declaration.

Properties Sub-Panel

Defines a separate dialog that is displayed when the user clicks a button on a properties panel. The properties sub-panel declaration is made as part of the main properties panel specification for a tab.

Format

```
<PropertiesSubPanel buttonLabel="display_label" buttonLabelKey="label_key"
     dialogTitle="display_title" dialogTitleKey="title_key" helplink="help_ID"
     mnemonic="mnemonic_char" mnemonicKey="mnemonic_key" >
     -- advanced custom layout options --
     -- property control specifications --
</PropertiesSubPanel>
```

where:

`buttonLabel` is the label of the button that provides access to the sub-panel.

`buttonLabelKey` identifies the button label for localization purposes.

`dialogTitle` is the text that is to appear on the title bar of the sub-panel dialog.

`dialogTitleKey` identifies the sub-panel dialog title for localization purposes.

`helpLink` is the identifier of a help topic to be displayed when the user invokes the help system, if any. The format of the identifier depends on the type of help system (see "Defining the Help System Location and Type" on page 155):

HTML help: URL of the help topic

JavaHelp: topic ID

`mnemonic` is the alphabetic character used in conjunction with the Alt key to activate this control (for example, if you give the value S, the user can activate this control by means of Alt-S).

mnemonicKey identifies the mnemonic for localization purposes. If neither mnemonic nor mnemonicKey is used, no mnemonic is available for this control. See the topic "Access Keys and Keyboard Shortcuts" on page 108 for more information.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

The individual property control specifications are described under "Property Control Specifications" on page 115.

Example

The following illustrates a properties sub-panel that is displayed when the user clicks the **Output** button on the main properties panel of a tab.

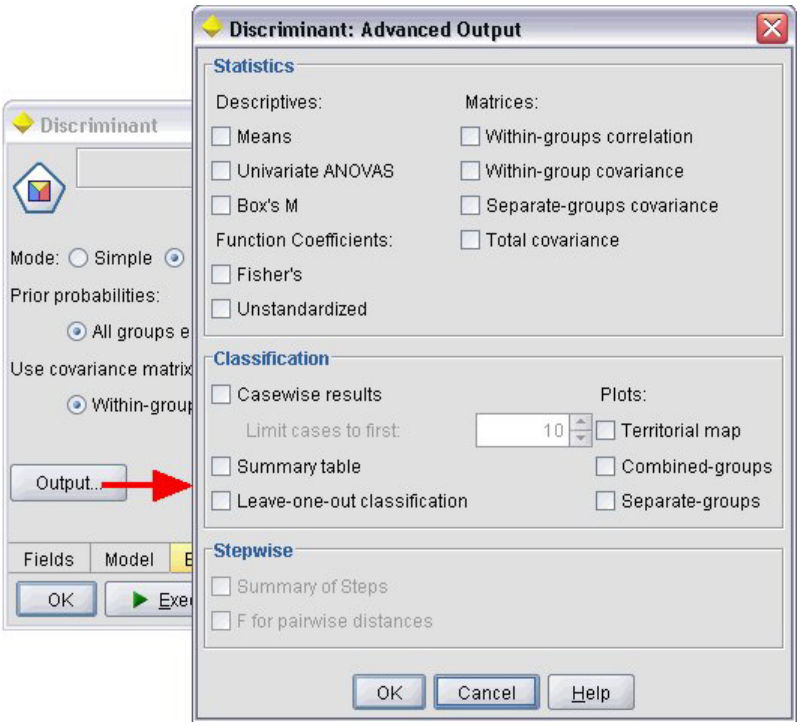

Figure 50. Properties sub-panel

The following code shows the most significant parts of the declaration used to obtain the properties sub-panel illustrated. Note how, within the sub-panel declaration, each of the field groups (**Statistics**, **Classification** and **Stepwise**) has its own properties panel specification:

```
<PropertiesSubPanel buttonLabel="Output..." buttonLabelKey="OutputSubPanel.LABEL"
     dialogTitle="Discriminant: Advanced Output" dialogTitleKey="AdvancedOutputSubDialog.LABEL"
     helpLink="discriminant_node_outputdlg.htm">
     ...
     <PropertiesPanel>
          <PropertiesPanel label="Statistics" ... >
           ...
          </PropertiesPanel>
          <PropertiesPanel label="Classification" ... >
           ...
          </PropertiesPanel>
          <PropertiesPanel label="Stepwise" ... >
```

```
          ...
        </PropertiesPanel>
    </PropertiesPanel>
</PropertiesSubPanel>
```

Properties Panel (nested)

You can nest a properties panel specification within a properties sub-panel declaration, to define the contents of the dialog displayed from the sub-panel. See the topic "Properties Sub-Panel" on page 119 for more information.

You can also nest a properties panel specification within a top-level properties panel declaration. One example of when you might want to do this would be where the contents of an entire tab, consisting of several properties panels, are enabled or disabled according to whether a particular button on the tab is selected. In this case, the tab specification would look similar to the following:

```
<Tab .... >
     <PropertiesPanel>
      --- button specification ---
          <PropertiesPanel>
               <Enabled>
                --- condition involving button value ---
               </Enabled>
           ...
          </PropertiesPanel>
          <PropertiesPanel>
               <Enabled>
                --- condition involving button value ---
               </Enabled>
           ...
          </PropertiesPanel>
           ...
     </PropertiesPanel>
</Tab>
```

The format of a nested properties panel specification is the same as that for the top-level element. See the topic "Properties Panel" on page 112 for more information.

Controllers

Controllers form the largest group of property controls.

Table 38. Controllers

Control	Description
`CheckBoxControl`	Check box.
`CheckBoxGroupControl`	Set of check boxes, one check box per `enum` value.
`ClientDirectoryChooserControl`	Single-line text field and associated button to allow user to select a directory on the client.
`ClientFileChooserControl`	Single-line text field and associated button to allow user to select a file on the client.
`ComboBoxControl`	Combo box drop-down containing the `enum` values.
`DBConnectionChooserControl`	Allows the user to select a data source and connect to a database.
`DBTableChooserControl`	Allows the user to select a database table following a successful database connection.
`MultiFieldChooserControl`	(Nodes only) List of field names allowing user to choose one or more fields from the list.
`MultiItemChooserControl`	Allows the user to choose one or more items from a list of values.

Table 38. Controllers (continued)

Control	Description
PasswordBoxControl	Single-line text field where input characters are hidden.
PropertyControl	A user-definable control for a property.
RadioButtonGroupControl	Set of radio buttons where only one button can be selected at a time. For enum properties, there is one radio button per enum value; for Boolean properties, two radio buttons are displayed.
ServerDirectoryChooserControl	Single-line text field and associated button to allow user to select a directory on the server.
ServerFileChooserControl	Single-line text field and associated button to allow user to select a file on the server.
SingleFieldChooserControl	(Nodes only) List of field names allowing user to choose a single field from the list.
SingleItemChooserControl	Allows the user to choose a single item from a list of values.
SpinnerControl	Spinner control (numeric field with up and down arrows to change the value).
TableControl	Adds a table to a dialog or window.
TextAreaControl	Multi-line text field.
TextBoxControl	Single-line text field.

Controller Attributes

Controller specifications can include the following attributes:

property="*value*" showLabel="*true_false*" label="*display_label*" labelKey="*label_key*" labelWidth="*label_width*" labelAbove="*true_false*" description="*description*" descriptionKey="*description_key*" mnemonic="*mnemonic_char*" mnemonicKey="*mnemonic_key*"

where:

property (required) is the unique identifier of the property control.

showLabel specifies whether to show (true) or hide (false) the display label of the property control. Default is true.

label is the display name for the property control as it is to appear on the user interface. This value also acts as the short accessible description of the property control. See the topic "Accessibility" on page 165 for more information.

labelKey identifies the label for localization purposes.

labelWidth is the number of display grid columns that the label spans. Default is 1.

labelAbove specifies whether the label for the control is to appear above (true) or adjacent to (false) the control. Default is false.

description is the text of the tooltip displayed when the cursor hovers over the control. This value also acts as the long accessible description of the property control. See the topic "Accessibility" on page 165 for more information.

descriptionKey identifies the description for localization purposes.

mnemonic is the alphabetic character used in conjunction with the Alt key to activate this control (for example, if you give the value S, the user can activate this control by means of Alt-S).

mnemonicKey identifies the mnemonic for localization purposes. If neither mnemonic nor mnemonicKey is used, no mnemonic is available for this control. See the topic "Access Keys and Keyboard Shortcuts" on page 108 for more information.

Check Box Control

Defines a check box.

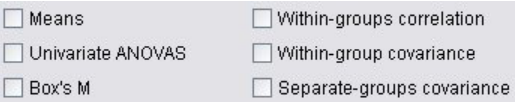

Figure 51. Check boxes

Format

```
<CheckBoxControl controller_attributes invert="true_false" >
     -- advanced custom layout options --
</CheckBoxControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

invert is rarely used but, if set to true, reverses the effect of the check box selection and deselection. Default is false.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

The following example shows the code used to lay out the check boxes shown earlier (the check box labels are defined elsewhere in the specification file). The Layout element is described under "Advanced Custom Layout" on page 143.

```
<CheckBoxControl property="means">
     <Layout rowIncrement="0" gridWidth="1" />
</CheckBoxControl>
<CheckBoxControl property="within_groups_correlation">
     <Layout gridColumn="2" />
</CheckBoxControl>
<CheckBoxControl property="univariate_anovas">
     <Layout gridWidth="1" rowIncrement="0" />
</CheckBoxControl>
<CheckBoxControl property="within_group_covariance">
     <Layout gridColumn="2" />
</CheckBoxControl>
<CheckBoxControl property="box_m">
     <Layout gridWidth="1" rowIncrement="0" />
</CheckBoxControl>
<CheckBoxControl property="separate_groups_covariance">
     <Layout gridColumn="2" />
</CheckBoxControl>
```

Check Box Group Control

Defines a set of check boxes grouped together and treated as a single unit. This can be used only in conjunction with an enumerated list property that defines the members of the group.

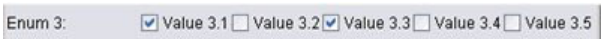

Figure 52. Check box group

Format

```
<CheckBoxGroupControl controller_attributes rows="integer" layoutByRow="true_false"
     useSubPanel="true_false" >
     -- advanced custom layout options --
</CheckBoxGroupControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`rows` is a positive integer specifying the number of rows on the screen that the check box group will occupy. Default is 1.

`layoutByRow` specifies whether the check boxes are to be laid out along the row first (`true`) or down the column first (`false`). Default is `true`. For a similar use of `layoutByRow` with a radio button group, see "Changing the Order of Controls" on page 143.

`useSubPanel` specifies whether (`true`) or not (`false`) the check boxes are to be laid out as a sub-panel. Default is `true`.

Check box groups are normally laid out as a sub-panel containing all the boxes in the group. However, this can result in alignment problems if the check box group is associated with an adjacent text field. Setting `useSubPanel` to `false` overcomes this problem.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

The code to create the check box group shown earlier is:

```
<CheckBoxGroupControl property="enum3" label="Enum 3" labelKey="enum3.LABEL"/>
```

The labels and values associated with the individual check boxes are defined in the Properties section for the relevant node:

```
<Property name="enum3" valueType="enum" isList="true" defaultValue="[value1 value3]">
     <Enumeration>
          <Enum value="value1" label="Value 3.1" labelKey="enum3.value1.LABEL"/>
          <Enum value="value2" label="Value 3.2" labelKey="enum3.value2.LABEL"/>
          <Enum value="value3" label="Value 3.3" labelKey="enum3.value3.LABEL"/>
          <Enum value="value4" label="Value 3.4" labelKey="enum3.value4.LABEL"/>
          <Enum value="value5" label="Value 3.5" labelKey="enum3.value5.LABEL"/>
     </Enumeration>
</Property>
```

Client Directory Chooser Control

Defines a single-line text field and associated button that allow the user to select a directory on the client. The directory must already exist. Users can either open a file from this directory or save a file to it,

depending on the mode setting.

Figure 53. Client directory chooser control

The user can either enter the directory path and name directly in the text field or click the adjacent button to display a dialog from where they can select a directory.

Format

```
<ClientDirectoryChooserControl controller_attributes mode="chooser_mode" >
     -- advanced custom layout options --
</ClientDirectoryChooserControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`mode` determines the button displayed on the dialog from where users choose a directory and is one of the following:

- `open` (default) displays an **Open** button.
- `save` displays a **Save** button.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

```
<ClientDirectoryChooserControl property="directory2" label="Client Directory"
     labelKey="directory2.LABEL"/>
```

Client File Chooser Control

Defines a single-line text field and associated button that allow the user to select a file on the client. The file must already exist. Users can either open the file or save it, depending on the mode setting.

Figure 54. Client file chooser control

The user can either enter the file path and name directly in the text field or click the adjacent button to display a dialog from where they can select a file.

Format

```
<ClientFileChooserControl controller_attributes mode="chooser_mode" >
     -- advanced custom layout options --
</ClientFileChooserControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`mode` determines the button displayed on the dialog from where users choose a file and is one of the following:

- `open` (default) displays an **Open** button.
- `save` displays a **Save** button.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

```
<ClientFileChooserControl property="file2" label="Client File" labelKey="file2.LABEL"/>
```

Combo Box Control

Defines a combo box drop-down list.

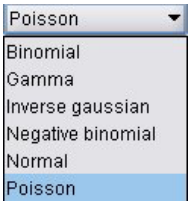

Figure 55. Combo box

Format

```
<ComboBoxControl controller_attributes >
     -- advanced custom layout options --
</ComboBoxControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

The following example shows the code used to lay out the combo box drop-down list in the preceding illustration:

```
<ComboBoxControl property="distribution" >
     <Layout rowIncrement="0" gridWidth="1" fill="none"/>
</ComboBoxControl>
```

The `Layout` element is described under "Advanced Custom Layout" on page 143.

Note: The actual list entries are defined in the Properties section for the relevant node; in this case, as an enumerated list in the declaration for the `distribution` property:

```
<Property name="distribution" valueType="enum" label="Distribution" labelKey="distribution.
LABEL" defaultValue="NORMAL">
     <Enumeration>
          <Enum value="BINOMIAL" label="Binomial" labelKey="distribution.BINOMIAL.LABEL"/>
          <Enum value="GAMMA" label="Gamma" labelKey="distribution.GAMMA.LABEL"/>
          <Enum value="IGAUSS" label="Inverse gaussian" labelKey="distribution.IGAUSS.LABEL"/>
          <Enum value="NEGBIN" label="Negative binomial" labelKey="distribution.NEGBIN.LABEL"/>
          <Enum value="NORMAL" label="Normal" labelKey="distribution.NORMAL.LABEL"/>
          <Enum value="POISSON" label="Poisson" labelKey="distribution.POISSON.LABEL"/>
     </Enumeration>
</Property>
```

Database Connection Chooser Control

Defines a control that allows the user to select a data source and connect to a database.

Figure 56. Database connection chooser control

Users cannot enter text into the text field--instead they must click the button to display the standard IBM SPSS Modeler Database Connections dialog.

On successful connection, the connection details are displayed in the text field of the database connection chooser control.

Format

```
<DBConnectionChooserControl controller_attributes >
     -- advanced custom layout options --
</DBConnectionChooserControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

The following example illustrates how the control requires the definition of a string property that it can use for the connection string.

```
<Node ... >
     <Properties>
      ...
          <Property name="dbconnect" valueType="databaseConnection" />
     </Properties>
      ...
     <UserInterface>
      ...
          <Tabs>
               <Tab label="Database">
                    <PropertiesPanel>
                         <DBConnectionChooserControl property="dbconnect" label="Database
                          Connection"/>
                     ...
                    </PropertiesPanel>
                     ...
     </UserInterface>
</Node>
```

Database Table Chooser Control

Defines a text field and associated button that allow the user to select a database table following a successful database connection.

Figure 57. Database table chooser control

Users can either enter the table name directly into the text field or click the button and select it from a list.

Format

```
<DBTableChooserControl connectionProperty="DB_connection_property" controller_attributes >
     -- advanced custom layout options --
</DBTableChooserControl>
```

where:

`connectionProperty` is the name of a database connection property that has already been defined. This is the value of the `property` attribute of a `DBConnectionChooserControl` element that has previously been defined for the node.

controller_attributes are as described under "Controller Attributes" on page 122.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

This example follows on from the one for `DBConnectionChooserControl` and shows how you can also include a `DBTableChooserControl` element to select a database table once a successful database connection has been established.

```
<Node ... >
     <Properties>
      ...
          <Property name="dbconnect" valueType="databaseConnection" />
          <Property name="dbtable" valueType="string" />
     </Properties>
       ...
     <UserInterface>
      ...
          <Tabs>
               <Tab label="Database">
                    <PropertiesPanel>
                         <DBConnectionChooserControl property="dbconnect" label="Database
                          connection"/>
                         <DBTableChooserControl property="dbtable" connectionProperty=
                          "dbconnect" label="Database Table" />
                     ...
                    </PropertiesPanel>
      ...
     </UserInterface>
</Node>
```

Multi-Field Chooser Control

Defines a control that enables the user to choose one or more field names from a list.

Figure 58. Multi-field chooser control

When the user clicks this control, a list of fields is displayed from which the user can choose one or more.

The set consists of all the fields that are visible at this node. If fields have been filtered further upstream from this node, only the fields that have passed through the filter are visible. The list can also be further restricted by specifying that only fields with particular storage and data types are to be available for selection.

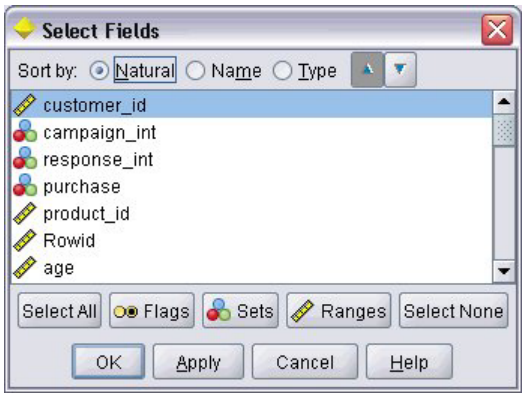

Figure 59. Multi-field list

Each multi-field chooser control specifies a property attribute, which is declared elsewhere in the file and defines how the list is displayed on the node dialog.

Format

```
<MultiFieldChooserControl controller_attributes storage="storage_types" onlyNumeric="true_false"
     onlySymbolic="true_false" onlyDatetime="true_false" types="data_types"
     onlyRanges="true_false" onlyDiscrete="true_false" >
     -- advanced custom layout options --
</MultiFieldChooserControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

In addition, you can further restrict the list of fields by specifying two other attributes, one of which must be from the following list:

- `storage` is a list property that specifies the storage type of fields to be allowed in the list--for example, `storage="[integer real]"` means that only fields with these storage types will be listed. For the set of possible storage types, see the table under "Data and Storage Types" on page 175.
- `onlyNumeric`, if set to `true`, specifies that only fields with a numeric storage type are listed.
- `onlySymbolic`, if set to `true`, specifies that only fields with a symbolic (that is, string) storage type are listed.
- `onlyDatetime`, if set to `true`, specifies that only fields with a date-and-time storage type are listed.

The second attribute specified must be from this list:

- `types` is a list property that specifies the data type of fields to be allowed in the list--for example, `types="[range flag]"` means that only fields with these storage types will be listed. The set of possible data types is:

  `range`

  `flag`

  `set`

  `orderedSet`

  `numeric`

discrete

typeless

- onlyRanges, if set to true, specifies that only fields with a range data type are listed.
- onlyDiscrete, if set to true, specifies that only fields with a discrete (that is, flag, set, or typeless) data type are listed.

Thus, for example, a control that specifies storage="[integer]" and types="[flag]" ensures that only integer fields that are flags will appear in the list.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Note: This control is used for Node element definitions only. To specify a multi-field chooser within an Output Data Model definition, use the following format:

```
<OutputDataModel mode="mode">
 ...
     <ForEach var="field" inProperty="prop_name">
          <AddField name="${field_name}_NEW" fieldRef="${field_name}" />
     </ForEach>
 ...
</OutputDataModel>
```

See the topic "Output Data Model" on page 55 for more information. The ForEach element is described under "Iteration with the ForEach Element" on page 64. AddField is described under "Add Field" on page 61.

Example

The following example shows the code used to specify the multi-field chooser control in the preceding illustration.

```
<MultiFieldChooserControl property="inputs" >
     <Enabled>
          <Condition control="custom_fields" op="equals" value="true"/>
     </Enabled>
</MultiFieldChooserControl>
```

The Enabled section causes the control to be enabled only if the custom_fields control is selected.

Note: The contents of this list are governed by the declaration for the inputs property in the Properties section for the relevant node:

```
<Property name="inputs" valueType="string" isList="true" label="Inputs" labelKey="inputs.
LABEL"/>
```

Multi-Item Chooser Control

Defines a control that enables the user to choose one or more items from a list of values. It associates a property with a catalog that holds a list of values. See the topic "Catalogs" on page 39 for more information.

Figure 60. Multi-item chooser control

Format

```
<MultiItemChooserControl controller_attributes catalog="catalog_name" >
     -- advanced custom layout options --
</MultiItemChooserControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`catalog` (required) is the name of the catalog to be associated. The library from which the catalog is obtained is the one specified in the `Module` element of the Execution section. See the topic "Modules" on page 53 for more information.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

```
<MultiItemChooserControl property="selection2" catalog="cat2" />
```

The property referenced by the `property` attribute (`selection2` in this case) must be one with an `isList="true"` attribute. For an explanation and example of the use of `MultiItemChooserControl`, see "Catalogs" on page 39.

Password Box Control

Defines a single-line text field where input characters are hidden as they are typed.

Figure 61. Password box control

Format

```
<PasswordBoxControl controller_attributes columns="integer" >
     -- advanced custom layout options --
</PasswordBoxControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`columns` is a positive integer that defines the number of character columns that the password box should occupy. Default is 20.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

```
<PasswordBoxControl property="encrypted_string1" label="Encrypted string 1" labelKey=
"encryptedString1.LABEL"/>
```

The text field is encrypted by associating it with a property that is defined as an encrypted string in the Properties section for the relevant node:

```
<Property name="encrypted_string1" valueType="encryptedString"/>
```

Property Control

A property control is a completely user-definable control that enables users to enter properties for the node. The processing is handled by a user-written Java class. The following illustration is an example of a

property control.

Figure 62. Section of dialog with example of a property control highlighted

Format

```
<PropertyControl controller_attributes controlClass="Java_class" >
     -- advanced custom layout options --
</PropertyControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`controlClass` (required) is the path within a *.jar* file to the Java class that implements the property control. (*Note*: The *.jar* file is declared in a `JarFile` element in the Resources section. See the topic "Jar Files" on page 33 for more information.)

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

```
<PropertyControl property="target_field_values_specify" labelAbove="true"
     controlClass="com.spss.clef.selflearning.propertycontrols.list.CustomListControl" label=""
     labelKey="target_field_values_specify.LABEL">
     <Enabled>
          <Condition control="target_field_values" op="equals" value="specify"/>
     </Enabled>
     <Layout rowIncrement="2" />
</PropertyControl>
```

The property control is associated with a property that is defined in the Properties section for the relevant node:

```
<Property name="target_field_values_specify" valueType="string" isList="true" label=""
     labelKey="target_field_values_specify.LABEL"/>
```

Radio Button Group Control

Defines a set of radio buttons, where only one button can be selected at a time.

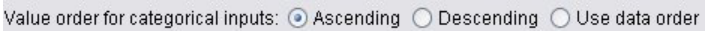

Figure 63. Radio button group control

Each radio button group control has a property attribute that associates the group with a particular property. This property is defined elsewhere in the file and specifies the buttons that make up the group.

The associated property can be either an enumerated list or a Boolean property. For enumerated lists (where the property attribute `valueType="enum"`), one radio button is displayed for each `enum` value. For Boolean properties (where `valueType="boolean"`), two radio buttons are always displayed.

Format

```
<RadioButtonGroupControl controller_attributes
     rows="integer" layoutByRow="true_false" useSubPanel="true_false"
     falseLabel="button_label" falseLabelKey="label_key" trueLabel="?button_label"
     trueLabelKey="label_key" trueFirst="true_false" >
     -- advanced custom layout options --
</RadioButtonGroupControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`rows` is a positive integer that specifies the number of screen rows over which the group will be displayed. Default is 1.

`layoutByRow` specifies whether the radio buttons are to be laid out along the row first (`true`) or down the column first (`false`). Default is `true`. For an example of using `layoutByRow` with a radio button group, see "Changing the Order of Controls" on page 143.

`useSubPanel` specifies whether (`true`) or not (`false`) the radio buttons are to be laid out as a sub-panel. Default is `true`.

Radio button groups are normally laid out as a sub-panel containing all the buttons in the group. However, this can result in alignment problems if the radio button group is associated with an adjacent text field. Setting `useSubPanel` to `false` overcomes this problem.

`falseLabel` is the label for the "false" value of a Boolean property (see the second example below). Only used with Boolean properties, in which case it is required.

`falseLabelKey` identifies the "false" label for localization purposes.

`trueLabel` is the label for the "true" value of a Boolean property (see the second example below). Only used with Boolean properties, in which case it is required.

`trueLabelKey` identifies the "true" label for localization purposes.

`trueFirst`, if set to `true`, causes the display order of the buttons for a Boolean property to be reversed, so that the button representing the "true" value is displayed first. Default is `false`, meaning that the button representing the "false" value is displayed first.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Examples

The first example illustrates the code used for the radio button group shown earlier.

```
<RadioButtonGroupControl property="value_order" labelWidth="2">
     <Layout gridWidth="4"/>
</RadioButtonGroupControl>
```

The `Layout` element is described under "Advanced Custom Layout" on page 143.

Note: The number of buttons and their labels are defined in the Properties section for the relevant node; in this case, as an enumerated list in the declaration for the `value_order` property. This declaration also includes the label for the group itself:

```
<Property name="value_order" valueType="enum" label="Value order for categorical
   inputs" labelKey="value_order.LABEL">
     <Enumeration>
          <Enum value="Ascending" label="Ascending" labelKey="value_order.Ascending.LABEL"/>
          <Enum value="Descending" label="Descending" labelKey="value_order.Descending.
           LABEL"/>
          <Enum value="DataOrder" label="Use data order" labelKey="value_order.UseDataOrder.
           LABEL"/>
     </Enumeration>
</Property>
```

The second example illustrates the use of `falseLabel` and `trueLabel` for a radio button group that controls a Boolean property, such as one to control whether standard or custom settings are enabled.

Figure 64. Radio button group controlling a Boolean property

The code to achieve this is:

```
<RadioButtonGroupControl property="boolean5" label="Boolean 5" labelKey="boolean5.LABEL"
     falseLabel="Standard" falseLabelKey="boolean5.false.LABEL" trueLabel="Custom"
     trueLabelKey="boolean5.true.LABEL" />
```

In this case, both the button labels and the group label are defined in the `RadioButtonGroupControl` element itself. The property with which the group is associated is defined in the Properties section for the node:

```
<Property name="boolean5" valueType="boolean" defaultValue="false"/>
```

Server Directory Chooser Control

Defines a single-line text field and associated button that allow the user to select a directory on the server. The directory must already exist. Users can either open a file from this directory or save a file to it, depending on the mode setting.

Figure 65. Server directory chooser control

The user can either enter the directory path and name directly in the text field or click the adjacent button to display a dialog from where they can select a directory.

Format

```
<ServerDirectoryChooserControl controller_attributes mode="chooser_mode" >
     -- advanced custom layout options --
</ServerDirectoryChooserControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`mode` determines the button displayed on the dialog from where users choose a directory and is one of the following:

- `open` (default) displays an **Open** button.

- `save` displays a **Save** button.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

```
<ServerDirectoryChooserControl property="directory1" label="Server Directory"
     labelKey="directory1.LABEL"/>
```

Server File Chooser Control

Defines a single-line text field and associated button that allow the user to select a file on the server. The file must already exist. Users can either open the file or save it, depending on the mode setting.

Figure 66. Server file chooser control

The user can either enter the file path and name directly in the text field or click the adjacent button to display a dialog from where they can select a file.

Format

```
<ServerFileChooserControl controller_attributes mode="chooser_mode" >
     -- advanced custom layout options --
</ServerFileChooserControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`mode` determines the button displayed on the dialog from where users choose a file and is one of the following:

- `open` (default) displays an **Open** button.
- `save` displays a **Save** button.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

```
<ServerFileChooserControl property="file1" label="Server File" labelKey="file1.LABEL"/>
```

Single-Field Chooser Control

Defines a control that enables the user to choose a single field from a list.

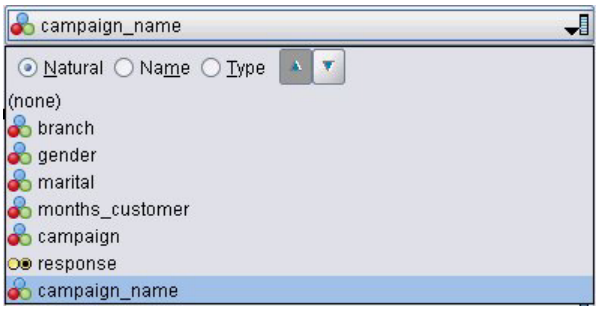

Figure 67. Single-field chooser control

When the user clicks this control, a list of fields is displayed from which a single field can be chosen.

The set consists of all the fields that are visible at this node. If fields have been filtered further upstream from this node, only the fields that have passed through the filter are visible. The list can also be further restricted by specifying that only fields with particular storage and data types are to be available for selection.

Format

```
<SingleFieldChooserControl controller_attributes storage="storage_types" onlyNumeric="true_
     false" onlySymbolic="true_false" onlyDatetime="true_false" types="data_types"
     onlyRanges="true_false" onlyDiscrete="true_false" >
     -- advanced custom layout options --
</SingleFieldChooserControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

In addition, you can further restrict the list of fields by specifying two other attributes, one of which must be from the following list:

- `storage` is a list property that specifies the storage type of fields to be allowed in the list--for example, `storage="[integer real]"` means that only fields with these storage types will be listed. For the set of possible storage types, see the table under "Data and Storage Types" on page 175.
- `onlyNumeric`, if set to `true`, specifies that only fields with a numeric storage type are listed.
- `onlySymbolic`, if set to `true`, specifies that only fields with a symbolic (that is, string) storage type are listed.
- `onlyDatetime`, if set to `true`, specifies that only fields with a date-and-time storage type are listed.

The second attribute specified must be from this list:

- `types` is a list property that specifies the data type of fields to be allowed in the list--for example, `types="[range flag]"` means that only fields with these storage types will be listed. The set of possible data types is:

  `range`

  `flag`

  `set`

  `orderedSet`

  `numeric`

  `discrete`

  `typeless`
- `onlyRanges`, if set to `true`, specifies that only fields with a range data type are listed.

- `onlyDiscrete`, if set to `true`, specifies that only fields with a discrete (that is, flag, set, or typeless) data type are listed.

Thus, for example, a control that specifies `storage="[integer]"` and `types="[flag]"` ensures that only integer fields that are flags will appear in the list.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Note: This control is used for node definitions only. To specify a multi-field chooser within an Output Data Model definition, use the following format:

```
<OutputDataModel mode="mode">
 ...
     <ForEach var="field" from="1" to="${integer}">
          <AddField name="${string}_${field}" fieldRef="${field_ref}" />
     </ForEach>
 ...
</OutputDataModel>
```

See the topic "Output Data Model" on page 55 for more information. The `ForEach` element is described under "Iteration with the ForEach Element" on page 64. `AddField` is described under "Add Field" on page 61.

Example

The following example shows the code used to specify the single-field chooser control in the preceding illustration.

```
<SingleFieldChooserControl property="target"  storage="string" onlyDiscrete="true"/>
```

Note: The actual contents of the list are defined in the Properties section for the relevant node; in this case, in the declaration for the `target` property:

```
<Property name="target" valueType="string" label="Target field" labelKey="target.LABEL"/>
```

Single-Item Chooser Control

Defines a control that enables the user to choose a single item from a list of values. It associates a property with a catalog that holds a list of values. See the topic "Catalogs" on page 39 for more information.

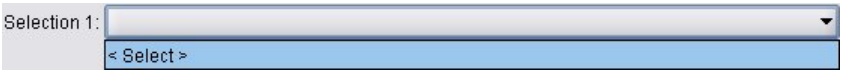

Figure 68. Single-item chooser control

Format

```
<SingleItemChooserControl controller_attributes catalog="catalog_name" >
     -- advanced custom layout options --
</MultiItemChooserControl
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`catalog` (required) is the name of the catalog to be associated. The library from which the catalog is obtained is the one specified in the `Module` element of the Execution section. See the topic "Modules" on page 53 for more information.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

```
<SingleItemChooserControl property="selection1" catalog="cat1" />
```

For an explanation and example of the use of this control, see "Catalogs" on page 39.

Spinner Control

Defines a spinner (a numeric field with up and down arrows to change the field value).

Figure 69. Spinner

Format

```
<SpinnerControl controller_attributes columns="integer" stepSize="increment"
     minDecimalDigits="number" maxDecimalDigits="number" >
     -- advanced custom layout options --
</SpinnerControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`columns` is a positive integer specifying the number of character columns that the control spans. Default is 5.

`stepSize` is a decimal number that specifies the amount by which the field value changes when the user clicks one of the arrows. Default is 1.0.

`minDecimalDigits` is the minimum number of decimal places to be displayed for the field value. Default is 1.

`maxDecimalDigits` is the maximum number of decimal places to be displayed for the field value.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

The following example shows the code used to specify the spinner control in the preceding illustration:

```
<SpinnerControl property="double1" label="Double 1" labelKey="double1.LABEL"/>
```

The precision and valid range for the numeric field contents are defined in the Properties section for the relevant node; in this case, in the declaration for the `double1` property:

```
<Property name="double1" valueType="double" min="0" max="100"/>
```

Table Control

Defines a table layout item to be displayed on a node dialog or output window.

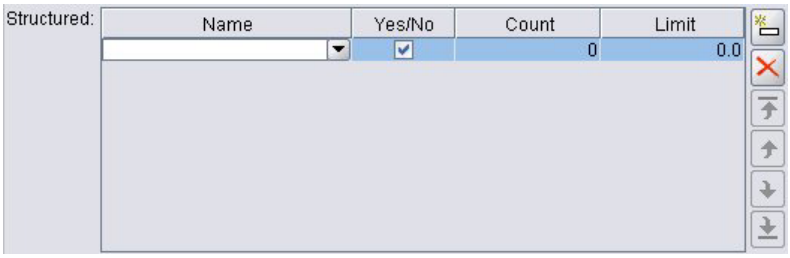

Figure 70. Table control

Format

```
<TableControl controller_attributes rows="integer" columns="integer" columnWidths="list" >
     -- advanced custom layout options --
</TableControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

`rows` is a positive integer specifying the number of table rows that are visible on the screen. Default is 8.

`columns` is a positive integer specifying the number of character columns that the table spans. Default is 20.

`columnwidths` is a list of values that specifies the relative column widths. Thus for example a value of `[30 5 10]` specifies that column 1 is three times as wide as column 3.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

The *ColumnControl* attribute in the following example is as described under "Column Control" on page 140.

Example

The code to specify the table control in the preceding illustration is:

```
<TableControl property="structure1" allowReorder="true" label="Structured"
  labelKey="structure1.LABEL" columnWidths="[20 6 10 10]">
     <ColumnControl column="0" editor="fieldValue" fieldControl="field1"/>
</TableControl>
```

The structure of the table control is defined as a property type in the Common Objects section of the specification file:

```
<PropertyType id="shared_structure1" valueType="structure" isList="true">
     <Structure>
          <Attribute name="id" valueType="string" label="Name" labelKey="structure1.id.LABEL"/>
          <Attribute name="yesno" valueType="boolean" label="Yes/No" labelKey="structure1.
           yesno.LABEL" defaultValue="true"/>
          <Attribute name="count" valueType="integer" label="Count" labelKey="structure1.
           count.LABEL" defaultValue="0"/>
          <Attribute name="limit" valueType="double" label="Limit" labelKey="structure1.
           limit.LABEL" defaultValue="0.0"/>
     </Structure>
</PropertyType>
```

In the node specification, the identifier of this property type is then associated with the identifier of the table control by means of a property declaration:

```
<Property name="structure1" type="shared_structure1"/>
```

If referring to the node in a script, you can set the values in the property using square brackets [] for the list and braces {} for the structure. For example, you could set a grid of two structures for the `structure1` property as follows:

```
set :node_ID.structure1 = [{"hello" true 4 0.21} {"bye" false 5 0.95}]
```

Note that the value order must be consistent with the order in which the `Attribute` definitions are made.

Column Control: Defines the layout of columns in tables.

Every column of the table control shares the same data type; based on this, you can specify one editor for a certain column for all rows. Therefore, every column only needs one editor to edit. For example, if column X needs the user to input an integer, you can set an integer editor for column X.

The attribute *editor* specifies the editor type of the column. There are four types of editor: *default*, *field*, *fieldValue*, and *enumeration*, and each type of editor is an editable comboBox.

For the *fieldValue* type editor, the drop down list contains all values of the field which you specified in *fieldControl*. So the following XML element defines that when you edit column 0, the editor is a combo box and the drop down list contains all values of field1:

```
<ColumnControl column="0" editor="fieldValue" fieldControl="field1" />
```

You can replace *fieldControl* with *fieldDirection*. For example: *fieldDirecton="[in out]"*, means the combo box drop down list will contain all values of the first one the fields whose direction is *in* or *out*.

For the *field* type editor, the drop down list contains all fields that fit the field filter condition. The following example defines that column 0 uses a field combo box as the editor and the drop down list contains all real and integer fields:

```
<ColumnControl column="0" editor="field" storage="[real integer]" />
```

In addition, you can use attribute *types* to specify the measure types that the fields in the drop down list should obey. The Boolean attributes which fit the field are: *onlyRanges*, *onlyDiscrete*, *onlyNumeric*, *onlySymbolic*, and *onlyDatetime*.

Text Area Control

Defines a multi-line text entry field.

Figure 71. Text area control

Format

```
<TextAreaControl controller_attributes rows="integer" columns="integer" wrapLines="true_false" >
     -- advanced custom layout options --
</TextAreaControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

rows is a positive integer specifying the number of screen rows that the text area occupies. Default is 8.

columns is a positive integer specifying the number of character columns that the text area spans. Default is 20.

wrapLines specifies whether to use line wrap for long text lines (true) or to require horizontal scrolling to read long text lines (false). Default is true.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

The code to create the example shown earlier is:

```
<TextAreaControl property="string2" label="String 2" labelKey="string2.LABEL"/>
```

In this case, the label for the text area is defined in the text area control declaration, while the input data type is defined in the Properties section for the relevant node; in the declaration for the string2 property:

```
<Property name="string2" valueType="string"/>
```

Text Box Control

Defines a single-line text entry field.

Figure 72. Text box control

Format

```
<TextBoxControl controller_attributes columns="integer" >
     -- advanced custom layout options --
</TextBoxControl>
```

where:

controller_attributes are as described under "Controller Attributes" on page 122.

columns is a positive integer specifying the number of character columns that the text box spans. Default is 20.

The advanced custom layout options provide a fine degree of control over the positioning and display of screen components. See the topic "Advanced Custom Layout" on page 143 for more information.

Example

The code to create the text box shown earlier is:

```
<TextBoxControl property="string1" label="String 1" labelKey="string1.LABEL"/>
```

The input data type for the text box is defined in the Properties section for the relevant node; in this case, in the declaration for the string1 property:

```
<Property name="string1" valueType="string"/>
```

Property Control Layouts

This section describes the standard layout methods used for dialogs and windows, as well as ways to modify these to obtain your own custom layouts.

Standard Control Layout

A properties panel can be considered as a two-dimensional grid of cells. Each row can have a different height, and each column can have a different width. The UI components can be allocated to multiple contiguous cells, although typically a UI component is allocated to only one cell.

By default, one property control is allocated to one row, and each control takes up two columns: one for the label and one for the control component or components. The column containing the labels expands to the width of the widest label. For example, given the following elements in the specification file:

```
<TextBoxControl property="string1" label="String 1"/>
<PasswordBoxControl property="encryptedString1" label="Encrypted string 1"/>
<TextAreaControl property="string2" label="String 2"/>
```

the resulting panel is shown in the following figure.

Figure 73. Simple properties panel

Note that the ":" character at the end of the label is added automatically.

A property control that contains multiple user interface components creates its own invisible rectangular area in which to lay out those components. The `RadioButtonGroupControl` and `CheckBoxGroupControl` elements are examples of such controls.

Note that the shape of the rectangular area in which the components are laid out can differ depending on the property control. Thus the layout of different controls may not always be strictly aligned.

Some property controls include components that completely fill the component column, and resize horizontally when the window width is enlarged or reduced. Examples of this are the controls specified by the `TextBoxControl`, `PasswordBoxControl` and `TextAreaControl` elements. However, not all components do this. For example, check boxes and spinner controls only take up a fixed amount of horizontal space, even when the window width is enlarged:

Custom Control Layout

The standard layout of controls can be modified in a number of ways, some of them simple and some of them more complex.

Simple Custom Layout

Three simple ways of customizing the control layout are:

- Position a label above its component
- Change the number of rows over which the controls are laid out
- Change the order in which the controls are laid out

Positioning a Label Above Its Component: You can position a label in a separate row above its component by setting the `labelAbove` attribute of the control to `true`. For example:

```
<TextBoxControl property="string0" label="String 0" labelAbove="true"/>
<TextBoxControl property="string1" label="String 1"/>
<PasswordBoxControl property="encryptedString1" label="Encrypted string 1"/>
```

As well as positioning the label above the component, the actual UI component or components are allocated to the label column of the display. This results in the following panel, with the label **String 0** displayed above its corresponding field.

Figure 74. Panel with field label in separate row

Changing the Number of Rows: By default, radio button and check box groups are laid out over a single row, and the width of the dialog is adjusted to accommodate them. If a radio button or check box group has a large number of options, this can result in a very wide dialog. You can avoid this by changing the number of rows used to lay out the control. You do this by setting the `rows` attribute of the control definition to the desired value. For example:

```
<RadioButtonGroupControl property="enum1" label="Enum 1" rows="2"/>
```

This results in a panel where the radio button group is laid out over two rows.

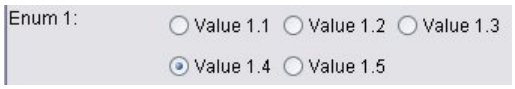

Figure 75. Panel with radio button group laid out over two rows

Changing the Order of Controls: For radio button and check box groups, you can also change the order in which the controls for each `enum` value are added to the panel.

By default, the controls are added in row order, as in the previous example, where the first, second, and third values are added to the first row, with the fourth and fifth values added to the second row. Instead, you can add the controls in column order within the specified number of rows by setting `layoutByRow` to `false`. For example:

```
<RadioButtonGroupControl property="enum1" label="Enum 1" rows="2" layoutByRow="false"/>
```

The values are still displayed over two rows, but now the first and second values are added to the first column, the third and fourth values to the second column, and the fifth value to the third column.

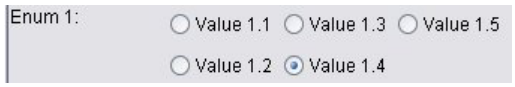

Figure 76. Panel with radio button group laid out in column order

For Boolean properties displayed as two radio buttons, the default ordering behavior is to display the "False" button before the "True" button. You can reverse this order by setting the `trueFirst` attribute to `true`.

You can also prevent radio button and check box groups from using a sub-panel by setting the `useSubPanel` attribute to `false`. However, this can lead to some undesirable layout behavior unless used in conjunction with the `Layout` element (see "Specifying Precise Control Positions with the Layout Element" on page 144).

Advanced Custom Layout

Within each control declaration, you can specify complex control layouts using various elements. You can:

- Specify precise control positions on the screen with the `Layout` element

- Control display characteristics with the `Enabled` element
- Control visibility of screen components with the `Visible` element

Specifying Precise Control Positions with the Layout Element: Precise layout positioning can be achieved by specifying an explicit `Layout` element and associating it with the control.

Format

```
<property_control ... >
     <Layout attributes
      --- cell specification ---
          ...
</property_control>
```

where:

property_control is one of the property controls (see "Property Control Specifications" on page 115).

attributes are any of the attributes shown in the following table.

Table 39. Layout attributes.

Attribute	Values	Description

`anchor`	`north`
	`northeast`
	`east`
	`southeast`
	`south`
	`southwest`
	`west`
	`northwest`
	`center`
		Defines the anchor point for the control.

`columnWeight`	0.0–1.0	Defines how a change in the horizontal size of the window affects the width of the control. The sum of all `columnWeight` attributes in a panel should not exceed 1.0.

`fill`	`none`
	`horizontal`
	`vertical`
	`both`
		Defines whether and how the control should fill the cells to which it has been allocated.

`gridColumn`	integer >= 0	Defines the first column where the control should start being laid out.
`gridHeight`	integer	Defines the number of rows across which the control should be laid out. A value of 0 (the default) allocates the control across all remaining rows.
`gridRow`	integer >= 0	Defines the first row where the control should be laid out. By default, the grid row index is incremented automatically.
`gridWidth`	integer	Defines the number of columns across which the control should be laid out. A value of 0 (the default) allocates the control across all remaining columns.
`leftIndent`	integer	Defines the number of pixels by which to indent the control from its default position.
`rowWeight`	0.0–1.0	Defines how a change in the vertical size of the window affects the height of the control. The sum of all `rowWeight` attributes in a panel should not exceed 1.0.

A **cell specification** enables you to specify the precise position of a control on the screen. The format is as follows:

```
<Cell row="integer" column="integer" width="integer" />
```

where:

`row` (required) is a non-negative integer specifying the row position for the start of the control.

`column` (required) is a non-negative integer specifying the column position for the start of the control.

`width` (required) is a non-negative integer specifying the number of screen grid columns that the control occupies.

Thus, for example, assuming a screen grid of three columns and three rows, consider a custom control layout in the following format.

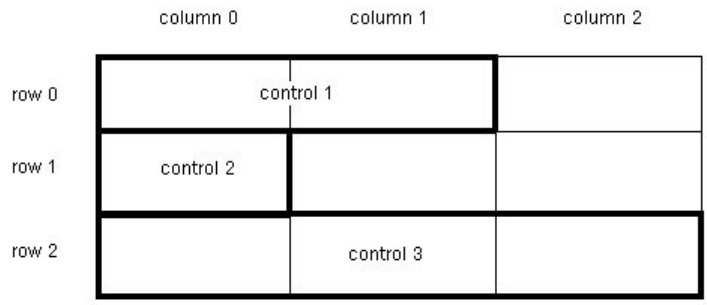

Figure 77. Example of control layout using cells

This format would need a `Layout` element with the following cell specifications:

```
<Layout ... >
     <Cell row="0" column="0" width="2">
     <Cell row="1" column="0" width="1">
     <Cell row="2" column="0" width="3">
</Layout>
```

Some more detailed examples follow providing further illustrations of how you can use the `Layout` element.

Example: Check Box Enabling a Text Field: This example illustrates the use of a check box to enable a text field on the same line of the display.

When using a check box to enable another control on the same line, a simple `Layout` element is needed to cause the controls to display correctly. (*Note*: The mechanism for enabling and disabling controls is described under "Controlling Display Characteristics with the Enabled Element" on page 150).

Assume that we want to achieve the following panel in a display.

Figure 78. Check box enabling a text field

There are two controls here:

- A check box with a label that also acts as the label for a text field
- The text field itself

The starting point is a normal declaration of the two controls:

```
<CheckBoxControl property="boolean3" label="Check box 3"/>
<TextBoxControl property="string3" label="String 3"/>
```

This results in the following panel.

Figure 79. Check box and text field in separate rows

First of all, we want to prevent display of the text field label **String 3**. This is achieved by setting the `showLabel` attribute of the text field control to `false`:

```
<CheckBoxControl property="boolean3" label="Check box 3"/>
<TextBoxControl property="string3" label="String 3" showLabel="false"/>
```

The text field expands to fill the area previously occupied by the label.

Figure 80. Check box, and text field with no label

Now we want to cause the text field to display on the same line as the check box. To do this, we add a `Layout` element within the `CheckBoxControl` element to set the row increment to 0 (by default the row is incremented by 1 for each control):

```
<CheckBoxControl property="boolean3" label="Check box 3">
     <Layout rowIncrement="0"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false"/>
```

However, this results in the following display.

Figure 81. Text field overwriting check box

The text field has moved up one line, but it still occupies the whole row, so it has overwritten the check box.

Note: The check box is being drawn after the text field if the display looks like the following display.

Figure 82. Check box overwriting text field

The first few characters of the text field are obscured.

Whichever object is drawn first, allocating multiple UI components to the same cell results in undesirable or undefined behavior and should be avoided. To resolve the problem, we need to add a second `Layout` element, this time within the `TextBoxControl` element, to force the text field to start in the second column of the display:

```
<CheckBoxControl property="boolean3" label="Check box 3">
     <Layout rowIncrement="0"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false">
     <Layout gridColumn="1"/>
</TextBoxControl>
```

However, this is only a partial solution. Both controls are placed correctly, but the text field is too short, as shown in the following display.

Figure 83. Correct placement but very short text field

The issue is that once a custom layout has been associated with a control, this overrides the "smart" defaults associated with each type of control. In this case, the default fill behavior of the `Layout` element (that is, how the component fills the cells available to it) is not to fill the available cells and so occupy as little space on the screen as possible. To change this, we simply tell the text field to fill the horizontal space:

```
<CheckBoxControl property="boolean3" label="Check box 3">
     <Layout rowIncrement="0"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

The addition of a small `columnWeight` value is required to cause Java to allocate the filled space correctly.

This gives us our intended layout.

Figure 84. Check box enabling a text field

This looks correct, but there is still one issue to address. Currently the check box attempts to occupy the whole of the row, even though we are now putting another control on the same row. The issue is not currently visible because the check box label is relatively short, and other labels on the panel (not shown in the illustration) have moved the second display column out so that there is no overlap. The problem becomes obvious if we make the check box label longer:

```
<CheckBoxControl property="boolean3" label="Check box 3 with a much longer label than we had">
     <Layout rowIncrement="0"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

Doing so gives us the following display.

Figure 85. Long check box label overwritten by text field

All we need to do is to tell the check box to limit its available width to a single column:

```
<CheckBoxControl property="boolean3" label="Check box 3 with a much longer label than we had">
     <Layout rowIncrement="0" gridWidth="1"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

This finally gives us what we need.

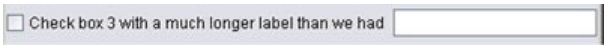

Figure 86. Long check box label fully displayed

Example: Radio Button Group and Text Fields: This example illustrates a way of associating each button in a radio button group with its own text field.

We want to define a panel that looks like the following display.

Figure 87. Radio button group with text fields

This time we have four controls:

- A radio button group for an enumerated list of three values
- Three text fields, one for each value

As in the previous example, we start with a simple declaration of the controls:

```
<RadioButtonGroupControl property="enum4" label="Enum 4"/>
<TextBoxControl property="string4" label="String 4"/>
<TextBoxControl property="string5" label="String 5"/>
<TextBoxControl property="string6" label="String 6"/>
```

Doing so gives us the following display.

Figure 88. Radio button group with text fields and labels

We want to use the radio button labels to identify the text fields, so our first task is to align the radio buttons in a single column of three rows and to hide the text field labels:

```
<RadioButtonGroupControl property="enum4" label="Enum 4" rows="3"/>
<TextBoxControl property="string4" label="String 4" showLabel="false"/>
<TextBoxControl property="string5" label="String 5" showLabel="false"/>
<TextBoxControl property="string6" label="String 6" showLabel="false"/>
```

This results in the following display.

Figure 89. Radio buttons in single column, and text fields

Already we can see a slight problem here--the radio button group label is not aligned with the first radio button. We will fix this later--right now we need to get our text fields roughly in line with each corresponding radio button.

The procedure is similar to what we did in Example 1. We need to:

- Change the radio button group row increment to 0.
- Limit the grid width so that the text fields and radio buttons do not overlap.
- Lay out each text field in the same row as its radio button.

So we add some `Layout` elements, just as we did in the previous example. In this case, we change the specification file as follows:

```
<RadioButtonGroupControl property="enum4" label="Enum 4" rows="3">
     <Layout rowIncrement="0" gridWidth="1"/>
</RadioButtonGroupControl>
<TextBoxControl property="string4" label="String 4" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
<TextBoxControl property="string5" label="String 5" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
<TextBoxControl property="string6" label="String 6" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

Unfortunately, we now have the following display.

Figure 90. Text fields overwriting the radio buttons

We used exactly the same `Layout` elements that worked in Example 1, so what happened?

The answer is that, unlike the check box control in the previous example, the radio button group (like most controls) has a separate label as well as the actual control. This means the radio button group requires an extra column, so we need to tell the text fields to start a column later, at column 2 instead of column 1. Thus in the `Layout` elements for the text fields, we set the `gridColumn` values to 2:

```
<RadioButtonGroupControl property="enum4" label="Enum 4" rows="3">
     <Layout rowIncrement="0" gridWidth="1"/>
</RadioButtonGroupControl>
<TextBoxControl property="string4" label="String 4" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
<TextBoxControl property="string5" label="String 5" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
```

```
</TextBoxControl>
<TextBoxControl property="string6" label="String 6" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

Note that, although we increment the text field grid column to 2, we do not increase the grid width of the radio button group from 1. This is because for property controls, most `Layout` attributes affect only the UI components that make up the editable part of the control, rather than the control label.

We now have the following display.

Figure 91. Text fields no longer overwrite the radio buttons

This is much closer to what we want. However, there are still some alignment issues between the radio buttons and the text fields.

The main problem is that the radio buttons are being laid out in a separate sub-panel, and so there is no real layout relationship between a radio button and its text field. All we need to do is to stop the radio button group using a sub-panel:

```
<RadioButtonGroupControl property="enum4" label="Enum 4" rows="3" useSubPanel="false">
     <Layout rowIncrement="0" gridWidth="1"/>
</RadioButtonGroupControl>
<TextBoxControl property="string4" label="String 4" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
<TextBoxControl property="string5" label="String 5" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
<TextBoxControl property="string6" label="String 6" showLabel="false">
     <Layout gridColumn="2" fill="horizontal" columnWeight="0.001"/>
</TextBoxControl>
```

Finally, we have the layout we want.

Figure 92. Radio button group with text fields

Controlling Display Characteristics with the Enabled Element: You can use the `Enabled` element to cause a control to be enabled or disabled, usually according to whether a particular condition is satisfied.

Panels and property controls can have conditions associated with them to determine various display characteristics. For example, a check box can be used to enable an associated text field, or a radio button can cause a group of otherwise hidden fields to become visible.

Conditions in the user interface are typically based on the value of another control rather than of a property. Conditions based on properties take effect only when changes have been applied back to the underlying object (for example, node, model output, or document output). In the user interface, controls need to be enabled as soon as a related control has been changed.

Format

```
<Enabled>
     <Condition .../>
     <And ... />
     <Or ... />
     <Not ... />
</Enabled>
```

The `Condition` element specifies a condition to be tested to determine whether the control is enabled.

The `And`, `Or` and `Not` elements enable you to specify compound conditions.

See the topic "Conditions" on page 67 for more information.

Example: Enabling Controls with a Simple Condition: In "Example: Check Box Enabling a Text Field" on page 145, we developed a check box designed to enable a text field when the box is selected.

We want the text field to be enabled as soon as the check box is selected and not when the property of the underlying object is changed. To achieve this, we need to add an `Enabled` condition:

```
<CheckBoxControl property="boolean3" label="Check box 3 with a much longer label than we had">
     <Layout rowIncrement="0" gridWidth="1"/>
</CheckBoxControl>
<TextBoxControl property="string3" label="String 3" showLabel="false">
     <Layout gridColumn="1" fill="horizontal" columnWeight="0.001"/>
          <Enabled>
               <Condition control="boolean3" op="equals" value="true"/>
          </Enabled>
</TextBoxControl>
```

This ensures the text field is only enabled if the Boolean value associated with the check box is true.

Example: Enabling Controls with a Complex Condition: To illustrate the coding of complex conditions, we will look at one of the dialog tabs for the Generalized Linear node, which was developed using CLEF.

The node dialog has an Expert tab, which contains options intended for users with detailed knowledge of such models. All of the options on the tab are initially disabled.

Setting the **Mode** check box to **Expert** enables a number of these options.

However, some are still disabled, such as the **Iterations** control towards the bottom of the dialog. This control is disabled only when **both** of the following conditions are true:

- **Distribution** is set to **Normal**
- **Link function** is set to **Identity**

This combination is actually the default setting for this tab in Expert mode, and changing the setting of either of these combo boxes causes **Iterations** to be enabled.

The code to achieve this is contained within a `PropertiesSubPanel` declaration for the **Iterations** button, as follows:

```
<PropertiesSubPanel buttonLabel="Iterations..." buttonLabelKey= ...
     <Enabled>
          <And>
               <Condition control="mode" op="equals" value="Expert"/>
                    <Not>
                         <And>
                              <Condition control="distribution" op="equals" value="NORMAL"/>
```

```
                              <Condition control="link_function" op="equals" value="IDENTITY"/>
                         </And>
                    </Not>
          </And>
     </Enabled>
      ...
</PropertiesSubPanel>
```

The `Condition` element in the outer `And` section specifies that **Mode** must be set to **Expert** before anything will change. Provided that this condition is true, the `Not` section specifies that the button is not enabled (that is, is disabled) only when *both* conditions of the inner `And` section are satisfied. Thus in Expert mode, **Iterations** is enabled if either **Distribution** or **Link function** is set to something other than their default value.

Controlling Display Characteristics with the Visible Element: You can also use conditions to cause controls to be displayed or hidden according to specified circumstances. You do this by means of the `Visible` element.

Format

```
<Visible>
     <Condition .../>
     <And ... />
     <Or ... />
     <Not ... />
</Visible>
```

The `Condition` element specifies a condition to be tested to determine whether the control is visible.

The `And`, `Or` and `Not` elements enable you to specify compound conditions.

See the topic "Conditions" on page 67 for more information.

Example

The following example causes the specified property panel to be displayed only if the `source_language` condition is satisfied:

```
<PropertiesPanel>
     <Visible>
          <Condition control="source_language" op="equals" value="eng" />
     </Visible>
       ...
</PropertiesPanel>
```

Custom Output Windows

For model output, document output and interactive output objects (but not nodes), it is possible for an extension to completely replace the default output window with a custom window. This is implemented as a standard `java.awt.Frame` class.

To provide a custom window, specify a Java class as the `frameClass` attribute of the `UserInterface` element, as follows:

```
<DocumentOutput id="my_modelling_node" type="modelBuilder" ...>
     <Properties>
          <Property name="use_custom_type" valueType="boolean" .../>
           ...
```

```
    </Properties>
    <UserInterface frameClass="com.myextension.MyOutputFrame"/>
     ...
</DocumentOutput>
```

The specified class must implement the `ExtensionObjectFrame` interface defined by the CLEF client-side API. This covers the life cycle of the window:

- Access to the underlying `java.awt.Frame`
- Window initialization, including access to the output object and the session
- Synchronization of the window and the underlying object when the object is about to be saved or deleted
- Window disposal

See the topic "Client-Side API Classes" on page 167 for more information.

Chapter 7. Adding a Help System

Types of Help Systems

When developing a CLEF extension, you will normally want to include an online help system to explain how to use the extension. CLEF supports the following types of help systems:

- HTML Help
- JavaHelp

HTML Help

HTML Help is a proprietary format developed by Microsoft that runs only on Windows platforms. An HTML Help system consists of individual `.htm` or `.html` files compiled into a compressed format as a single file with the `.chm` extension. IBM SPSS Modeler's own help system is supplied in HTML Help format.

HTML Help supports table of contents, index, and full-text search features (glossary terms can be implemented as pop-up windows). You can create the source `.htm` or `.html` topic files using an HTML editor or a commercial help authoring tool. To produce the `.chm` file, one option is to use the HTML Help Workshop, available as a free download from the Microsoft Download Centre Web site (for more information on producing `.chm` files, see the HTML Help Workshop help system). Alternatively you can use a help authoring tool that supports the HTML Help format to compile your topic files and any graphic files used into a `.chm` file.

JavaHelp

JavaHelp is an open-source help format developed by Sun Microsystems that runs on any platform that supports Java. A JavaHelp system consists of the following files:

- The source `.htm` or `.html` topic files
- Any graphic files used in the topics
- A helpset file (with the extension `.hs`) that controls the help system
- A `map.xml` file, used to associate topic IDs with topic files and to define the window that displays help topics
- An `index.xml` file, which contains the index entries
- A `toc.xml` file, which contains the entries for the table of contents

JavaHelp supports table of contents, index, full-text search, and glossary features. You can create the source `.htm` or `.html` files using an HTML editor or a commercial help authoring tool. You will also need the JavaHelp software, available as a free download from the Sun Developer Network Web site (for more information, see the *JavaHelp System User's Guide*, also available from the Web site).

Implementing a Help System

This section describes how you define the relevant components of the help system in the specification file.

Defining the Help System Location and Type

The type of help system, if any, used for the extension is defined in a `HelpInfo` element in the Resources section of the specification file for the extension.

Format

```
<Resources>
 ...
     <HelpInfo id="name" type="help_type" path="help_path" helpset="helpset_loc"
      default="default_topicID" />
 ...
</Resources>
```

where:

`id` (required) is the identifier of the help information for this extension.

`type` (required) indicates the type of help and is one of the following:

- `htmlhelp`—HTML Help, contained in a `.chm` file identified by the `path` attribute.
- `javahelp`—JavaHelp, using a helpset (`.hs`) file identified by the `helpset` attribute, together with the help source and associated files.

If the help type is `htmlhelp`, the following additional attribute is required:

- `path`—the location (relative to the specification file) and name of the `.chm` file containing the help system.

If the help type is `javahelp`, the following additional attributes are required:

- `helpset`—the location (relative to the specification file) and name of the `.hs` helpset file to be used.
- `default`—the identifier of the default topic to be displayed if no topic has been specified for a particular tab.

If no `HelpInfo` element is specified, no help is associated with this extension.

Examples

The first example illustrates a `HelpInfo` element for HTML Help:

```
<HelpInfo id="help" type="htmlhelp" path="help/mynode.chm" />
```

The equivalent for a JavaHelp system is:

```
<HelpInfo id="help" type="javahelp" helpset="help/mynode.hs"/>
```

Note that in the case of JavaHelp, the associated files (images, map file, index, and contents files) must be located in the same folder as the `.hs` helpset file.

Specifying a Particular Help Topic To Display

You can specify a particular help topic to be displayed if the user invokes help on a node dialog, on a particular tab, or on a properties sub-panel. This is achieved by means of the `helpLink` attribute of the node, tab, or properties sub-panel specification.

If no `helpLink` attribute is specified, the default topic for the help system is displayed if the user invokes help.

For more information, see the descriptions of the `helpLink` attribute in “Node” on page 45, “Tabs” on page 107, and “Properties Sub-Panel” on page 119.

Example

This example assumes that you are using HTML help and shows how you can cause different context-sensitive topics to display depending on which window has focus when the user selects help.

```
<Resources>
      ...
     <HelpInfo id="help" type="htmlhelp" path="help/mynode.chm"/>
      ...
</Resources>
 ...
<Node id="mynode" scriptName="my_node" type="dataTransformer" palette="recordOp"
  label="Sorter" description="Sorts a data file" >
  ...
     <Tabs defaultTab="1">
          <Tab label="Basic Controls" labelKey="basicControlsTab.LABEL"
            helpLink="basic_controls.htm">
              <PropertiesPanel>
               ...
                  <PropertiesSubPanel buttonLabel="Additional settings..."
                    buttonLabelKey="AdditionalOptions.LABEL" dialogTitle="Additional
                    Settings" dialogTitleKey="AdditionalOptionsDialog.LABEL" helpLink=
                    "addsettingsdlg.htm">
  ...
          </Tab>
          <Tab label="Selector Controls" labelKey="selectorControlsTab.LABEL"
           helpLink="selector_controls.htm">
  ...
          </Tab>
  ...
</Node>
```

This specifies that, if the Basic Controls tab has focus and the user selects help, the topic from `basic_controls.htm` in the `mynode.chm` help file is displayed. If the user then clicks the **Additional settings** button to open the Additional Settings dialog and chooses **Help** on that dialog, the topic from `addsettingsdlg.htm` is displayed. If the user then dismisses the Additional Settings dialog, selects the Selector Controls tab and chooses **Help** again, the topic from `selector_controls.htm` is displayed.

For JavaHelp, the value of the `helpLink` attribute must match the value of the `target` attribute in the `map.xml` file. For example, if the `map.xml` file includes the following:

```
<map version="1.0">
 ...
     <mapID target="basic_controls" url="basic_controls.htm"/>
 ...
</map>
```

you must give the corresponding `helpLink` attribute the following value:

```
helpLink="basic_controls"
```

This is because when JavaHelp is called, it reads the value of the `target` attribute and maps it to the associated `url` value to find the correct file to display.

Chapter 8. Localization and Accessibility

Introduction

Localization refers to the process of adapting software, help, and documentation for a particular locale. It includes translation of the user interface, help, and documentation and testing the system in the appropriate locale. If you will be distributing your extension to users in regions other than your own, you can distribute localized versions of the extension.

Accessibility refers, in this context, to the inclusion of features in the user interface to make access to the system easier for users with certain disabilities, such as vision impairments or limited manual dexterity.

Localization

IBM SPSS Modeler itself has been localized for a number of world regions. For any of the supported languages, when the user sets the Windows regional option to their own locale, standard IBM SPSS Modeler UI components are displayed in that language, for example:

- System menus and menu entries
- System buttons (Generate, OK, Execute, Cancel, Apply, Reset)
- Standard dialog box tabs (Annotations and Debug, if used)
- Error and system messages (for example, "This object has not been saved.")

Where your extension uses these standard IBM SPSS Modeler components, they will automatically be displayed in the selected language if it is supported.

For the other components of your extension, CLEF provides a feature to help with localization. You can localize:

- Node names (on palette and canvas)
- Model names (on Models tab of manager pane)
- Document names (on Outputs tab of manager pane)
- Location of icon image associated with an action
- Tooltip text
- Help systems
- Node dialogs:
  - Title bar text
  - Custom menus and menu entries
  - Field, property, button and tab labels
  - Static text
- System and error messages

Text strings should be kept reasonably short to allow for longer text when the item is translated.

System and error messages can be localized through a combination of the specification file, properties files and the server-side API. See the topic “Status Detail Document” on page 184 for more information.

Property Files

The text strings for the items that you can localize are stored in files known as **property files**, which use a standard Java format for storing localization resource bundles. Each property file consists of a series of

records, one for each localized item of the extension. A field in each record corresponds to a `labelKey` attribute in the specification file, enabling CLEF to read the corresponding text string from the property file and display it in the correct place.

A property file must have the extension `.properties`, and must be located in the same directory as the specification file of the node to which it relates. IBM SPSS Modeler looks initially for the default property file, which is named:

path`.properties`

where *path* is the value of the `path` attribute in the `Bundle` element (in the Resources section) that defines the property bundle. For example:

```
<Bundle id="bundle" path="my_resources"/>
```

If there is no default property file, IBM SPSS Modeler reads text strings from the definitions in the specification file.

There must be a property file for each language supported by localization. Files for languages other than the default are distinguished by a suffix in the file name. For example:

```
my_resources.properties
my_resources_de.properties
my_resources_fr.properties
```

Suffixes conform to the two-character ISO 639-1 standard for language codes.

Each record in a property file has the following format:

id=text_string

where:

id is the identifier from a `buttonLabelKey`, `descriptionKey`, `dialogTitleKey`, `falseLabelKey`, `imagePathKey`, `labelKey`, `messageKey`, `textKey`, or `trueLabelKey` attribute in the specification file. This identifier typically has the suffix `.LABEL` to enable it to be distinguished easily, although it can have any suffix, or none at all, depending on how it appears in the specification file.

text_string is the text for the item.

Example: Localizing a Dialog Tab

This example of a localized tab on a node dialog uses two property files, the default (English) version and the French version, with the following locations:

*extension_folder***my_resources**.properties
*extension_folder***my_resources**_fr.properties

where *extension_folder* is the folder containing the specification file.

In the specification file, the property files are referenced by means of a `Bundle` element in the `Resources` section:

```
<Resources>
     <Bundle id="bundle" type="properties" path="my_resources"/>
</Resources>
```

Note that the `path` attribute must not include language extensions or the `.properties` suffix.

The other relevant portions of the specification file are:

```
<Node id="uiTest" scriptName="ui_test" type="dataTransformer" palette="recordOp" label=
"UI Test" ... >
     <Properties>
          <Property name="enum1" valueType="enum" defaultValue="value4">
               <Enumeration>
                    <Enum value="value1" label="Value 1.1" labelKey="enum1.value1.LABEL"/>
                    <Enum value="value2" label="Value 1.2" labelKey="enum1.value2.LABEL"/>
                    <Enum value="value3" label="Value 1.3" labelKey="enum1.value3.LABEL"/>
                    <Enum value="value4" label="Value 1.4" labelKey="enum1.value4.LABEL"/>
                    <Enum value="value5" label="Value 1.5" labelKey="enum1.value5.LABEL"/>
               </Enumeration>
          </Property>
     </Properties>
 ...
     <UserInterface ... >
          <Tabs defaultTab="1">
               <Tab label="Basic Controls" labelKey="basicControlsTab.LABEL" ... >
      ...
     </UserInterface>
 ...
</Node>
```

In the property file, the English version of the property file includes the following records:

```
basicControlsTab.LABEL=Basic Controls
enum1.value1.LABEL=Value 1.1
enum1.value2.LABEL=Value 1.2
enum1.value3.LABEL=Value 1.3
enum1.value4.LABEL=Value 1.4
enum1.value5.LABEL=Value 1.5
```

The parts of the dialog affected by these records are highlighted in the following illustration.

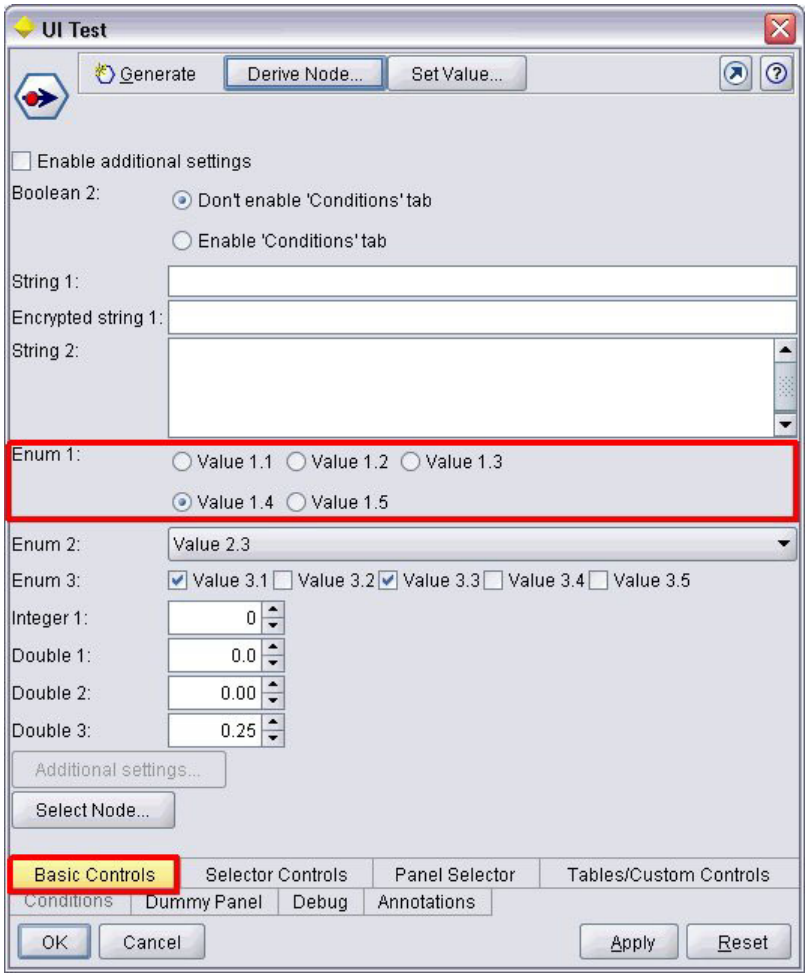

Figure 93. Non-localized tab

The corresponding section in the French version of the property file (`my_resources_fr.properties`) is:

```
basicControlsTab.LABEL=Contrôles de Base
enum1.value1.LABEL=Valeur 1,1
enum1.value2.LABEL=Valeur 1,2
enum1.value3.LABEL=Valeur 1,3
enum1.value4.LABEL=Valeur 1,4
enum1.value5.LABEL=Valeur 1,5
```

These records cause the relevant parts of the screen to display the translated text, as shown in the following illustration.

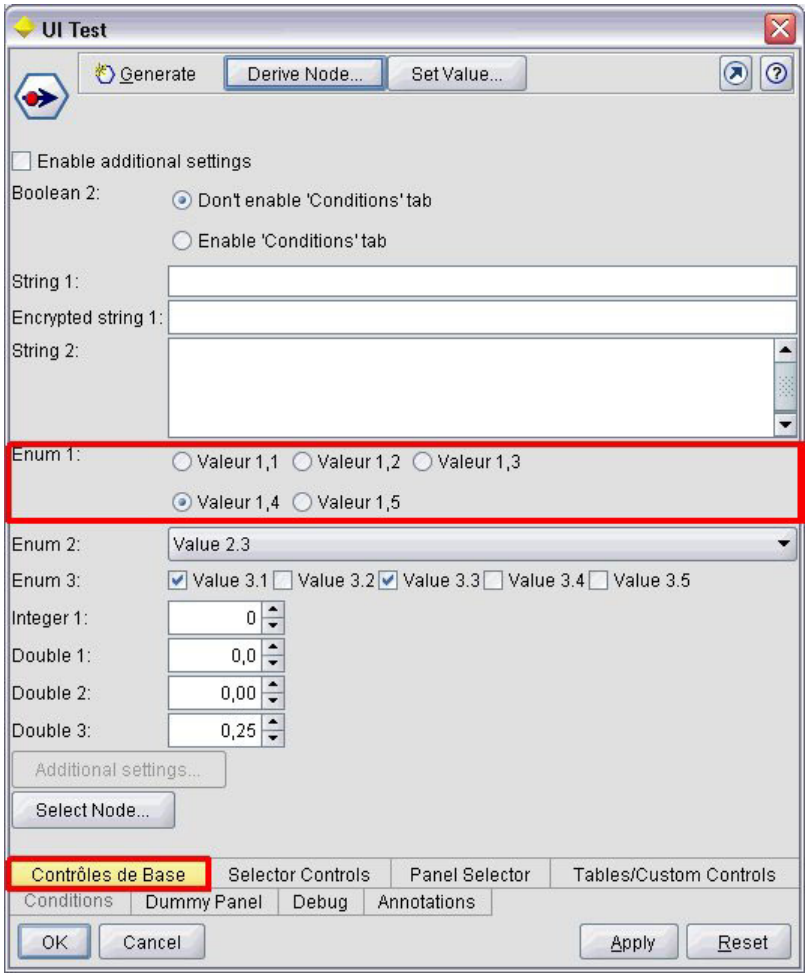

Figure 94. Localized tab

Note that localization of the four buttons at the bottom of the screen is handled by a standard IBM SPSS Modeler feature and not by CLEF.

Example: Using Special Characters

In the property file, you will need to use Unicode escape sequences for any special characters in standard ASCII text strings. Here is part of a property file that has been localized for French:

```
Genlinnode.LABEL=Lin\u00e9aire g\u00e9n\u00e9ralis\u00e9

Fields.LABEL=Champs
Model.LABEL=Mod\u00e8le
Expert.LABEL=Expert

inputFields.LABEL=Entr\u00e9es
targetField.LABEL=Cible
 ...
```

For languages that use non-Latin characters, such as Japanese or Chinese, you will need to use Unicode escape sequences for the entire text strings. Here is the same set of records localized for Japanese:

```
Genlinnode.LABEL=\u4e00\u822c\u5316\u7dda\u578b

Fields.LABEL=\u30d5\u30a3\u30fc\u30eb\u30c9
Model.LABEL=\u30e2\u30c7\u30eb
Expert.LABEL=\u30a8\u30ad\u30b9\u30d1\u30fc\u30c8
```

```
inputFields.LABEL=\u5165\u529b
targetField.LABEL=\u5bfe\u8c61
 ...
```

Help Files

If you are localizing an extension that has a help system, you should also supply a localized version of the help system. You supply one localized help system for each localized extension.

Localizing HTML Help

If the extension you are localizing uses an HTML Help file (with the suffix `.chm`), you can substitute the default `.chm` file with a localized version. For more information on HTML Help systems, see "HTML Help" on page 155.

To create a localized `.chm` file:

1. Create translated versions of the individual help topic (`.htm` or `.html`) files that make up the help system, keeping the same file names.
2. If desired, use localized versions of graphics included in the help system (for example, screen shots).
3. Use Microsoft HTML Help Workshop or another help authoring tool to compile the files into the localized `.chm` file.
4. Test the help system with the localized node. See the topic "Testing a Localized CLEF Node" for more information.

Localizing JavaHelp

If the extension you are localizing uses a JavaHelp system, you will need to provide localized versions of the help source files for each supported language. JavaHelp takes care of displaying the correct localized version if it exists. See the topic "JavaHelp" on page 155 for more information.

To create a localized JavaHelp system:

1. Create translated versions of the individual help topic (`.htm` or `.html`) files that make up the help system, keeping the same file names.
2. If desired, use localized versions of graphics included in the help system (for example, screen shots).
3. Generate the helpset and other required files (map files, contents, and index files).
4. Test the help system with the localized node. See the topic "Testing a Localized CLEF Node" for more information.

Testing a Localized CLEF Node

To test a localized node and its help system:

1. In the Resources section of the specification file for the localized node, change the `path` attribute of the `HelpInfo` element to reference the localized `.chm` or `.hs` file. For example, for HTML Help you might use:

   ```
   <Resources>
    ...
        <HelpInfo id="help" type="HTMLHelp" path="help/mynode_fr.chm "/>
   </Resources>
   ```

   For JavaHelp you might use:

   ```
   <Resources>
    ...
        <HelpInfo id="help" type="javahelp" helpset="help/mynode_fr.hs "/>
   </Resources>
   ```

2. Copy the localized`.chm` or `.jar` file to the location indicated in the `path` attribute.
3. Set the Windows region for the desired locale:

Control Panel > **Regional and Language Options** > **Regional Options** > **Standards and formats** > **<language>**

4. Start IBM SPSS Modeler, ensuring that it displays in the desired language.
5. Add the localized node to IBM SPSS Modeler. See the topic "Testing a CLEF Extension" on page 191 for more information.
6. Place a copy of the node on the canvas.

   Open the node dialog and check that it displays correctly in the desired language.
7. Click the Help button on the dialog and ensure that the correct help topic displays in the desired language.

Accessibility

CLEF nodes benefit from all the standard IBM SPSS Modeler accessibility features, such as keyboard equivalents for mouse actions, and screen reader support.

In addition, you can provide a CLEF node with customized tooltip text for accessibility purposes.

You can also specify keyboard combinations to provide end users with alternative access to various user interface features that you have added in CLEF. See the topic "Access Keys and Keyboard Shortcuts" on page 108 for more information.

For action buttons, and for each of the screen components that are classified as controllers (such as check boxes or radio button groups), you can define:

- label
- description

The **label** is the display text that appears on the screen as the name of the component, and which can be read by screen reader software. For users with visual impairments, the display font size for the label can be changed by controls on the Display tab of the User Options dialog, which can be obtained from:

Tools > **User Options**

The **description** is the tooltip text that is displayed when the mouse pointer hovers over the component. The tooltip provides more information about the component than can be conveyed by the name alone. Tooltips can also be read by a screen reader that is configured to read them.

Labels and descriptions are defined by means of the `label` and `description` attributes in the element that defines the component in the specification file. Both can be localized, by means of the `labelKey` and `descriptionKey` attributes respectively.

Example

This example of an action button illustrates the use of both the label and description features.

```
<Action id="setValue" label="Set Value..." labelKey="setValue.LABEL"
     description="Sets a value" descriptionKey="setValue.TOOLTIP"/>
```

Chapter 9. Programming

About Programming for CLEF Nodes

To enable a node to perform processing that cannot be defined in the specification file, CLEF provides the following application programming interfaces (APIs) to which your programs can make calls:

- **Client-side API**. A Java API that can be used by extensions requiring additional controls, user interface components, or interactivity not provided directly by the specification file.
- **Predictive Server API (PSAPI)**. A Java API that exposes IBM SPSS Modeler functionality for use by applications requiring data mining and predictive analytics capabilities. The PSAPI and the IBM SPSS Modeler data mining workbench share the same underlying technology.
- **Server-side API**. A C-based API that covers aspects such as the setting and getting of execution settings, persistence of those settings, execution feedback, job control (for example, interrupting execution), SQL generation, and returned objects.

CLEF API Documentation

The sections that follow provide an overview of the client-side and server-side APIs. More complete API documentation is included as a zip file in an IBM SPSS Modeler installation and must be extracted before it can be used.

To extract the API documentation:

1. Locate the file *clef_apidoc.zip* on the product DVD, under the *\Documentation\en* folder.
2. Using WinZip or a similar tool, extract the zip file contents to any convenient directory. Doing so creates a *clef_apidoc* subfolder in that directory, containing all of the API documentation.

To view the API documentation:

1. Navigate to the *clef_apidoc* subfolder and open the *clef_apidoc.htm* file.
2. Choose the PSAPI/client-side or the server-side option as desired.

Client-Side API

CLEF provides a number of Java classes containing methods that you can use for client-side processing. For example, the `DataModelProvider` class enables you to compute the output data model where changes to the input data model are too complex to use the features provided by the specification file.

Client-Side API Classes

The client-side classes are as follows.

Table 40. Client-side API classes

Class	Description
`ActionHandler`	Allows the extension to manage actions requested by the user via menu options and toolbar buttons
`DataModelProvider`	Allows nodes that make complex changes to the data model to use Java to compute the output data model
`ExtensionObjectFrame`	Defines the functionality associated with windows used to display model or document outputs
`ExtensionObjectPanel`	Defines the functionality associated with extension object panels

Table 40. Client-side API classes (continued)

Class	Description
`PropertyControl`	Defines the functionality associated with a custom property control on a properties panel

Full details of these classes are given in the client-side API documentation. See the topic "CLEF API Documentation" on page 167 for more information.

Using the Client-Side API

To include client-side function calls in a CLEF node:

1. Create the *.java* source files that contain the function calls.
2. Compile the source files into *.class* files.
3. You can combine multiple *.class* files into a *.jar* file and include a reference to the *.jar* file in the specification file—for example:

```
<Resources>
 ...
     <JarFile id="selfjar" path="selflearning.jar"/>
 ...
</Resources>
```

Some CLEF elements enable you to refer to a class explicitly. For example, you can include a class reference in the `controlClass` attribute of a `PropertyControl` element in the specification file, as shown below:

```
<PropertyControl property="target_field_values_specify" ...
     controlClass="com.spss.clef.selflearning.propertycontrols.list.CustomListControl" ... />
```

where `CustomListControl` is the name of the class that implements the property control, and `com.spss.clef.selflearning.propertycontrols.list` is the path to that class within the *.jar* file declared in the `JarFile` element.

See the topic "Resources Section" on page 32 for more information.

You might also find it useful to take a look at the source code for the example nodes that are supplied with this release. See the topic "Examining the Source Code" on page 26 for more information.

Predictive Server API (PSAPI)

The PSAPI provides a programming interface to the underlying Predictive Server technology. Major elements of the PSAPI are specified as Java interfaces. Most of these interfaces are implemented via internal classes provided by the PSAPI but that do not form part of the PSAPI specification. This approach aims to protect the PSAPI user from changes made to the Predictive Server technology (such as architectural changes, private client/server protocol changes, etc.).

Full details of these classes are given in the PSAPI documentation. See the topic "CLEF API Documentation" on page 167 for more information.

Server-Side API

The server-side API is defined as a C language API but will support implementations in C++. The developer of an extension module can choose to program directly against the C language API by programming in C or C++. Other languages can be used if the developer has a method for binding to the C API. CLEF also provides several C++ helper source files that act as wrappers for some of the C API.

Architecture

An extension **node** on the client is complemented by an extension **peer** on the server. A peer is defined by an **extension module**, which is implemented as a shared library hosted by the server. Communication between a node and its peer is mediated by an extension **resource** managed by the server. A resource calls **service functions** defined by the extension module to create and manipulate its peer, while the peer uses **callback functions** to request information and services from its host.

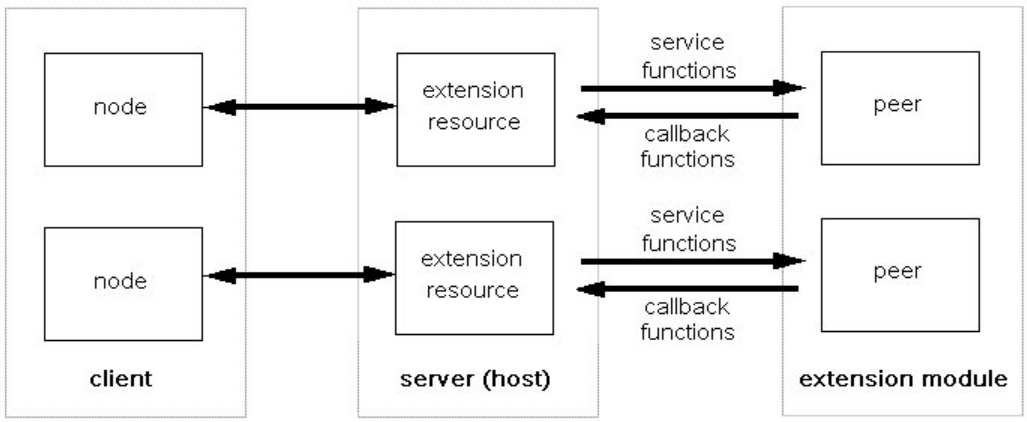

Figure 95. CLEF API architecture

Service Functions

Service functions are implemented by the extension module. An extension module must implement all of the functions marked as required and may implement any or all of those not so marked.

There are two types of service functions:

- Module functions
- Peer functions

The following sections provide an overview of the service functions. More detailed descriptions of these functions can be found in the server-side API documentation, as follows:

1. From the CLEF API Documentation screen, choose **Server-side API overview**.
2. Click the **Modules** tab.
3. Select **API Service Functions to be implemented by the extension module**.

For information on accessing the CLEF API documentation, see "CLEF API Documentation" on page 167.

Module Functions

Module functions are called from a single thread.

Table 41. Module functions

Function	Required?	Description
`clemext_initialise`	Yes	Initializes an extension module
`clemext_cleanup`	Yes	Disposes of resources allocated by an extension module
`clemext_getModuleInformation`	No	Obtains information about an extension module
`clemext_create_peer`	Yes	Creates a peer instance for an extension node
`clemext_destroy_peer`	Yes	Disposes of a peer instance

Peer Functions

Peer functions are applied to a peer instance handle returned by a previous call to `clemext_create_peer`. Peer functions may be called concurrently from separate threads only when the peer handles are distinct. The one exception to this is that the `clemext_peer_cancelExecution` function (if defined) may be called from any thread at any time to interrupt a long-running execution on a different thread.

Table 42. Peer functions

Function	Required?	Description
`clemext_peer_configure`	Yes	Instructs a peer instance to process its parameters and data model
`clemext_peer_getDataModel`	Yes	Obtains the output data model from a peer instance
`clemext_peer_getCatalogueInformation`	No	Obtains catalog information from a module
`clemext_peer_getExecutionRequirements`	No	Obtains the execution requirements of a peer instance
`clemext_peer_beginExecution`	Yes	Begins execution of a peer instance
`clemext_peer_nextRecord`	Yes	Moves to the next record in a peer's result set
`clemext_peer_getRecordValue`	Yes	Returns the value of a specified field within the last-fetched result record
`clemext_peer_endExecution`	Yes	Indicates to a peer instance that execution is complete
`clemext_peer_cancelExecution`	No	Called from a separate thread to interrupt a long-running `beginExecution` or `nextRecord` function call
`clemext_peer_getSQLGeneration`	No	Generates SQL from a peer instance
`clemext_peer_getErrorDetail`	Yes	Retrieves supplementary information about the last module-specific error on a peer

The peer functions shown in the following table are designed for use with Interactive Model Builder.

Table 43. Peer functions used for use with Interactive Model Builder

Function	Required?	Description
`clemext_peer_beginInteraction`	No	Begins interaction with a peer instance
`clemext_peer_request`	No	Performs an interactive request on a peer
`clemext_peer_getRequestReply`	No	Retrieves the reply from the last successfully executed request
`clemext_peer_endInteraction`	No	Indicates to a peer instance that interaction is complete

Callback Functions

When an extension module requires information or service from the host process, it must do so through a **callback**. A callback is applied to a **handle**, which is a pointer that identifies the target of the request.

Callbacks are invoked by passing in the handle of the IBM SPSS Modeler object to which the call is directed. Handles are passed into an extension module as parameters in service functions.

If a callback function fails, it should return some more detail in the associated module-specific error code (denoted by `CLEMEXTErrorCode`). The module may in turn manage this by returning a callback error and passing on this detail so that it can be inspected by the host.

The following types of callback functions are available:

- Host functions
- Node functions
- Iterator functions
- Progress functions
- Channel functions (for interactive models only)

The following sections provide an overview of the callback functions. More detailed descriptions of these functions can be found in the server-side API documentation, as follows:

1. From the CLEF API Documentation screen, choose **Server-side API overview**.
2. Click the **Modules** tab.
3. Select **General callbacks**.

For information on accessing the CLEF API documentation, see "CLEF API Documentation" on page 167.

Host Functions

Host functions are defined on a host handle passed through `clemext_initialise`.

Table 44. Host functions

Function	Description
`clemext_host_getHostInformation`	Obtains static information about the host environment

Node Functions

Node functions are defined on a node handle passed through `clemext_create_peer`.

Table 45. Node functions

Function	Description
`clemext_node_getNodeInformation`	Obtains static information about a node
`clemext_node_getParameters`	Obtains parameters from a node
`clemext_node_getDataModel`	Obtains the input data model for a node
`clemext_node_getOutputDataModel`	Obtains the output data model for a node
`clemext_node_getSQLGeneration`	Obtains the upstream SQL generation information for a node
`clemext_node_getPassword`	Retrieves the plaintext for a password identifier
`clemext_node_getFilePath`	Retrieves the path for a file being exchanged between client and server during execution

Iterator Functions

Iterator functions are defined on an iterator handle passed through `clemext_peer_beginExecution`. An iterator exposes an input dataset to an extension module.

Table 46. Iterator functions

Function	Description
`clemext_iterator_nextRecord`	Moves to the next record in the input dataset
`clemext_iterator_getRecordValue`	Returns the value of a specified field within the last-fetched input record
`clemext_iterator_rewind`	Restores the state of the input dataset such that the next call to `nextRecord` will move to the first record in the set

Progress Functions

Progress functions are defined on a progress handle passed through `clemext_peer_beginExecution`.

Table 47. Progress functions

Function	Description
`clemext_progress_report`	Reports progress back to the host

Channel Functions

Channel functions are used with interactive models only and are defined on a channel handle passed through `clemext_peer_beginInteraction`.

Table 48. Channel functions

Function	Description
`clemext_channel_send`	Sends an asynchronous message to the client, enabling a peer thread to report progress and status information

Process Flow

An extension module calls a number of service and callback functions to perform its processing. The actual functions that are called depend on the processing that the module is required to perform.

Example

The sequence diagram for a typical module execution is as shown in the following illustration.

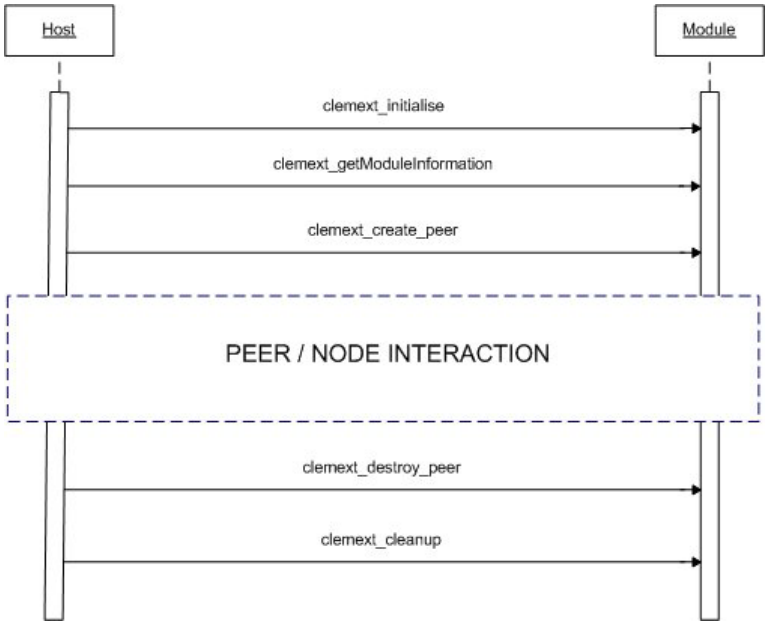

Figure 96. Typical process flow

The peer/node interaction block is shown in the following illustration.

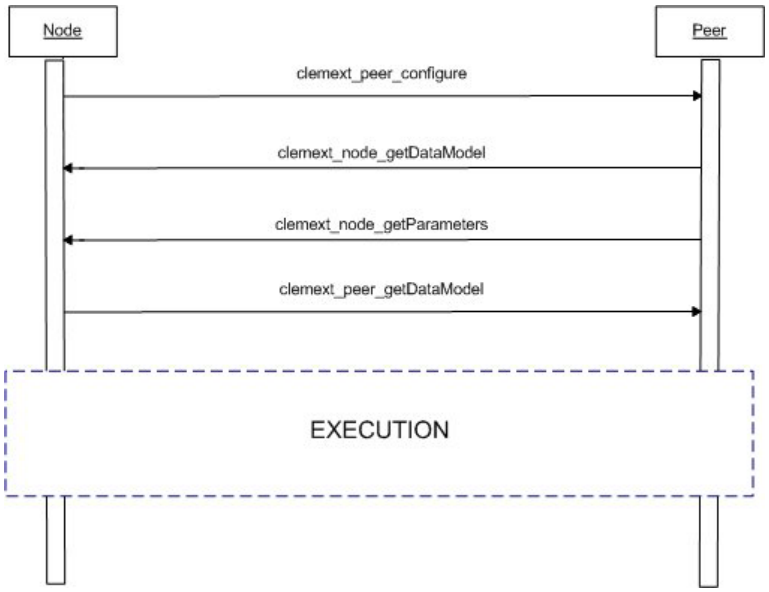

Figure 97. Typical peer/node interaction block

A typical execution block is shown in the following illustration.

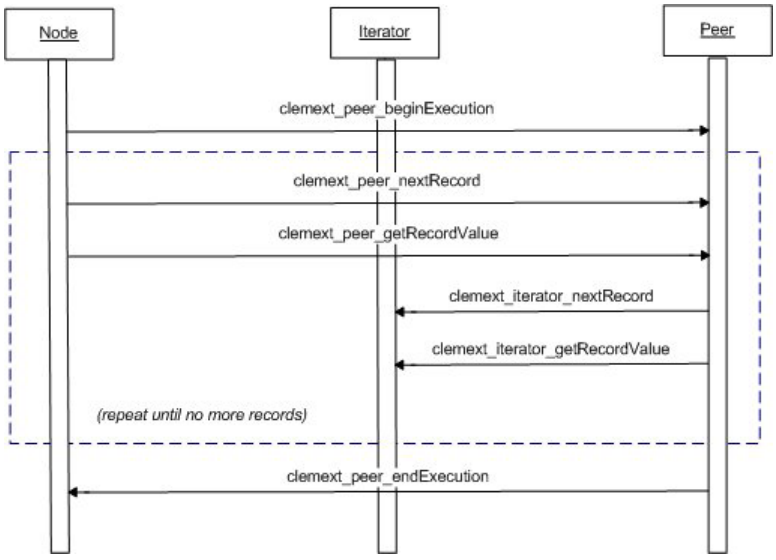

Figure 98. Typical execution block

Notes:

- A module may be loaded into the IBM SPSS Modeler server process at the time the server starts, or it may be loaded later on demand when its services are first required.
- When the module has been loaded, the host invokes the service function `clemext_initialise` once.
- When the module is loaded and initialized, the host may query the module using the service function `clemext_getModuleInformation`.
- After the module is loaded, its services are invoked via peer objects that the module provides. Within the module, the peer object is created by the service function `clemext_create_peer` as a counterpart to the host's node object to manage the execution of a task as directed by the host application. Multiple peer objects of the same type can exist and be executed concurrently within a process at one time.

- When a peer object has been created, it can be configured by the service function `clemext_peer_configure`.
- At this point, the peer may execute callback functions to obtain information from the client, such as `clemext_node_getDataModel` and `clemext_node_getParameters`.
- IBM SPSS Modeler obtains the output data model from the peer instance by means of a `clemext_peer_getDataModel` service function.
- Execution of the peer instance begins with the `clemext_peer_beginExecution` service function.
- The `clemext_peer_nextRecord` service function moves the focus to the next record in the peer's result set (or to the first record if the function is called for the first time). This is followed by the `clemext_peer_getRecordValue` service function, which returns the value of a specified field within the current record.
- The iterator callback functions `clemext_iterator_nextRecord` and `clemext_iterator_getRecordValue` can be called by the CLEF module to sequence through the input records and return specified field values.
- Execution of the peer instance ends with a call to the `clemext_peer_endExecution` service function.
- The peer instance is removed by calling `clemext_destroy_peer`.
- Before the module is unloaded, the host invokes the service function `clemext_cleanup`.
- A module may be unloaded either when the server process shuts down or earlier when its services are no longer required.

Server-Side API Features

This section highlights some of the features of the server-side API:

- Node type information
- Data types representing different types of data storage
- Server-side shared libraries
- Filespaces and temporary files
- SQL pushback to execute SQL instructions in the database
- Exchange of data model information between IBM SPSS Modeler and the extension
- Output documents
- C++ helpers

Node Types

In the specification file, a node definition takes the following format:

```
<Node id="identifier" type="node_type" .../>
```

The `id` attribute is a string that uniquely identifies the node.

The `type` attribute identifies the node as being of one of the following types:

- Data reader
- Data writer
- Data transformer
- Model builder
- Model applier
- Document builder

See the topic "Overview of Nodes" on page 9 for more information.

The `clemext_create_peer` function includes the values of both the `id` and the `type` attributes of the `Node` element as arguments.

A single extension module may implement nodes of various types, performing various functions within each type. For example, a module may implement:

- A data reader and a data writer for a data source
- Model builders and model appliers for various modelling algorithms
- Document builders for various graph types

Data and Storage Types

A peer instance obtains input data by calling `clemext_iterator_getRecordValue` on the iterator supplied to it at the start of execution and supplies output data in response to a `clemext_peer_getRecordValue` request from the host. Data is transferred in binary format in memory, and the peer and the host must agree on the data type.

The binary data type is determined by the data model and is related to the storage attribute of a field.

The following table lists possible storage types along with the data types used to represent them.

Table 49. Storage types

Storage type	Represented by	Notes
string	`char *`	Character strings are always UTF-8 encoded
real	`CLEMEXTReal`	Double precision floating point number
integer	`CLEMEXTInteger`	64-bit signed integer
date	`CLEMEXTDate`	64-bit signed integer representing number of days since 1st January 1970
time	`CLEMEXTTime`	64-bit signed integer representing number of seconds since midnight
timestamp	`CLEMEXTTimestamp`	64-bit signed integer representing number of seconds since midnight on 1st January 1970
unknown	-	Denotes an unknown data type—values cannot be represented

Libraries

A server-side shared library can be declared in the specification file to support node execution. The shared library path is used to locate a shared library that is dynamically loaded into the host process. The shared library must define all of the required API functions. See the topic “Shared Libraries” on page 33 for more information.

If a module name is supplied in the specification file (in the `Execution` section of the node definition), the name is passed to the `nodeId` parameter of the service function `clemext_create_peer` to create a peer object. In this way, the extension can create a peer module of the appropriate kind. The `nodeType` parameter value may also influence the kind of peer that is created. The module name can be blank because a shared library may not implement more than one module of each type.

Dependent libraries may be required by a shared library that implements an extension module. These should be located in the same directory as the extension shared library.

Temporary Files

The client specification file and the server extension module can specify pathnames relative to a **filespace**, which is a temporary private space where a peer can create files for use while it is executing. A filespace is a subdirectory of the server’s temporary directory created for a peer. It is created as required and deleted when the peer is destroyed.

The peer has complete control over the filespace while it exists. The full pathname of the filespace is included in a **node information document**. This is information in XML format that is returned as a result of executing a `clemext_node_getNodeInformation` callback function. See the topic "Node Information Document" on page 182 for more information.

SQL Pushback

Where an IBM SPSS Modeler stream reads data from a SQL database and performs processing on the data, advanced users can improve the efficiency of this operation by pushing back the SQL instructions to execute in the database itself.

Several standard IBM SPSS Modeler nodes support SQL pushback, and the server-side API includes function calls to make this possible for CLEF nodes as well.

The `clemext_peer_getSQLGeneration` service function generates SQL from a peer instance and is used to push back SQL execution to the database. For a data reader node, the generated SQL must be sufficient on its own to create the peer result set. For any other type of node, the generated SQL will most likely depend on the SQL generated for upstream nodes that provide input to the peer. A peer can obtain the upstream SQL by calling the `clemext_node_getSQLGeneration` callback function on its associated node handle.

Data Model Handling

Some of the server-side API calls relate to the exchange of data model information between IBM SPSS Modeler and the extension module:

- `clemext_node_getDataModel` gets the input data model for a node
- `clemext_peer_getDataModel` gets the output data model from a peer instance
- `clemext_node_getOutputDataModel` gets the output data model for a node

Other calls relate to the methods for passing data into and out of the module. The data model determines the index used to look up field values in the following functions, which return the value of a specified field within the last-fetched input record:

- `clemext_peer_getRecordValue`
- `clemext_iterator_getRecordValue`

IBM SPSS Modeler calls `clemext_node_getDataModel` to get information on the fields of the input data model. The information is returned in XML format—for example:

```
<DataModel>
     <Fields>
          <Field name="abc" storage="string" type="set" />
          <Field name="uvw" storage="integer" type="range" />
          <Field name="xyz" storage="real" type="range" />
     <Fields>
</DataModel>
```

A module can use this information to provide the field index when retrieving values from an input record using the `clemext_iterator_getRecordValue` function.

The way in which the module affects the input data model is controlled by the value of the `mode` attribute of the `OutputDataModel` element in the specification file. The module can:

- Extend the model by adding new fields to it.
- Modify the model by removing or renaming existing fields.
- Replace the existing model with new fields.
- Leave the model unchanged.

The following examples illustrate extending and replacing the model.

Example—Extending the Input Data Model: This is the simplest case: enabling a module to add new fields and set their values but not to remove or change the values of existing fields.

Assume that the specification file contains the following instructions in the node definition:

```
<OutputDataModel mode="extend">
     <AddField name="field1" storage="string" ... />
     <AddField name="field2" storage="real" ... />
     ...
</OutputDataModel>
```

Here, the output data model is defined as comprising all of the fields in the input data model plus the two extra fields specified in the `OutputDataModel` element. Thus, the output data model consists of five fields.

The `clemext_peer_getDataModel` function returns information only on the added fields, for example:

```
<DataModel>
     <Fields>
          <Field name="field1" storage="string" ... />
          <Field name="field2" storage="real" ... />
     <Fields>
</DataModel>
```

The returned information must match the type and number values (but not the name) of the `<AddField>` elements in the specification file.

A module can use the callback function `clemext_node_getOutputDataModel` to get the details of the fields that IBM SPSS Modeler expects will be added. This information can be passed straight back to IBM SPSS Modeler in response to a call to `clemext_peer_getDataModel`. Doing so is useful in situations where the specification file logic for creating and naming output fields is complicated.

The module provides the new values for each output record when IBM SPSS Modeler calls `clemext_peer_getRecordValue`. The field indices for the new fields start after the last index for the input fields. In this example, the input data model contains three fields (at index positions 0, 1, and 2), so the two output fields are assigned field indices 3 and 4. IBM SPSS Modeler will not call `clemext_peer_getRecordValue` with field indices corresponding to the input fields because the module cannot change these fields.

Example—Replacing the Input Data Model (1): In this example, the extension module discards all input data model fields from its output, replacing them with new fields.

The specification file contains the following:

```
<OutputDataModel mode="modify">
     <AddField name="key" storage="integer" ... />
     <AddField name="field1" storage="real" ... />
     <AddField name="field2" storage="real" ... />
     ...
</OutputDataModel>
```

This time, the XML data returned by a call to `clemext_peer_getDataModel` describes all of the fields in the output data model:

```
<DataModel>
     <Fields>
          <Field name="key" storage="integer" ... />
          <Field name="field1" storage="real" ... />
          <Field name="field2" storage="real" ... />
     <Fields>
</DataModel>
```

The field indices used in calls to clemext_peer_getRecordValue start at 0 for the first output field (key), 1 for the next field (field1), and so on.

Example—Replacing the Input Data Model (2): In this example, the output data model provided by the extension also replaces the input data model, as in the previous example. However, in this case, the output data model is not defined in the specification file but is instead computed at run time by the extension module on the server. The specification file includes the following:

```
<OutputDataModel mode="modify" method="sharedLibrary" libraryId="myLibraryId" />
```

To compute the output data model, IBM SPSS Modeler first calls clemext_peer_configure, followed by clemext_peer_getDataModel. As in the previous example, none of the fields in the input data model are automatically included in the output data model, which is completely defined by the response from clemext_peer_getDataModel.

Note: In cases like this, where the extension module is defining the output data model on the server, the module cannot use clemext_node_getOutputDataModel to obtain the output data model because this would result in an "invalid operation" error.

XML Output Documents

Some service and callback functions transfer information between the host and the extension module in the form of XML output documents. A number of different documents are available, and are shown in the following table.

Table 50. XML output documents

XML output document	Notes	Returned by call to...
Catalog document	Contains the list of values for a control associated with a catalog.	clemext_peer_getCatalogInformation
Data Model document	Describes the set of fields coming into or out of a node	clemext_peer_getDataModel clemext_node_getDataModel clemext_node_getOutputDataModel
Error Detail document	Provides information about an error or other condition	clemext_peer_getErrorDetail
Execution Requirements document	Describes the execution support, such as a data cache or mandatory input fields, required by a peer instance	clemext_peer_getExecutionRequirements
Host Information document	Provides information on the host environment, including: product identifier, description, version, provider, copyright, and platform details	clemext_host_getHostInformation
Module Information document	Provides information on the extension module, including: module identifier, description, version, provider, copyright, and licence details	clemext_getModuleInformation
Node Information document	Provides information on the node associated with a peer instance, including: node identifier, type, and filespace details	clemext_node_getNodeInformation
Parameters document	Contains configuration parameters from the client node; content is determined by the extension	clemext_node_getParameters

Table 50. XML output documents (continued)

XML output document	Notes	Returned by call to...
SQL Generation document	Describes how execution of a peer can be translated to SQL	`clemext_peer_getSQLGeneration` `clemext_node_getSQLGeneration`
Status Detail document	Provides supplementary information about progress and warnings or other conditions arising during execution	`clemext_progress_report`

Catalog Document: A Catalog document describes the contents of a catalog, which contains a list of values that can be displayed from a UI control.

The CLEF module implements a call to `getCatalogInformation` as follows:

```
CLEMEXTStatus
getCatalogInformation(
          const char *catalogId,
            char* buffer,
            size_t buffer_size,
            size_t* data_size,
            CLEMEXTErrorCode* errorCode) {
      ...
}
```

where `catalogId` is the identifier of a particular catalog, as defined in the `Catalog` element in the specification file.

This function returns the Catalog document.

Example

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<CatalogInformation>
     <CatalogEntry>
          <CatalogValue>apples</CatalogValue>
          <CatalogValue>0</CatalogValue>
     </CatalogEntry>
     <CatalogEntry>
          <CatalogValue>oranges</CatalogValue>
          <CatalogValue>1</CatalogValue>
     </CatalogEntry>
     <CatalogEntry>
          <CatalogValue>bananas</CatalogValue>
          <CatalogValue>2</CatalogValue>
     </CatalogEntry>
</CatalogInformation>
```

Data Model Document: A Data Model document describes the **data model**—the set of fields with their names, types, and related information—coming into or out of a node. It encapsulates the information available at a type node.

A peer that does not consume input (a source node) has an empty input model, and a peer that does not produce output (a terminal node) has an empty output model. A peer that consumes input and produces output (a process node) must know how to compute its output model from its input.

A peer can obtain its input data model by calling `clemext_node_getDataModel` on the associated node handle. A peer provides its output data model in response to a `clemext_peer_getDataModel` request from the host.

Any data model can be expressed directly as a data dictionary that enumerates all of the fields in the data model together with their properties. An input data model provided by a node to a peer is always of this form. An output data model produced by a peer may have the same form or may instead be expressed as a sequence of operations (add field, remove field, modify field) applied to the input model. This significantly simplifies the output model for some nodes.

The order of fields in a Data Model document is significant and determines the order in which data will be presented in the corresponding input or output dataset.

A data model may be incomplete and provide only a partial specification of the data. An input model that is sufficiently specified to allow a peer to compute an execution plan is said to be **executable** for that peer. An executable data model must always contain the binary type for each field so that the input and output data can be correctly marshalled.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<DataModel>
  <Fields>
    <Field name="Age" type="range" storage="integer" direction="in">
      <Range minValue="15" maxValue="74"/>
    </Field>
    <Field name="Sex" type="flag" storage="string">
      <Values>
        <Value value="F" flagValue="false" displayLabel="Female"/>
        <Value value="M" flagValue="true" displayLabel="Male"/>
      </Values>
    </Field>
    <Field name="BP" type="orderedSet" storage="integer">
      <Values>
        <Value value="-1" />
        <Value value="0" />
        <Value value="1" />
      </Values>
    </Field>
    <Field name="Cholesterol" type="flag" storage="string">
      <Values>
        <Value value="NORMAL" flagValue="false"/>
        <Value value="HIGH" flagValue="true"/>
      </Values>
    </Field>
    <Field name="Na" type="range" storage="real" displayLabel="Blood sodium">
      <Range minValue="0.500517" maxValue="0.899774"/>
    </Field>
    <Field name="K" type="range" storage="real" displayLabel="Potassium concentration">
      <Range minValue="0.020152" maxValue="0.079925"/>
    </Field>
    <Field name="Drug" type="set" storage="string" direction="out">
      <Values>
        <Value value="drugA"/>
        <Value value="drugB"/>
        <Value value="drugC"/>
        <Value value="drugX"/>
        <Value value="drugY"/>
      </Values>
    </Field>
  </Fields>
</DataModel>
```

Error Detail Document: An Error Detail document is used to send messages (error, warning, information) back to IBM SPSS Modeler and provides information about an error or other condition. An extension module can provide an Error Detail document to explain a module-specific error in response to a `clemext_peer_getErrorDetail` request from the host.

The error detail is a set of one or more `Diagnostic` elements where each diagnostic includes at least an error code, a message, and a set of one or more parameters containing further information to be inserted in the message. The error codes match the values in `StatusCode` elements in the specification file.

The message may have different language variants, or the client can use the error code to select a localized message from a resource bundle. A sequence of diagnostic elements describes a causal chain of errors.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<ErrorDetail>
  <Diagnostic code="123" severity="error">
    <Message>You can't do that ({0})</Message>
    <Parameter>Permission denied</Parameter>
  </Diagnostic>
  <Diagnostic code="456" severity="warning">
    <Message>That was silly!</Message>
    <Message lang="fr">Quel idiot!</Message>
  </Diagnostic>
</ErrorDetail>
```

Execution Requirements Document: An Execution Requirements document describes the execution support required by a peer instance. A peer instance can provide an Execution Requirements document in response to a `clemext_peer_getExecutionRequirements` request from the host. The host consults the requirements document before calling `clemext_peer_beginExecution` on the peer in order to provide the proper execution environment.

The host can provide a data cache service to enable the module to make multiple passes over input data using the `clemext_iterator_rewind` function.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<ExecutionRequirements>
  <Cache/><!-- this ensures that the CLEF module can make multiple passes over the input
    data -->
</ExecutionRequirements>
```

Host Information Document: A Host Information document describes the host environment. An extension module can obtain host information by calling `clemext_host_getHostInformation` on the host handle.

The information returned includes details of the product identifier, description, version, provider, copyright, and platform.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<HostInformation>
  <Host name="clemlocal" externalEncoding="cp1252" language="english_us"
        locale="English_United Kingdom.1252" provider="IBM Corp." version="16" platform=
        "Windows XP SP2" copyright="Copyright 1995-2011 IBM Corp. All rights reserved.">
    <VersionDetail major="12" minor="0"/>
```

```
    <PlatformDetail osType="windows" osName="WindowsNT" osMajor="5" osMinor="1"/>
    <LibraryDetail path="C:\Program Files\IBM\SPSS\Modeler\16\ext\bin\my.module\myModule.dll"/>
  </Host>
</HostInformation>
```

Module Information Document: A Module Information document describes an extension module. An extension module must provide a Module Information document in response to a `clemext_getModuleInformation` request from the host.

The information returned includes details of the module identifier, description, version, provider, copyright, and licence.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<ModuleInformation>
  <Module name="MyModule" provider="My Company Inc." version="10.1.0.329"
          copyright="Copyright 2006 My Company Inc. All rights reserved.">
    <VersionDetail major="10" minor="1" release="0" build="329"/>
    <Licence code="1234" type="mandatory"/>
    <Description>Provides a thorough test of the new extensions framework.</Description>
  </Module>
</ModuleInformation>
```

Node Information Document: A Node Information document describes the node associated with a peer instance. A peer instance can obtain node information by calling `clemext_node_getNodeInformation` on a node handle. Node information includes details of the node identifier, type, and filespace.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<NodeInformation>
  <Node name="databaseImport" type="dataReader">
    <FileSpace path="C:\Program Files\IBM SPSS Modeler Server
16\tmp\ext-8005-6711-01"/>
  </Node>
</NodeInformation>
```

Parameters Document: A Parameters document contains details of each `Property` element defined in the specification file. The details are returned in the form of configuration parameters, which a peer can obtain by calling `clemext_node_getParameters` on the node handle.

A parameter has a name and a value, where the value can be a:

- Simple value (string)
- Keyed value (key and value)
- Structured value (set of named values)
- List of values

The content of the Parameters document is determined entirely by the extension package. Parameters are defined in the client specification file and interpreted by the server extension module.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<Parameters>
  <Parameter name="linesToScan" value="50"/>
  <Parameter name="useCaption" value="true"/>
  <Parameter name="caption" value="My Caption"/>
  <Parameter name="captionPosition" value="north"/>
```

```
  <Parameter name="defaultAggregation">
    <ListValue>
      <Value value="min"/>
      <Value value="max"/>
      <Value value="mean"/>
      <Value value="stddev"/>
    </ListValue>
  </Parameter>
</Parameters>
```

SQL Generation Document: A SQL Generation document describes how execution of a peer can be translated to SQL.

A peer can provide a SQL Generation document in response to a `clemext_peer_getSQLGeneration` request from the host. The host will try to execute the SQL in preference to executing the peer internally.

A peer that consumes input can obtain its input SQL by calling `clemext_node_getSQLGeneration` on the associated node handle.

The main component of a SQL Generation document is a SQL statement that replicates the execution behavior of a node or stream fragment. For a node that produces data (that is, a data reader or data transformer node), the statement must be a `SELECT` statement and must be accompanied by a dictionary that maps field names in the data model to the column names in the `SELECT` statement.

A SQL Generation document may also include the properties of the database connection against which the statement will be executed, such as the data source name and product name. A peer can use these properties to help determine the SQL that it produces.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<SqlGeneration>
  <Properties
    datasourceName="SQL Server"
    databaseName="DataMining"
    serverName="GB1-RDUNCAN1"
    passwordKey="PW0"
    userName="fred"
    dbmsName="Microsoft SQL Server"
    dbmsVersion="09.00.1399"/>
  <Statement>
    <Bindings>
      <Binding columnName="C0" fieldName="ID"/>
      <Binding columnName="C1" fieldName="START_DATE"/>
    </Bindings>
    <TableParameters>
 <TableParameter name="${TABLE26}" value="dbo.DRUG4N"/>
    </TableParameters>
    <Sql>
      SELECT
      T0.ID AS C0,T0."START_DATE" AS C1
      FROM ${TABLE26} T0
      WHERE (T0."START_DATE" > '2003-01-01')
      ORDER BY 2 ASC
    </Sql>
  </Statement>
</SqlGeneration>
```

Status Detail Document: A Status Detail document provides information about progress and nonfatal warnings or other conditions arising during execution. An extension module can dispatch Status Detail documents asynchronously using the `clemext_progress_report` callback function.

The Status Detail document consists of a set of one or more `Diagnostic` elements where each diagnostic includes at least a condition code, a message (unless supplied in a properties file), and a set of one or more parameters containing further information to be inserted in the message. The `StatusDetail` element can also have an optional `destination` attribute to direct the message to be passed to one of the following:

- A local trace file managed by IBM SPSS Modeler
- The client (for messages of interest to the user)
- All (send to all possible destinations)

The format of the `Diagnostic` element is:

```
<Diagnostic code="integer" severity="severity_level">
     <Message>message_text</Message>
     <Parameter>value</Parameter>
</Diagnostic>
```

where:

`code` (required) is an integer denoting the condition code.

`severity` indicates the severity of the condition, and is one of: `unknown`, `information`, `warning`, `error` or `fatal`.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<StatusDetail destination="client">
  <Diagnostic code="654" severity="information">
    <Message>Processed {0} records</Message>
    <Parameter>10000</Parameter>
  </Diagnostic>
</StatusDetail>
```

Using Localized Messages

If you want to use localized messages from a properties file, you omit the `Message` element from the Status Detail document, and use message keys in the specification file, as in the following example:

```
...
<Execution ...>
...
    <StatusCodes>
        ...
        <StatusCode code="21" status="warning" messageKey="fieldIgnoredMsg.LABEL"/>
        ...
    </StatusCodes>
</Execution>
...
```

The messages.properties file would then contain:

```
fieldIgnoredMsg.LABEL=Field "{0}" cannot be used for model building and was ignored
```

In the Status Detail document, you could then send a parameter (such as a field name) to be substituted into the localized message, for example:

```
<?xml version="1.0" encoding="utf-8"?>
<StatusDetail>
    <Diagnostic code="21">
        <Parameter>BP</Parameter>
    </Diagnostic>
</StatusDetail>
```

C++ Helpers

Some of the CLEF example nodes include a number of predefined C++ source files, known as **helpers**. These act as wrappers for some of the C-based server-side API and can be easily compiled into a C++ CLEF.

Table 51. C++ helpers

Helper	Description
BufferHelper	Manages resizable memory buffers used in the C API
DataHelper	Helps to wrap data read and write operations
HostHelper	Wraps the CLEMEXT HostHandle object
XMLHelper	Wraps XML processing

The helpers take the form of a pair of *.cpp* and *.h* files—for example, *BufferHelper.cpp* and *BufferHelper.h*.

For information on viewing these helper files, see "Examining the Source Code" on page 26.

More detailed descriptions of these files can be found in the server-side API documentation, as follows:

1. From the CLEF API Documentation screen, choose **Server-side API overview**.
2. Click the **Files** tab.
3. Click the name of the `.h` file corresponding to the helper for which you want information.
4. Under **Data Structures**, click the corresponding class name to display the documentation.

For information on accessing the CLEF API documentation, see "CLEF API Documentation" on page 167.

Error Handling

Each API function call returns a status code (`CLEMEXTStatus`) and an optional module-specific error code (`CLEMEXTErrorCode`). The status code can be `success` (no error) or one of the error codes enumerated for the API function; these almost always include "module-specific error." The module-specific error code can be 0 to mean "no module-specific error."

Status code messages are supplied by IBM SPSS Modeler. Details of common status codes can be found in the server-side API documentation, as follows:

1. From the CLEF API Documentation screen, choose **Server-side API overview**.
2. Click the **Modules** tab.
3. Select **Common Status Codes**.

For information on accessing the CLEF API documentation, see "CLEF API Documentation" on page 167.

Module-specific error messages can be supplied:

- In the specification file (in the `StatusCodes` element of the Module section)
- In a resource bundle referenced in the specification file
- By the extension module

For a module-specific error code, a module can provide extra error detail consisting of a default error message (for errors not accounted for in the specification file or resource bundle) and parameters to be inserted in the message. Multiple error messages can describe causal error chains.

An error report on the client has the following format:

node_label:*message*

where:

- *node_label* is the value of the `label` attribute of the `Node` element in which the module is specified.
- *message* is the text of the message, which can either be supplied by the server or defined in the specification file (or in a *.properties* file for localization).

XML Parsing API

IBM SPSS Modeler includes the Xerces-C XML parser from Apache, providing a number of callbacks that allow a module to read and write XML data. You can substitute your own XML parser if preferred.

Using the Server-Side API

To include server-side function calls in a node:

1. Create the C++ *.cpp* and *.h* source files that include the function calls.
2. Compile the source files into a dynamic link library (*.dll*) file.
3. Include a reference to the *.dll* file from the specification file—for example:

```
<Resources>
.
     <SharedLibrary id="mylib1" path="mycorp.mynode/mylib" />
.
</Resources>
```

See the topic "Shared Libraries" on page 33 for more information.

You might also find it useful to take a look at the source code for the example nodes that are supplied with this release. See the topic "Examining the Source Code" on page 26 for more information.

Server-Side Programming Guidelines

The server-side dynamic link library (DLL) part of a CLEF module should follow a number of guidelines to ensure that the module functions correctly, and to avoid affecting the operation of IBM SPSS Modeler. A CLEF module should:

- ensure that peer execution is self-contained
- support multiple peer instances in a single process
- ensure thread safety
- avoid altering the thread or process environment
- restrict thread usage within the module
- correctly handle requests to cancel execution
- restart interrupted system calls (UNIX)
- take care when calling CoInitialize or CoUninitialize (Windows)
- avoid making assumptions about when the module will be unloaded
- take care when starting sub-processes
- avoid writing to standard output or standard error

The following sections cover each of these areas in more detail.

Ensure that Peer Execution is Self Contained

Peer instances should not make any assumptions about the existence of other peer instances within the IBM SPSS Modeler server process. IBM SPSS Modeler may schedule execution so that peer instances corresponding to nodes that are directly adjacent in a stream are actually executed in different phases, so that existence and execution of the instances does not overlap.

Peer instances should therefore be self contained and not try to communicate with other peer instances directly, for example through pipes or sockets. All communication between different peer instances should be performed either directly, by reading and writing data to the stream, or indirectly, through some external agent (for example a database server that manages data shared between peers).

Support Multiple Peer Instances in a Single Process

End users may create multiple peer instances of a particular CLEF module (i.e., more than one node of the same type) in a server process when executing a stream. Thus any static data in the CLEF module is shared across the multiple peer instances, and should not be used to store data that is private to a peer object. Examples of static data are static members of C++ classes, and global or static variables in C compilation units.

CLEF module API functions must be re-entrant and avoid making any non re-entrant system calls. For example, when a peer instance fetches input data from its input iterator using `clemext_iterator_nextRecord`, it is possible for this in turn to call `clemext_peer_nextRecord` on a second peer instance located upstream from the first peer and which produces the data that is eventually used by the first peer.

System calls like `strtok` are not re-entrant and should not be used. For details of alternatives that are re-entrant, consult the operating system documentation for the platform you are using.

Ensure Thread Safety

IBM SPSS Modeler may interleave the execution of multiple peer instances from different execution threads. Access to any resources shared between peer objects should therefore be protected, for example by synchronization with mutexes (mutual exclusion objects) or similar threading library services.

CLEF modules must avoid making any system calls that are not thread-safe. For more information, consult your operating system documentation or UNIX `man` pages.

Avoid Altering the Thread or Process Environment

Avoid using system calls that might change the environment of the calling thread or process.

Some examples of such calls are listed here - the list is by no means exhaustive:

- `setlocale` when used to change locale rather than read locale information
- `SetCurrentDirectory` (Windows) or `chdir` (UNIX)
- `LogonUser` (Windows) or `seteuid` (UNIX)
- `putenv`
- `exit`
- `signal`

Note: On Windows, one call that changes the environment of a thread but which may be necessary is `CoInitialize`. See the topic "Take Care when Calling CoInitialize or CoUninitialize (Windows)" on page 188 for more information.

Restrict Thread Usage Within the Module

In general, modules are free to use threading internally. However, callback functions should be invoked only on the thread that IBM SPSS Modeler used to call the CLEF module functions (except for `clemext_peer_cancelExecution`).

The following callback functions can be called asynchronously from any thread executing within the module:

- clemext_progress_report
- clemext_channel_send

A peer instance should ensure that multiple threads do not invoke each of these calls simultaneously.

Correctly Handle Requests to Cancel Execution

When an end user requests cancelling execution of a peer instance, IBM SPSS Modeler makes an asynchronous call to the clemext_peer_cancelExecution function of the module. Developers should try to implement this call. Note that it is intended for this function to be called asynchronously, and called while another CLEF API function call is being executed by the module.

Restart Interrupted System Calls (UNIX)

On UNIX, the IBM SPSS Modeler application uses signals and signal handlers. Some UNIX system calls may return code EINTR if the process receives a signal during the execution of the call--check the man pages for the system call on your particular UNIX platform.

If this event occurs, the calling code should check for the EINTR return code and restart the call. One way to achieve this is to create a simple wrapper function (open_safe) and have your application call that wrapper:

```
int
open_safe(const char* path, int oflag, mode_t mode) {
  int res;
  while ((res = ::open(path, oflag, mode)) == -1
                                     && errno == EINTR) {
  }
  return res;
}
```

Take Care when Calling CoInitialize or CoUninitialize (Windows)

On Windows, threads that need to use the Windows Component Object Model (COM) library services should call the system API function CoInitialize before using COM services, and should call CoUninitialize on completion. The threads from which IBM SPSS Modeler invokes the CLEF API for a module may or may not already have called CoInitialize.

A CLEF module wishing to use COM services from these threads should call CoInitialize, usually in a clemext_create_peer or clemext_peer_beginExecution function. If that call succeeds, the module must also call CoUninitialize at a later time when the thread completes execution, usually in clemext_destroy_peer or clemext_peer_endExecution respectively.

For more information on the CoInitiaize call, see the documentation on the Microsoft Developer Network (MSDN) site at http://msdn.microsoft.com.

Avoid Making Assumptions About When the Module Will be Unloaded

Currently, a CLEF module remains loaded until a session ends (i.e., modules cannot be unloaded and reloaded on demand). The function clemext_cleanup is never called, even on exit from the IBM SPSS Modeler server process into which the module is loaded. Thus developers should not assume that a module will be unloaded, and its resources freed up, at any point.

Take Care When Starting Sub-Processes

Starting a sub-process, by means of CreateProcess (Windows) or fork (UNIX), can introduce a number of complications in the way that the parent and child processes interact, and in the way that the child process inherits resources that are open in the parent.

If a CLEF module needs to invoke out-of-process execution, consider using a suitable alternative architecture. For example, the CLEF module might use the services offered by an application server to perform the required task.

In particular, Windows processes should avoid starting sub-processes using the `CreateProcess` function with the `bInheritHandles` parameter set to TRUE. Doing so causes the child process to inherit all file descriptors that are open in the parent (IBM SPSS Modeler server) process.

Avoid Writing to Standard Output or Standard Error

If a CLEF module writes to the standard output or standard error stream of a process (perhaps for debugging purposes), this will generally not be visible to an end user. However, when a stream containing CLEF nodes is deployed using IBM SPSS Modeler Solution Publisher and executed from a command line shell (either in Windows or UNIX), this output will become visible and may confuse users.

CLEF modules can instead invoke a tracing service by calling the host callback function `clemext_host_trace` and passing the message to display in string form. Tracing also needs to be enabled in the IBM SPSS Modeler installation by using the following setting in the IBM SPSS Modeler Server configuration options file (*/config/options.cfg* in the IBM SPSS Modeler installation directory):

```
trace_extension, 1
```

Traced messages are output to the file */log/trace-<process_ID>-<process_ID>.log* in the IBM SPSS Modeler installation directory, where *process_ID* is the identifier of the IBM SPSS Modeler Server process. Avoid tracing multiple sessions concurrently as they all share the same log file.

Chapter 10. Testing and Distribution

Testing CLEF Extensions

We strongly recommend that you test a new extension locally before distributing it to other users.

After creating a specification file and any associated resource bundles, .jar files, shared libraries and user help files, you can test the extension by arranging the files in the required file structure and copying them to your local IBM SPSS Modeler installation. The next time you start IBM SPSS Modeler, you should see the new extension in the IBM SPSS Modeler user interface.

Testing a CLEF Extension

1. Close IBM SPSS Modeler if it is open.
2. If the extension defines a CLEF node or output, we recommend activating the Debug tab of the extension dialog until you are satisfied that the extension is working correctly. See the topic "Using the Debug Tab" on page 192 for more information.
3. Arrange the client-side and server-side files in the required structure. Make sure that the specification file, and any associated resources that the node needs (such as `.jar` or `.dll` files), are copied to the correct locations. See the topic "File Structure" on page 5 for more information.
4. Copy the client-side directory to the *\ext\lib* folder of the IBM SPSS Modeler installation directory.
5. Copy the server-side directory to the *\ext\bin* folder of the IBM SPSS Modeler installation directory.
6. Start IBM SPSS Modeler.
7. If the extension defines a menu or menu item, ensure that these appear correctly on the main menu system. If the extension defines a new node, ensure that the node appears in the desired position on the correct node palette as defined in the specification file.
8. Test the extension thoroughly.

   For example, ensure that:

   - node performance does not degrade as the number of fields and records increases
   - null values are handled consistently
   - different locales are supported if required (e.g. European, Far East)
9. Once you have added an extension, you can still make changes to its specification file. However, any changes you make do not take effect until you restart IBM SPSS Modeler.

Debugging a CLEF Extension

CLEF provides the following features to aid in debugging an extension:

- XML syntax error messages
- external execution
- Debug tab

XML Syntax Errors

Incorrect XML syntax in a specification file is flagged by an error message from the XML parser.

The message displays the approximate line number of the error, together with an indication of what has gone wrong.

To recover from this situation:

1. Correct the error in the file.
2. Retest the file by following the procedure under "Testing a CLEF Extension."

3. Repeat this procedure until no more syntax errors are found in the specification file.

External Execution

Normally, a user-written CLEF extension executes in its own process, separate from the IBM SPSS Modeler process. This can help when debugging--if an extension process fails, it does not bring down the entire IBM SPSS Modeler Server process.

Note: It is possible to override this default setting. See the topic "Changing the Execution Options" for more information.

Using the Debug Tab

For any dialog or frame associated with a CLEF node or output, you can activate a Debug tab to enable you to inspect the property settings for the object. You can also view the contents of any containers defined in the extension and save these contents to a file for further inspection. See the topic "Containers" on page 50 for more information.

You activate the Debug tab by setting to `true` the value of the `debug` attribute of the `Extension` element in the specification file. See the topic "Extension Element" on page 31 for more information.

The fields on the tab are as follows:

Element ID. The unique identifier of the extension--the value of the `id` attribute of the `ExtensionDetails` element in the specification file.

Script Name. The unique identifier of the node when referenced in a script--the value of the `scriptName` attribute of the `Node` element.

Extension ID. The name of the extension folder under which the file and directory resources for the extension reside. The value is formed by concatenating the `providerTag` and `id` attributes (separated by a "`.`" character) of the `ExtensionDetails` element. Note that for the `providerTag` portion of the identifier, the value should not include the string `spss`, which is reserved for internal use.

Properties. This table displays selected details of the `Property` declarations for the node:

- **Property**. The unique identifier of the property--the value of the `name` field of the `Property` element.
- **Script Name**. The unique identifier of the property when referenced in a script--the value of the `scriptName` attribute of the `Property` element.
- **Value Type**. The type of value that this property can take, as defined by the `valueType` attribute of the `Property` element.
- **List?** Indicates whether the property is a list of values of the specified value type--the value of the `isList` attribute of the `Property` element.
- **Shared?** If checked, indicates that this property is used in more than one place in the extension (e.g. model builder node, model output, model applier).
- **Value**. The default value, if any, of the property.

Containers. Displays the contents of the selected container (for example, model data). Click the field to display a list of any other containers defined for the extension, and select different ones to display their contents. Click the adjacent **Save Container** button to save the contents of the selected container in XML format for further inspection.

Trace. Displays a dialog that enables you to view trace output when the node is executed.

Changing the Execution Options

By default, a user-written CLEF extension module executes in a separate process from the main IBM SPSS Modeler process. In this way, a failure of the extension process does not cause the IBM SPSS Modeler process to fail. In contrast to this, IBM Corp.-provided modules execute in the main process by default.

Two server configuration options are provided to enable the system administrator to change either of these cases to the opposite for one or more named modules. Both options take a comma-separated list of module identifiers to indicate which modules are affected by the change.

Note: Changing either of these options should normally be done only on request from a Customer Support representative.

The options are as follows:

In-Process Execution Option

This allows extension modules that would normally be loaded into an external process (typically, user-written modules), to be loaded directly into IBM SPSS Modeler. The format is:

```
clef_inprocess_execution, "moduleID1[,moduleID2[,...moduleIDn]]"
```

where *moduleID* is the value of the `id` attribute of the `ExtensionDetails` element in the relevant specification file. An example follows:

```
clef_inprocess_execution, "test.example_filereader"
```

External Execution Option

This allows extension modules that would normally be loaded directly into IBM SPSS Modeler (typically, IBM Corp.-provided modules), to be loaded into an external process. The format is:

```
clef_external_execution, "moduleID1[,moduleID2[,...moduleIDn]]"
```

where *moduleID* is the same as for `clef_inprocess_execution`. A (fictitious) example follows:

```
clef_external_execution, "spss.naivebayes,spss.terminator"
```

Adding or Changing an Execution Option

To add or change an execution option, follow the procedure shown under "Using the options.cfg File" in the *IBM SPSS Modeler 16 Server Administration and Performance Guide*.

Distributing CLEF Extensions

When the new extension is fully tested and ready for distribution:

1. Deactivate the Debug tab if this was activated. See the topic “Using the Debug Tab” on page 192 for more information.
2. Create a file structure that exactly reflects the way you want the extension files to be installed. See the topic “File Structure” on page 5 for more information.
3. Compress the file structure into a `.zip` file. You may find it easier to make separate `.zip` files for the client-side and server-side installations.
4. Distribute the `.zip` files to the end users.

Installing CLEF Extensions

To install a CLEF extension:

1. On receiving a `.zip` file containing an extension file structure, extract the client-side files to the `\ext\lib` folder under the IBM SPSS Modeler installation directory.
2. Extract the server-side files to the `\ext\bin` folder under the IBM SPSS Modeler installation directory (or the equivalent if using IBM SPSS Modeler Server).
3. Start IBM SPSS Modeler and check that the new nodes appear in the expected location on the node palette.

Deinstalling CLEF Extensions

To deinstall a CLEF extension:

1. Locate the extension folder in the `\ext\lib` directory under the IBM SPSS Modeler installation directory.

   If the extension also installed a server-side extension folder, locate this folder in the `\ext\bin` directory under the IBM SPSS Modeler or IBM SPSS Modeler Server installation directory.
2. Delete the extension folder or folders.

The change takes effect the next time you start IBM SPSS Modeler.

Appendix. CLEF XML Schema

CLEF Element Reference

This section provides a reference for all the elements in CLEF.

Each topic lists the valid attributes for an element and its parent and child elements. The diagram displays all the element's children. Note that the arrows in the diagram indicate elements that may be shared among other elements. These elements are listed in the table of contents as a child of this topic ("CLEF Element Reference") rather than as a child of the parent topic.

Elements

Action Element

Table 52. Attributes for Action

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
id	**required**		*string*
imagePath	optional		*string*
imagePathKey	optional		*string*
label	**required**		*string*
labelKey	optional		*string*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
shortcut	optional		*string*
shortcutKey	optional		*string*

XML Representation

```
<xs:element name="Action">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="imagePath" type="xs:string" use="optional"/>
  <xs:attribute name="imagePathKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="shortcut" type="xs:string" use="optional"/>
  <xs:attribute name="shortcutKey" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Actions

ActionButton Element

Table 53. Attributes for ActionButton

Attribute	Use	Description	Valid Values
action	**required**		*string*
showIcon	optional		*boolean*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="ActionButton">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="action" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="showIcon" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

ComboBoxControl, ExtensionObjectPanel, ModelViewerPanel, SelectorPanel, StaticText, SystemControls, TabbedPanel, TextBrowserPanel

Actions Element

XML Representation

```
<xs:element name="Actions">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="Action"/>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Parent Elements

CommonObjects

Child Elements

Action

AddField Element

Table 54. Attributes for AddField

Attribute	Use	Description	Valid Values
direction	optional		**in** **out** **both** **none** **partition**
directionRef	optional		
fieldRef	optional		
group	optional		*string*
label	optional		*string*
missingValuesRef	optional		
name	**required**		
prefix	optional		*string*
role	optional		**unknown** **predictedValue** **predictedDisplayValue** **probability** **residual** **standardError** **entityId** **entityAffinity** **upperConfidenceLimit** **lowerConfidenceLimit** **propensity** **value** **supplementary**
storage	optional		**unknown** **integer** **real** **string** **date** **time** **timestamp**
storageRef	optional		
tag	optional		*string*
targetField	optional		*string*
type	optional		**auto** **range** **discrete** **set** **orderedSet** **flag** **typeless**
typeRef	optional		
value	optional		*string*

XML Representation

```
<xs:element name="AddField">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Range" minOccurs="0"/>
      <xs:element ref="Values" minOccurs="0"/>
      <xs:element ref="NumericInfo" minOccurs="0"/>
      <xs:element name="MissingValues" minOccurs="0">
        <xs:sequence>
          <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="Range" minOccurs="0"/>
        </xs:sequence>
      </xs:element>
      <xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
      </xs:element>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
  <xs:attribute name="storage" type="FIELD-STORAGE">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="real"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="time"/>
    <xs:enumeration value="timestamp"/>
  </xs:attribute>
  <xs:attribute name="type" type="FIELD-TYPE">
    <xs:enumeration value="auto"/>
    <xs:enumeration value="range"/>
    <xs:enumeration value="discrete"/>
    <xs:enumeration value="set"/>
    <xs:enumeration value="orderedSet"/>
    <xs:enumeration value="flag"/>
    <xs:enumeration value="typeless"/>
  </xs:attribute>
  <xs:attribute name="direction" type="FIELD-DIRECTION">
    <xs:enumeration value="in"/>
    <xs:enumeration value="out"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
    <xs:enumeration value="partition"/>
  </xs:attribute>
  <xs:attribute name="label" type="xs:string"/>
  <xs:attribute name="fieldRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="storageRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="typeRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="directionRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="missingValuesRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="role" type="MODEL-FIELD-ROLE" use="optional">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="predictedValue"/>
    <xs:enumeration value="predictedDisplayValue"/>
    <xs:enumeration value="probability"/>
    <xs:enumeration value="residual"/>
    <xs:enumeration value="standardError"/>
    <xs:enumeration value="entityId"/>
    <xs:enumeration value="entityAffinity"/>
    <xs:enumeration value="upperConfidenceLimit"/>
    <xs:enumeration value="lowerConfidenceLimit"/>
    <xs:enumeration value="propensity"/>
    <xs:enumeration value="value"/>
    <xs:enumeration value="supplementary"/>
  </xs:attribute>
  <xs:attribute name="targetField" type="xs:string" use="optional"/>
  <xs:attribute name="value" type="xs:string" use="optional"/>
  <xs:attribute name="group" type="xs:string" use="optional"/>
  <xs:attribute name="tag" type="xs:string" use="optional"/>
  <xs:attribute name="prefix" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

ForEach, ModelFields

Child Elements

MissingValues, ModelField, NumericInfo, Range, Range, Values, Values

Related Elements

ChangeField

MissingValues Element:

Table 55. Attributes for MissingValues

Attribute	Use	Description	Valid Values
treatNullAsMissing	optional		*boolean*
treatWhitespaceAsMissing	optional		*boolean*

XML Representation

```
<xs:element name="MissingValues" minOccurs="0">
  <xs:sequence>
    <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Range" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="treatWhitespaceAsMissing" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="treatNullAsMissing" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Parent Elements

Field

Child Elements

Range, Range, Values, Values

ModelField Element:

Table 56. Attributes for ModelField

Attribute	Use	Description	Valid Values
group	optional		
role	**required**		**unknown** **predictedValue** **predictedDisplayValue** **probability** **residual** **standardError** **entityId** **entityAffinity** **upperConfidenceLimit** **lowerConfidenceLimit** **propensity** **value** **supplementary**
tag	optional		*string*
targetField	optional		*string*
value	optional		*string*

XML Representation

```
<xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
  <xs:attribute name="role" type="MODEL-FIELD-ROLE" use="required">
    <xs:enumeration value="unknown"/>
```

```
    <xs:enumeration value="predictedValue"/>
    <xs:enumeration value="predictedDisplayValue"/>
    <xs:enumeration value="probability"/>
    <xs:enumeration value="residual"/>
    <xs:enumeration value="standardError"/>
    <xs:enumeration value="entityId"/>
    <xs:enumeration value="entityAffinity"/>
    <xs:enumeration value="upperConfidenceLimit"/>
    <xs:enumeration value="lowerConfidenceLimit"/>
    <xs:enumeration value="propensity"/>
    <xs:enumeration value="value"/>
    <xs:enumeration value="supplementary"/>
  </xs:attribute>
  <xs:attribute name="targetField" type="xs:string"/>
  <xs:attribute name="value" type="xs:string"/>
  <xs:attribute name="group" type="MODEL-FIELD-GROUP"/>
  <xs:attribute name="tag" type="xs:string"/>
</xs:element>
```

Parent Elements

Field

And Element

XML Representation

```
<xs:element name="And">
  <xs:sequence minOccurs="2" maxOccurs="unbounded">
    <xs:group ref="CONDITION-EXPRESSION">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Parent Elements

And, Command, Constraint, CreateDocument, CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModel, CreateModelApplier, CreateModelOutput, Enabled, Not, Option, Or, Run, Visible

Child Elements

And, Condition, Not, Or

Attribute Element

Table 57. Attributes for Attribute

Attribute	Use	Description	Valid Values
defaultValue	optional		
description	optional		*string*
descriptionKey	optional		*string*
isList	optional		*boolean*
label	**required**		*string*
labelKey	optional		*string*
name	**required**		*string*

Table 57. Attributes for Attribute (continued)

Attribute	Use	Description	Valid Values
valueType	optional		**string** **encryptedString** **integer** **double** **boolean** **date** **enum**

XML Representation

```
<xs:element name="Attribute">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="valueType" type="ATTRIBUTE-VALUE-TYPE">
    <xs:enumeration value="string"/>
    <xs:enumeration value="encryptedString"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="double"/>
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="enum"/>
  </xs:attribute>
  <xs:attribute name="defaultValue" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="isList" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Parent Elements

Catalog, Structure

BinaryFormat Element

XML Representation

```
<xs:element name="BinaryFormat"/>
```

Parent Elements

FileFormatType

Catalog Element

Table 58. Attributes for Catalog

Attribute	Use	Description	Valid Values
id	**required**		*string*
valueColumn	**required**		*integer*

XML Representation

```
<xs:element name="Catalog">
  <xs:sequence minOccurs="1" maxOccurs="unbounded">
    <xs:element ref="Attribute"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="valueColumn" type="xs:integer" use="required"/>
</xs:element>
```

Parent Elements

Catalogs

Child Elements

Attribute

Catalogs Element

XML Representation

```
<xs:element name="Catalogs">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="Catalog"/>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Parent Elements

CommonObjects

Child Elements

Catalog

ChangeField Element

Table 59. Attributes for ChangeField

Attribute	Use	Description	Valid Values
direction	optional		**in** **out** **both** **none** **partition**
directionRef	optional		
fieldRef	**required**		
label	optional		*string*
missingValuesRef	optional		
name	**required**		
storage	optional		**unknown** **integer** **real** **string** **date** **time** **timestamp**
storageRef	optional		

Table 59. Attributes for ChangeField (continued)

Attribute	Use	Description	Valid Values
type	optional		**auto** **range** **discrete** **set** **orderedSet** **flag** **typeless**
typeRef	optional		

XML Representation

```
<xs:element name="ChangeField">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Range" minOccurs="0"/>
      <xs:element ref="Values" minOccurs="0"/>
      <xs:element ref="NumericInfo" minOccurs="0"/>
      <xs:element name="MissingValues" minOccurs="0">
        <xs:sequence>
          <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="Range" minOccurs="0"/>
        </xs:sequence>
      </xs:element>
      <xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
      </xs:element>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
  <xs:attribute name="storage" type="FIELD-STORAGE">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="real"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="time"/>
    <xs:enumeration value="timestamp"/>
  </xs:attribute>
  <xs:attribute name="type" type="FIELD-TYPE">
    <xs:enumeration value="auto"/>
    <xs:enumeration value="range"/>
    <xs:enumeration value="discrete"/>
    <xs:enumeration value="set"/>
    <xs:enumeration value="orderedSet"/>
    <xs:enumeration value="flag"/>
    <xs:enumeration value="typeless"/>
  </xs:attribute>
  <xs:attribute name="direction" type="FIELD-DIRECTION">
    <xs:enumeration value="in"/>
    <xs:enumeration value="out"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
    <xs:enumeration value="partition"/>
  </xs:attribute>
  <xs:attribute name="label" type="xs:string"/>
  <xs:attribute name="fieldRef" type="EVALUATED-STRING" use="required"/>
  <xs:attribute name="storageRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="typeRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="directionRef" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="missingValuesRef" type="EVALUATED-STRING" use="optional"/>
</xs:element>
```

Parent Elements

ForEach, ModelFields

Child Elements

MissingValues, ModelField, NumericInfo, Range, Range, Values, Values

Related Elements

AddField

MissingValues Element:

Table 60. Attributes for MissingValues

Attribute	Use	Description	Valid Values
treatNullAsMissing	optional		*boolean*
treatWhitespaceAsMissing	optional		*boolean*

XML Representation

```
<xs:element name="MissingValues" minOccurs="0">
  <xs:sequence>
    <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Range" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="treatWhitespaceAsMissing" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="treatNullAsMissing" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Parent Elements

Field

Child Elements

Range, Range, Values, Values

ModelField Element:

Table 61. Attributes for ModelField

Attribute	Use	Description	Valid Values
group	optional		
role	**required**		**unknown** **predictedValue** **predictedDisplayValue** **probability** **residual** **standardError** **entityId** **entityAffinity** **upperConfidenceLimit** **lowerConfidenceLimit** **propensity** **value** **supplementary**
tag	optional		*string*
targetField	optional		*string*
value	optional		*string*

XML Representation

```
<xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
  <xs:attribute name="role" type="MODEL-FIELD-ROLE" use="required">
    <xs:enumeration value="unknown"/>
```

```
    <xs:enumeration value="predictedValue"/>
    <xs:enumeration value="predictedDisplayValue"/>
    <xs:enumeration value="probability"/>
    <xs:enumeration value="residual"/>
    <xs:enumeration value="standardError"/>
    <xs:enumeration value="entityId"/>
    <xs:enumeration value="entityAffinity"/>
    <xs:enumeration value="upperConfidenceLimit"/>
    <xs:enumeration value="lowerConfidenceLimit"/>
    <xs:enumeration value="propensity"/>
    <xs:enumeration value="value"/>
    <xs:enumeration value="supplementary"/>
  </xs:attribute>
  <xs:attribute name="targetField" type="xs:string"/>
  <xs:attribute name="value" type="xs:string"/>
  <xs:attribute name="group" type="MODEL-FIELD-GROUP"/>
  <xs:attribute name="tag" type="xs:string"/>
</xs:element>
```

Parent Elements

Field

CheckBoxControl Element

Table 62. Attributes for CheckBoxControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
invert	optional		*boolean*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="CheckBoxControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="invert" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

CheckBoxGroupControl Element

Table 63. Attributes for CheckBoxGroupControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
layoutByRow	optional		*boolean*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
rows	optional		*positiveInteger*
showLabel	optional		*boolean*
useSubPanel	optional		*boolean*

XML Representation

```
<xs:element name="CheckBoxGroupControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="rows" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="layoutByRow" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="useSubPanel" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

ClientDirectoryChooserControl Element

Table 64. Attributes for ClientDirectoryChooserControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
mode	**required**		**open** **save** **import** **export**
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="ClientDirectoryChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="mode" type="FILE-CHOOSER-MODE" use="required">
    <xs:enumeration value="open"/>
    <xs:enumeration value="save"/>
    <xs:enumeration value="import"/>
    <xs:enumeration value="export"/>
  </xs:attribute>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

ClientFileChooserControl Element

Table 65. Attributes for ClientFileChooserControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
mode	**required**		**open** **save** **import** **export**
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="ClientFileChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="mode" type="FILE-CHOOSER-MODE" use="required">
    <xs:enumeration value="open"/>
    <xs:enumeration value="save"/>
```

```
    <xs:enumeration value="import"/>
    <xs:enumeration value="export"/>
  </xs:attribute>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

ComboBoxControl Element

Table 66. Attributes for ComboBoxControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="ComboBoxControl" type="CONTROLLER">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
</xs:element>
```

Table 67. Extended Types

Type	Description
ItemChooserControl	

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

ActionButton, ExtensionObjectPanel, ModelViewerPanel, SelectorPanel, StaticText, SystemControls, TabbedPanel, TextBrowserPanel

Command Element

Table 68. Attributes for Command

Attribute	Use	Description	Valid Values
path	**required**		

XML Representation

```
<xs:element name="Command">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
  <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Parent Elements

Run

Child Elements

And, Condition, Not, Or

CommonObjects Element

Provides a location for definitions that are global to the extension

Table 69. Attributes for CommonObjects

Attribute	Use	Description	Valid Values
extensionListenerClass	optional		*string*

XML Representation

```
<xs:element name="CommonObjects">
  <xs:all>
    <xs:element ref="PropertyTypes" minOccurs="0"/>
```

```
    <xs:element ref="PropertySets" minOccurs="0"/>
    <xs:element ref="FileFormatTypes" minOccurs="0"/>
    <xs:element ref="ContainerTypes" minOccurs="0"/>
    <xs:element ref="Actions" minOccurs="0"/>
    <xs:element ref="Catalogs" minOccurs="0"/>
  </xs:all>
  <xs:attribute name="extensionListenerClass" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Extension

Child Elements

Actions, Catalogs, ContainerTypes, FileFormatTypes, PropertySets, PropertyTypes

Condition Element

Table 70. Attributes for Condition

Attribute	Use	Description	Valid Values
container	optional		*string*
control	optional		*string*
expression	optional		
listMode	optional		**all** **any**

Table 70. Attributes for Condition (continued)

Attribute	Use	Description	Valid Values
op	**required**		**equals** **notEquals** **isEmpty** **isNotEmpty** **lessThan** **lessOrEquals** **greaterThan** **greaterOrEquals** **equalsIgnoreCase** **isSubstring** **startsWith** **endsWith** **startsWithIgnoreCase** **endsWithIgnoreCase** **isSubstring** **hasSubstring** **isSubstringIgnoreCase** **hasSubstringIgnoreCase** **in** **countEquals** **countLessThan** **countLessOrEquals** **countGreaterThan** **countGreaterOrEquals** **contains** **storageEquals** **typeEquals** **directionEquals** **isMeasureDiscrete** **isMeasureContinuous** **isMeasureTypeless** **isMeasureUnknown** **isStorageString** **isStorageNumeric** **isStorageDatetime** **isStorageUnknown** **isModelOutput** **modelOutputRoleEquals** **modelOutputTargetField Equals** **modelOutputHasValue** **modelOutputTagEquals** **enumRestriction**
property	optional		*string*
value	optional		

XML Representation

```
<xs:element name="Condition">
  <xs:attribute name="expression" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="control" type="xs:string" use="optional"/>
  <xs:attribute name="property" type="xs:string" use="optional"/>
  <xs:attribute name="container" type="xs:string" use="optional"/>
  <xs:attribute name="op" type="CONDITION-TEST" use="required">
    <xs:enumeration value="equals"/>
    <xs:enumeration value="notEquals"/>
    <xs:enumeration value="isEmpty"/>
    <xs:enumeration value="isNotEmpty"/>
```

```
    <xs:enumeration value="lessThan"/>
    <xs:enumeration value="lessOrEquals"/>
    <xs:enumeration value="greaterThan"/>
    <xs:enumeration value="greaterOrEquals"/>
    <xs:enumeration value="equalsIgnoreCase"/>
    <xs:enumeration value="isSubstring"/>
    <xs:enumeration value="startsWith"/>
    <xs:enumeration value="endsWith"/>
    <xs:enumeration value="startsWithIgnoreCase"/>
    <xs:enumeration value="endsWithIgnoreCase"/>
    <xs:enumeration value="isSubstring"/>
    <xs:enumeration value="hasSubstring"/>
    <xs:enumeration value="isSubstringIgnoreCase"/>
    <xs:enumeration value="hasSubstringIgnoreCase"/>
    <xs:enumeration value="in"/>
    <xs:enumeration value="countEquals"/>
    <xs:enumeration value="countLessThan"/>
    <xs:enumeration value="countLessOrEquals"/>
    <xs:enumeration value="countGreaterThan"/>
    <xs:enumeration value="countGreaterOrEquals"/>
    <xs:enumeration value="contains"/>
    <xs:enumeration value="storageEquals"/>
    <xs:enumeration value="typeEquals"/>
    <xs:enumeration value="directionEquals"/>
    <xs:enumeration value="isMeasureDiscrete"/>
    <xs:enumeration value="isMeasureContinuous"/>
    <xs:enumeration value="isMeasureTypeless"/>
    <xs:enumeration value="isMeasureUnknown"/>
    <xs:enumeration value="isStorageString"/>
    <xs:enumeration value="isStorageNumeric"/>
    <xs:enumeration value="isStorageDatetime"/>
    <xs:enumeration value="isStorageUnknown"/>
    <xs:enumeration value="isModelOutput"/>
    <xs:enumeration value="modelOutputRoleEquals"/>
    <xs:enumeration value="modelOutputTargetFieldEquals"/>
    <xs:enumeration value="modelOutputHasValue"/>
    <xs:enumeration value="modelOutputTagEquals"/>
    <xs:enumeration value="enumRestriction"/>
  </xs:attribute>
  <xs:attribute name="value" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="listMode" use="optional" default="all">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="all"/>
        <xs:enumeration value="any"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>
```

Parent Elements

And, Command, Constraint, CreateDocument, CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModel, CreateModelApplier, CreateModelOutput, Enabled, ExpertSettings, Not, Option, Or, Run, Visible

Constraint Element

Table 71. Attributes for Constraint

Attribute	Use	Description	Valid Values
property	**required**		*string*
singleSelection	optional		*boolean*

XML Representation

```
<xs:element name="Constraint">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
```

```
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="singleSelection" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Parent Elements

AutoModeling

Child Elements

And, Condition, Not, Or

Constructors Element

XML Representation

```
<xs:element name="Constructors">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="CreateModelOutput"/>
      <xs:element ref="CreateDocumentOutput"/>
      <xs:element ref="CreateInteractiveModelBuilder"/>
      <xs:element ref="CreateInteractiveDocumentBuilder"/>
      <xs:element ref="CreateModelApplier"/>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Parent Elements

DocumentOutput, Execution, InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput, Node

Child Elements

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

Container Element

Table 72. Attributes for Container

Attribute	Use	Description	Valid Values
name	**required**		*string*
type	optional		*string*

XML Representation

```
<xs:element name="Container">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="type" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Containers, Containers, Containers, Containers, Containers

ContainerFile Element

Table 73. Attributes for ContainerFile

Attribute	Use	Description	Valid Values
container	optional		
containerType	optional		

Table 73. Attributes for ContainerFile (continued)

Attribute	Use	Description	Valid Values
path	**required**		

XML Representation

```
<xs:element name="ContainerFile" type="SERVER-CONTAINER-FILE">
  <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
  <xs:attribute name="container" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="containerType" type="EVALUATED-STRING" use="optional"/>
</xs:element>
```

Parent Elements

InputFiles, OutputFiles

ContainerTypes Element

XML Representation

```
<xs:element name="ContainerTypes">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="DocumentType"/>
      <xs:element ref="ModelType"/>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Parent Elements

CommonObjects

Child Elements

DocumentType, ModelType

Controls Element

XML Representation

```
<xs:element name="Controls">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="Menu"/>
      <xs:element ref="MenuItem"/>
      <xs:element ref="ToolbarItem"/>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Parent Elements

UserInterface

Child Elements

Menu, MenuItem, ToolbarItem

CreateDocument Element

Table 74. Attributes for CreateDocument

Attribute	Use	Description	Valid Values
sourceFile	**required**		*string*

Table 74. Attributes for CreateDocument (continued)

Attribute	Use	Description	Valid Values
target	**required**		*string*
type	optional		*string*

XML Representation

```
<xs:element name="CreateDocument">
  <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
    <xs:choice>
      <xs:element ref="Condition"/>
      <xs:element ref="And"/>
      <xs:element ref="Or"/>
      <xs:element ref="Not"/>
    </xs:choice>
  </xs:group>
  <xs:attribute name="sourceFile" type="xs:string" use="required"/>
  <xs:attribute name="target" type="xs:string" use="required"/>
  <xs:attribute name="type" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

Child Elements

And, Condition, Not, Or

Related Elements

CreateModel

CreateDocumentOutput Element

Table 75. Attributes for CreateDocumentOutput

Attribute	Use	Description	Valid Values
type	**required**		*string*

XML Representation

```
<xs:element name="CreateDocumentOutput">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="SetProperty"/>
        <xs:element ref="SetContainer"/>
        <xs:element ref="CreateModel"/>
        <xs:element ref="CreateDocument"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

Constructors

Child Elements

And, Condition, CreateDocument, CreateModel, Not, Or, SetContainer, SetProperty

Related Elements

CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

CreateInteractiveDocumentBuilder Element

Table 76. Attributes for CreateInteractiveDocumentBuilder

Attribute	Use	Description	Valid Values
type	**required**		*string*

XML Representation

```
<xs:element name="CreateInteractiveDocumentBuilder">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="SetProperty"/>
        <xs:element ref="SetContainer"/>
        <xs:element ref="CreateModel"/>
        <xs:element ref="CreateDocument"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

Constructors

Child Elements

And, Condition, CreateDocument, CreateModel, Not, Or, SetContainer, SetProperty

Related Elements

CreateDocumentOutput, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

CreateInteractiveModelBuilder Element

Table 77. Attributes for CreateInteractiveModelBuilder

Attribute	Use	Description	Valid Values
type	**required**		*string*

XML Representation

```
<xs:element name="CreateInteractiveModelBuilder">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="SetProperty"/>
        <xs:element ref="SetContainer"/>
        <xs:element ref="CreateModel"/>
        <xs:element ref="CreateDocument"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

Constructors

Child Elements

And, Condition, CreateDocument, CreateModel, Not, Or, SetContainer, SetProperty

Related Elements

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateModelApplier, CreateModelOutput

CreateModel Element

Table 78. Attributes for CreateModel

Attribute	Use	Description	Valid Values
signatureFile	optional		*string*
sourceFile	**required**		*string*
target	**required**		*string*
type	optional		*string*

XML Representation

```
<xs:element name="CreateModel">
  <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
    <xs:choice>
      <xs:element ref="Condition"/>
      <xs:element ref="And"/>
      <xs:element ref="Or"/>
      <xs:element ref="Not"/>
    </xs:choice>
  </xs:group>
  <xs:attribute name="sourceFile" type="xs:string" use="required"/>
  <xs:attribute name="target" type="xs:string" use="required"/>
  <xs:attribute name="type" type="xs:string" use="optional"/>
  <xs:sequence>
    <xs:element name="ModelDetail" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="signatureFile" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

Child Elements

And, Condition, ModelDetail, Not, Or

Related Elements

CreateDocument

ModelDetail Element:

Table 79. Attributes for ModelDetail

Attribute	Use	Description	Valid Values
algorithm	**required**		*string*

XML Representation

```
<xs:element name="ModelDetail" maxOccurs="unbounded">
  <xs:attribute name="algorithm" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

CreateModel

CreateModelApplier Element

Table 80. Attributes for CreateModelApplier

Attribute	Use	Description	Valid Values
type	**required**		*string*

XML Representation

```
<xs:element name="CreateModelApplier">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="SetProperty"/>
        <xs:element ref="SetContainer"/>
        <xs:element ref="CreateModel"/>
        <xs:element ref="CreateDocument"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

Constructors

Child Elements

And, Condition, CreateDocument, CreateModel, Not, Or, SetContainer, SetProperty

Related Elements

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelOutput

CreateModelOutput Element

Table 81. Attributes for CreateModelOutput

Attribute	Use	Description	Valid Values
type	**required**		*string*

XML Representation

```
<xs:element name="CreateModelOutput">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="SetProperty"/>
        <xs:element ref="SetContainer"/>
        <xs:element ref="CreateModel"/>
        <xs:element ref="CreateDocument"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="type" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

Constructors

Child Elements

And, Condition, CreateDocument, CreateModel, Not, Or, SetContainer, SetProperty

Related Elements

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier

DatabaseConnectionValue Element

A value specifying the details of a database connection.

Table 82. Attributes for DatabaseConnectionValue

Attribute	Use	Description	Valid Values
connectionType	**required**		*string*
datasourceName	**required**		*string*
password	**required**		*string*
userName	**required**		*string*

XML Representation

```
<xs:element name="DatabaseConnectionValue" type="DATABASE-CONNECTION-VALUE">
  <xs:attribute name="connectionType" type="xs:string" use="required"/>
  <xs:attribute name="datasourceName" type="xs:string" use="required"/>
  <xs:attribute name="userName" type="xs:string" use="required"/>
  <xs:attribute name="password" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

Attribute, Attribute, ListValue, ListValue, ListValue, Parameter

DataFile Element

Table 83. Attributes for DataFile

Attribute	Use	Description	Valid Values
path	**required**		

XML Representation

```
<xs:element name="DataFile" type="SERVER-DATA-FILE">
  <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
  <xs:choice>
    <xs:element ref="DelimitedDataFormat"/>
  </xs:choice>
</xs:element>
```

Parent Elements

InputFiles, OutputFiles

Child Elements

DelimitedDataFormat

DataFormat Element

XML Representation

```
<xs:element name="DataFormat">
  <xs:group ref="DATA-FORMAT-TYPE">
    <xs:choice>
      <xs:element ref="DelimitedDataFormat"/>
      <xs:element ref="SPSSDataFormat"/>
    </xs:choice>
  </xs:group>
</xs:element>
```

Parent Elements

FileFormatType

Child Elements

DelimitedDataFormat, SPSSDataFormat

DataModel Element

The data model coming into or out of a node. An input/Output data model is a set of Fields.

XML Representation

```
<xs:element name="DataModel" type="DATA-MODEL">
  <xs:sequence>
    <xs:element name="FieldFormats" type="FIELD-FORMATS" minOccurs="0">
      <xs:sequence>
```

```
        <xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION" minOccurs="0" maxOccurs="unbounded">
        </xs:element>
      </xs:sequence>
    </xs:element>
    <xs:element name="FieldGroups" type="FIELD-GROUPS" minOccurs="0">
      <xs:sequence>
        <xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION" minOccurs="0" maxOccurs="unbounded">
          <xs:sequence>
            <xs:element name="FieldName">
            </xs:element>
          </xs:sequence>
        </xs:element>
      </xs:sequence>
    </xs:element>
    <xs:element name="Fields" type="FIELDS">
      <xs:sequence>
        <xs:element name="Field" type="FIELD" minOccurs="0" maxOccurs="unbounded">
          <xs:group ref="FIELD-CONTENT">
            <xs:sequence>
              <xs:element ref="DisplayLabel"/>
              <xs:choice minOccurs="0">
                <xs:element ref="Range"/>
                <xs:element ref="Values"/>
              </xs:choice>
              <xs:element ref="MissingValues"/>
            </xs:sequence>
          </xs:group>
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
</xs:element>
```

Child Elements

FieldFormats, FieldGroups, Fields

FieldFormats Element: Defines the default field formats. Field formats are used when displaying values in output such as the general format (standard number, scientific or currency formats), number of decimal places to display, decimal separator etc. Currently field formats are only used for numeric fields although this may change in future versions.

Table 84. Attributes for FieldFormats

Attribute	Use	Description	Valid Values
count	optional		*nonNegativeInteger*
defaultNumberFormat	**required**		*string*

XML Representation

```
<xs:element name="FieldFormats" type="FIELD-FORMATS" minOccurs="0">
  <xs:sequence>
    <xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION" minOccurs="0" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="defaultNumberFormat" type="xs:string" use="required"/>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Parent Elements

DataModel

Child Elements

NumberFormat

NumberFormat Element: Defines format information for a numeric field.

Table 85. Attributes for NumberFormat

Attribute	Use	Description	Valid Values
decimalPlaces	**required**		*nonNegativeInteger*
decimalSymbol	**required**		**period** **comma**
formatType	**required**		**standard** **scientific** **currency**
groupingSymbol	**required**		**none** **period** **comma** **space**
name	**required**		*string*

XML Representation

```
<xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="formatType" type="NUMBER-FORMAT-TYPE" use="required">
    <xs:enumeration value="standard"/>
    <xs:enumeration value="scientific"/>
    <xs:enumeration value="currency"/>
  </xs:attribute>
  <xs:attribute name="decimalPlaces" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="decimalSymbol" type="DECIMAL-SYMBOL" use="required">
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
  </xs:attribute>
  <xs:attribute name="groupingSymbol" type="NUMBER-GROUPING-SYMBOL" use="required">
    <xs:enumeration value="none"/>
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
    <xs:enumeration value="space"/>
  </xs:attribute>
</xs:element>
```

Parent Elements

FieldFormats

FieldGroups Element: Defines the field groups. Field groups are used to associate related fields. For example, a survey question that asks a respondent to select which locations they have visited from a set of options will be represented as a set of flag fields. A field group may be used to identify which fields are associated with that survey question.

Table 86. Attributes for FieldGroups

Attribute	Use	Description	Valid Values
count	optional		*nonNegativeInteger*

XML Representation

```
<xs:element name="FieldGroups" type="FIELD-GROUPS" minOccurs="0">
  <xs:sequence>
    <xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION" minOccurs="0" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="FieldName">
        </xs:element>
      </xs:sequence>
```

```
    </xs:element>
  </xs:sequence>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Parent Elements

DataModel

Child Elements

FieldGroup

FieldGroup Element: Defines a field group. A field group consists of a list of field names and information about the field group such as the group name and optional label, type of group and for multi-dichotomy groups, the counted value i.e. the value which represents "true".

Table 87. Attributes for FieldGroup

Attribute	Use	Description	Valid Values
count	optional		*nonNegativeInteger*
countedValue	optional		*string*
displayLabel	optional		*string*
groupType	**required**		**fieldGroup** **multiCategorySet** **multiDichotomySet**
name	**required**		

XML Representation

```
<xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION" minOccurs="0" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="FieldName">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-GROUP-NAME" use="required"/>
  <xs:attribute name="displayLabel" type="xs:string"/>
  <xs:attribute name="groupType" type="FIELD-GROUP-TYPE" use="required">
    <xs:enumeration value="fieldGroup"/>
    <xs:enumeration value="multiCategorySet"/>
    <xs:enumeration value="multiDichotomySet"/>
  </xs:attribute>
  <xs:attribute name="countedValue" type="xs:string"/>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Parent Elements

FieldGroups

Child Elements

FieldName

FieldName Element:

Table 88. Attributes for FieldName

Attribute	Use	Description	Valid Values
name	**required**		

XML Representation

```
<xs:element name="FieldName">
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
</xs:element>
```

Parent Elements

FieldGroup

Fields Element:

Table 89. Attributes for Fields

Attribute	Use	Description	Valid Values
count	optional		*nonNegativeInteger*

XML Representation

```
<xs:element name="Fields" type="FIELDS">
  <xs:sequence>
    <xs:element name="Field" type="FIELD" minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="FIELD-CONTENT">
        <xs:sequence>
          <xs:element ref="DisplayLabel"/>
          <xs:choice minOccurs="0">
            <xs:element ref="Range"/>
            <xs:element ref="Values"/>
          </xs:choice>
          <xs:element ref="MissingValues"/>
        </xs:sequence>
      </xs:group>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Parent Elements

DataModel

Child Elements

Field

Field Element:

Table 90. Attributes for Field

Attribute	Use	Description	Valid Values
direction	optional		**in** **out** **both** **none** **partition**
displayLabel	optional		*string*
name	**required**		*string*

Table 90. Attributes for Field (continued)

Attribute	Use	Description	Valid Values
storage	optional		**unknown** **integer** **real** **string** **date** **time** **timestamp**
type	optional		**auto** **range** **discrete** **set** **orderedSet** **flag** **typeless**

XML Representation

```
<xs:element name="Field" type="FIELD" minOccurs="0" maxOccurs="unbounded">
  <xs:group ref="FIELD-CONTENT">
    <xs:sequence>
      <xs:element ref="DisplayLabel"/>
      <xs:choice minOccurs="0">
        <xs:element ref="Range"/>
        <xs:element ref="Values"/>
      </xs:choice>
      <xs:element ref="MissingValues"/>
    </xs:sequence>
  </xs:group>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="type" type="FIELD-TYPE" default="auto">
    <xs:enumeration value="auto"/>
    <xs:enumeration value="range"/>
    <xs:enumeration value="discrete"/>
    <xs:enumeration value="set"/>
    <xs:enumeration value="orderedSet"/>
    <xs:enumeration value="flag"/>
    <xs:enumeration value="typeless"/>
  </xs:attribute>
  <xs:attribute name="storage" type="FIELD-STORAGE" default="unknown">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="real"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="time"/>
    <xs:enumeration value="timestamp"/>
  </xs:attribute>
  <xs:attribute name="direction" type="FIELD-DIRECTION" default="in">
    <xs:enumeration value="in"/>
    <xs:enumeration value="out"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
    <xs:enumeration value="partition"/>
  </xs:attribute>
  <xs:attribute name="displayLabel" type="xs:string"/>
</xs:element>
```

Parent Elements

Fields

Child Elements

DisplayLabel, MissingValues, Range, Range, Values, Values

DBConnectionChooserControl Element

Table 91. Attributes for DBConnectionChooserControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="DBConnectionChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

DBTableChooserControl Element

Table 92. Attributes for DBTableChooserControl

Attribute	Use	Description	Valid Values
connectionProperty	**required**		*string*

Table 92. Attributes for DBTableChooserControl (continued)

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="DBTableChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="connectionProperty" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

DefaultValue Element

XML Representation

```
<xs:element name="DefaultValue">
  <xs:choice>
    <xs:element name="ServerTempFile">
    </xs:element>
    <xs:element name="ServerTempDir">
```

```
    </xs:element>
    <xs:element name="Identifier">
    </xs:element>
  </xs:choice>
</xs:element>
```

Parent Elements

Property, PropertyType

Child Elements

Identifier, ServerTempDir, ServerTempFile

ServerTempFile Element:

Table 93. Attributes for ServerTempFile

Attribute	Use	Description	Valid Values
basename	**required**		

XML Representation

```
<xs:element name="ServerTempFile">
  <xs:attribute name="basename" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Parent Elements

DefaultValue

ServerTempDir Element:

Table 94. Attributes for ServerTempDir

Attribute	Use	Description	Valid Values
basename	**required**		

XML Representation

```
<xs:element name="ServerTempDir">
  <xs:attribute name="basename" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Parent Elements

DefaultValue

Identifier Element:

Table 95. Attributes for Identifier

Attribute	Use	Description	Valid Values
basename	**required**		

XML Representation

```
<xs:element name="Identifier">
  <xs:attribute name="basename" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Parent Elements

DefaultValue

DelimitedDataFormat Element

Table 96. Attributes for DelimitedDataFormat

Attribute	Use	Description	Valid Values
delimeter	optional		**tab** **comma** **semicolon** **colon** **verticalBar** **other**
eol	optional		**cr** **crlf** **lf** **other**
includeFieldNames	optional		*boolean*
otherDelimiter	optional		*string*
otherEol	optional		*string*
quoteStrings	optional		*boolean*
stringQuote	optional		*string*

XML Representation

```
<xs:element name="DelimitedDataFormat">
  <xs:attribute name="delimeter" use="optional" default="tab">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="tab"/>
        <xs:enumeration value="comma"/>
        <xs:enumeration value="semicolon"/>
        <xs:enumeration value="colon"/>
        <xs:enumeration value="verticalBar"/>
        <xs:enumeration value="other"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="otherDelimiter" type="xs:string" use="optional"/>
  <xs:attribute name="eol" use="optional" default="cr">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="cr"/>
        <xs:enumeration value="crlf"/>
        <xs:enumeration value="lf"/>
        <xs:enumeration value="other"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="otherEol" type="xs:string" use="optional"/>
  <xs:attribute name="includeFieldNames" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="quoteStrings" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="stringQuote" type="xs:string" use="optional" default="""/>
</xs:element>
```

Parent Elements

DataFile, DataFormat

DisplayLabel Element

A display label for a field or value in a specified language. The displayLabel attribute can be used where there is no label for a particular language.

Table 97. Attributes for DisplayLabel

Attribute	Use	Description	Valid Values
lang	optional		*NMTOKEN*
value	**required**		*string*

XML Representation

```
<xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="lang" type="xs:NMTOKEN" default="en"/>
</xs:element>
```

Parent Elements

Field

DocumentBuilder Element

XML Representation

```
<xs:element name="DocumentBuilder">
  <xs:sequence>
    <xs:element name="DocumentGeneration">
    </xs:element>
  </xs:sequence>
</xs:element>
```

Parent Elements

Node

Child Elements

DocumentGeneration

DocumentGeneration Element:

Table 98. Attributes for DocumentGeneration

Attribute	Use	Description	Valid Values
controlsId	**required**		*string*

XML Representation

```
<xs:element name="DocumentGeneration">
  <xs:attribute name="controlsId" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

DocumentBuilder

DocumentOutput Element

Table 99. Attributes for DocumentOutput

Attribute	Use	Description	Valid Values
deprecatedScriptNames	optional		*string*
id	**required**		*string*
scriptName	optional		*string*

XML Representation

```
<xs:element name="DocumentOutput">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="Properties"/>
      <xs:element name="Containers" minOccurs="0">
        <xs:sequence maxOccurs="unbounded">
          <xs:element ref="Container"/>
        </xs:sequence>
      </xs:element>
      <xs:element ref="UserInterface"/>
      <xs:element ref="Constructors" minOccurs="0"/>
      <xs:element ref="ModelProvider" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Extension

Child Elements

Constructors, Containers, ModelProvider, Properties, UserInterface

Related Elements

InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput, Node

Containers Element:

XML Representation

```
<xs:element name="Containers" minOccurs="0">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Container"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

Node

Child Elements

Container

DocumentType Element

Defines a new document type

Table 100. Attributes for DocumentType

Attribute	Use	Description	Valid Values
format	**required**		**utf8** **binary**
id	**required**		*string*
type	optional		**unknown** **rowSet** **report** **graph**

XML Representation

```
<xs:element name="DocumentType">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="format" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="utf8"/>
        <xs:enumeration value="binary"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="type" type="DOCUMENT-TYPE" use="optional">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="rowSet"/>
    <xs:enumeration value="report"/>
    <xs:enumeration value="graph"/>
  </xs:attribute>
</xs:element>
```

Parent Elements

ContainerTypes

Related Elements

ModelType

Enabled Element

XML Representation

```
<xs:element name="Enabled">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Parent Elements

ActionButton, CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, ComboBoxControl, DBConnectionChooserControl, DBTableChooserControl, ExtensionObjectPanel, ItemChooserControl, ModelViewerPanel, MultiFieldChooserControl, MultiItemChooserControl, PasswordBoxControl, PropertiesPanel, PropertiesSubPanel, PropertyControl,

RadioButtonGroupControl, SelectorPanel, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SingleItemChooserControl, SpinnerControl, StaticText, SystemControls, TabbedPanel, TableControl, TextAreaControl, TextBoxControl, TextBrowserPanel

Child Elements

And, Condition, Not, Or

Enumeration Element

XML Representation

```
<xs:element name="Enumeration">
  <xs:sequence>
    <xs:element name="Enum" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:element>
```

Parent Elements

PropertyType

Child Elements

Enum

Enum Element:

Table 101. Attributes for Enum

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
imagePath	optional		*string*
imagePathKey	optional		*string*
label	**required**		*string*
labelKey	optional		*string*
value	**required**		*string*

XML Representation

```
<xs:element name="Enum" maxOccurs="unbounded">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="imagePath" type="xs:string" use="optional"/>
  <xs:attribute name="imagePathKey" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Enumeration

ErrorDetail Element

Suuplementary information about an error or other condition.

XML Representation

```
<xs:element name="ErrorDetail" type="ERROR-DETAIL">
  <xs:sequence>
    <xs:element name="Diagnostic" type="DIAGNOSTIC" minOccurs="0" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
        </xs:element>
        <xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
</xs:element>
```

Child Elements

Diagnostic

Diagnostic Element:

Table 102. Attributes for Diagnostic

Attribute	Use	Description	Valid Values
code	**required**		*integer*
severity	optional		**unknown** **information** **warning** **error** **fatal**
source	optional		*string*
subCode	optional		*integer*

XML Representation

```
<xs:element name="Diagnostic" type="DIAGNOSTIC" minOccurs="0" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
    </xs:element>
    <xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="code" type="xs:integer" use="required"/>
  <xs:attribute name="subCode" type="xs:integer" default="0"/>
  <xs:attribute name="severity" type="DIAGNOSTIC-SEVERITY" default="error">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="information"/>
    <xs:enumeration value="warning"/>
    <xs:enumeration value="error"/>
    <xs:enumeration value="fatal"/>
  </xs:attribute>
  <xs:attribute name="source" type="xs:string"/>
</xs:element>
```

Parent Elements

ErrorDetail

Child Elements

Message, Parameter

Message Element:

Table 103. Attributes for Message

Attribute	Use	Description	Valid Values
lang	optional		*NMTOKEN*

XML Representation

```
<xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
  <xs:attribute name="lang" type="xs:NMTOKEN"/>
</xs:element>
```

Parent Elements

Diagnostic

Parameter Element:

XML Representation

```
<xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
```

Parent Elements

Diagnostic

Executable Element

XML Representation

```
<xs:element name="Executable">
  <xs:sequence>
    <xs:element ref="Run" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

Execution

Child Elements

Run

Execution Element

XML Representation

```
<xs:element name="Execution">
  <xs:sequence>
    <xs:element ref="Properties" minOccurs="0"/>
    <xs:element ref="InputFiles"/>
    <xs:element ref="OutputFiles"/>
    <xs:choice>
      <xs:element ref="Executable"/>
      <xs:element ref="Module"/>
    </xs:choice>
    <xs:element ref="Constructors" minOccurs="0"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

Node

Child Elements

Constructors, Executable, InputFiles, Module, OutputFiles, Properties

Extension Element

Defines the top-level extension container.

Table 104. Attributes for Extension

Attribute	Use	Description	Valid Values
debug	optional		*boolean*
version	**required**		*string*

XML Representation

```
<xs:element name="Extension">
  <xs:sequence>
    <xs:element ref="ExtensionDetails"/>
    <xs:element ref="Resources"/>
    <xs:element ref="License" minOccurs="0"/>
    <xs:element ref="CommonObjects"/>
    <xs:element ref="UserInterface" minOccurs="0"/>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
      <xs:choice>
        <xs:element ref="Node"/>
        <xs:element ref="ModelOutput"/>
        <xs:element ref="DocumentOutput"/>
        <xs:element ref="InteractiveModelBuilder"/>
        <xs:element ref="InteractiveDocumentBuilder"/>
      </xs:choice>
    </xs:sequence>
  </xs:sequence>
  <xs:attribute name="version" type="xs:string" use="required"/>
  <xs:attribute name="debug" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Child Elements

CommonObjects, DocumentOutput, ExtensionDetails, InteractiveDocumentBuilder, InteractiveModelBuilder, License, ModelOutput, Node, Resources, UserInterface

ExtensionDetails Element

Defines information about the extension such as the extension id, the extension provider and version information.

Table 105. Attributes for ExtensionDetails

Attribute	Use	Description	Valid Values
copyright	optional		*string*
description	optional		*string*
id	**required**		*string*
label	**required**		*string*
provider	optional		*string*
providerTag	**required**		*string*
version	optional		*string*

XML Representation

```
<xs:element name="ExtensionDetails">
  <xs:attribute name="providerTag" type="xs:string" use="required"/>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="version" type="xs:string"/>
```

```
  <xs:attribute name="provider" type="xs:string" use="optional" default="(unknown)"/>
  <xs:attribute name="copyright" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Extension

ExtensionObjectPanel Element

Table 106. Attributes for ExtensionObjectPanel

Attribute	Use	Description	Valid Values
id	optional		*string*
panelClass	**required**		*string*

XML Representation

```
<xs:element name="ExtensionObjectPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="panelClass" type="xs:string" use="required"/>
  <xs:attribute name="id" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel, Tab

Child Elements

Enabled, Layout, Visible

Related Elements

ActionButton, ComboBoxControl, ModelViewerPanel, SelectorPanel, StaticText, SystemControls, TabbedPanel, TextBrowserPanel

Field Element

Table 107. Attributes for Field

Attribute	Use	Description	Valid Values
direction	optional		**in** **out** **both** **none** **partition**
label	optional		*string*
name	**required**		

Table 107. Attributes for Field (continued)

Attribute	Use	Description	Valid Values
storage	optional		**unknown** **integer** **real** **string** **date** **time** **timestamp**
type	optional		**auto** **range** **discrete** **set** **orderedSet** **flag** **typeless**

XML Representation

```
<xs:element name="Field" type="FIELD-DECLARATION">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Range" minOccurs="0"/>
      <xs:element ref="Values" minOccurs="0"/>
      <xs:element ref="NumericInfo" minOccurs="0"/>
      <xs:element name="MissingValues" minOccurs="0">
        <xs:sequence>
          <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="Range" minOccurs="0"/>
        </xs:sequence>
      </xs:element>
      <xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
      </xs:element>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
  <xs:attribute name="storage" type="FIELD-STORAGE">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="real"/>
    <xs:enumeration value="string"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="time"/>
    <xs:enumeration value="timestamp"/>
  </xs:attribute>
  <xs:attribute name="type" type="FIELD-TYPE">
    <xs:enumeration value="auto"/>
    <xs:enumeration value="range"/>
    <xs:enumeration value="discrete"/>
    <xs:enumeration value="set"/>
    <xs:enumeration value="orderedSet"/>
    <xs:enumeration value="flag"/>
    <xs:enumeration value="typeless"/>
  </xs:attribute>
  <xs:attribute name="direction" type="FIELD-DIRECTION">
    <xs:enumeration value="in"/>
    <xs:enumeration value="out"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
    <xs:enumeration value="partition"/>
  </xs:attribute>
  <xs:attribute name="label" type="xs:string"/>
</xs:element>
```

Child Elements

MissingValues, ModelField, NumericInfo, Range, Range, Values, Values

MissingValues Element:

Table 108. Attributes for MissingValues

Attribute	Use	Description	Valid Values
treatNullAsMissing	optional		*boolean*
treatWhitespaceAsMissing	optional		*boolean*

XML Representation

```
<xs:element name="MissingValues" minOccurs="0">
  <xs:sequence>
    <xs:element ref="Values" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Range" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="treatWhitespaceAsMissing" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="treatNullAsMissing" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Parent Elements

Field

Child Elements

Range, Range, Values, Values

ModelField Element:

Table 109. Attributes for ModelField

Attribute	Use	Description	Valid Values
group	optional		
role	**required**		**unknown** **predictedValue** **predictedDisplayValue** **probability** **residual** **standardError** **entityId** **entityAffinity** **upperConfidenceLimit** **lowerConfidenceLimit** **propensity** **value** **supplementary**
tag	optional		*string*
targetField	optional		*string*
value	optional		*string*

XML Representation

```
<xs:element name="ModelField" type="MODEL-FIELD-INFORMATION" minOccurs="0">
  <xs:attribute name="role" type="MODEL-FIELD-ROLE" use="required">
    <xs:enumeration value="unknown"/>
```

```
    <xs:enumeration value="predictedValue"/>
    <xs:enumeration value="predictedDisplayValue"/>
    <xs:enumeration value="probability"/>
    <xs:enumeration value="residual"/>
    <xs:enumeration value="standardError"/>
    <xs:enumeration value="entityId"/>
    <xs:enumeration value="entityAffinity"/>
    <xs:enumeration value="upperConfidenceLimit"/>
    <xs:enumeration value="lowerConfidenceLimit"/>
    <xs:enumeration value="propensity"/>
    <xs:enumeration value="value"/>
    <xs:enumeration value="supplementary"/>
  </xs:attribute>
  <xs:attribute name="targetField" type="xs:string"/>
  <xs:attribute name="value" type="xs:string"/>
  <xs:attribute name="group" type="MODEL-FIELD-GROUP"/>
  <xs:attribute name="tag" type="xs:string"/>
</xs:element>
```

Parent Elements

Field

FieldFormats Element

Defines the default field formats. Field formats are used when displaying values in output such as the general format (standard number, scientific or currency formats), number of decimal places to display, decimal separator etc. Currently field formats are only used for numeric fields although this may change in future versions.

Table 110. Attributes for FieldFormats

Attribute	Use	Description	Valid Values
count	optional		*nonNegativeInteger*
defaultNumberFormat	**required**		*string*

XML Representation

```
<xs:element name="FieldFormats" type="FIELD-FORMATS">
  <xs:sequence>
    <xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION" minOccurs="0" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="defaultNumberFormat" type="xs:string" use="required"/>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Child Elements

NumberFormat

NumberFormat Element: Defines format information for a numeric field.

Table 111. Attributes for NumberFormat

Attribute	Use	Description	Valid Values
decimalPlaces	**required**		*nonNegativeInteger*
decimalSymbol	**required**		**period** **comma**
formatType	**required**		**standard** **scientific** **currency**

Table 111. Attributes for NumberFormat (continued)

Attribute	Use	Description	Valid Values
groupingSymbol	**required**		**none** **period** **comma** **space**
name	**required**		*string*

XML Representation

```
<xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="formatType" type="NUMBER-FORMAT-TYPE" use="required">
    <xs:enumeration value="standard"/>
    <xs:enumeration value="scientific"/>
    <xs:enumeration value="currency"/>
  </xs:attribute>
  <xs:attribute name="decimalPlaces" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="decimalSymbol" type="DECIMAL-SYMBOL" use="required">
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
  </xs:attribute>
  <xs:attribute name="groupingSymbol" type="NUMBER-GROUPING-SYMBOL" use="required">
    <xs:enumeration value="none"/>
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
    <xs:enumeration value="space"/>
  </xs:attribute>
</xs:element>
```

Parent Elements

FieldFormats

FieldGroup Element

Defines a field group. A field group consists of a list of field names and information about the field group such as the group name and optional label, type of group and for multi-dichotomy groups, the counted value i.e. the value which represents "true".

Table 112. Attributes for FieldGroup

Attribute	Use	Description	Valid Values
count	optional		*nonNegativeInteger*
countedValue	optional		*string*
displayLabel	optional		*string*
groupType	**required**		**fieldGroup** **multiCategorySet** **multiDichotomySet**
name	**required**		

XML Representation

```
<xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION">
  <xs:sequence>
    <xs:element name="FieldName">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-GROUP-NAME" use="required"/>
  <xs:attribute name="displayLabel" type="xs:string"/>
  <xs:attribute name="groupType" type="FIELD-GROUP-TYPE" use="required">
    <xs:enumeration value="fieldGroup"/>
    <xs:enumeration value="multiCategorySet"/>
```

```
    <xs:enumeration value="multiDichotomySet"/>
  </xs:attribute>
  <xs:attribute name="countedValue" type="xs:string"/>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Child Elements

FieldName

FieldName Element:

Table 113. Attributes for FieldName

Attribute	Use	Description	Valid Values
name	**required**		

XML Representation

```
<xs:element name="FieldName">
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
</xs:element>
```

Parent Elements

FieldGroup

FieldGroups Element

Defines the field groups. Field groups are used to associate related fields. For example, a survey question that asks a respondent to select which locations they have visited from a set of options will be represented as a set of flag fields. A field group may be used to identify which fields are associated with that survey question.

Table 114. Attributes for FieldGroups

Attribute	Use	Description	Valid Values
count	optional		*nonNegativeInteger*

XML Representation

```
<xs:element name="FieldGroups" type="FIELD-GROUPS">
  <xs:sequence>
    <xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION" minOccurs="0" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="FieldName">
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Child Elements

FieldGroup

FieldGroup Element: Defines a field group. A field group consists of a list of field names and information about the field group such as the group name and optional label, type of group and for multi-dichotomy groups, the counted value i.e. the value which represents "true".

Table 115. Attributes for FieldGroup

Attribute	Use	Description	Valid Values
count	optional		*nonNegativeInteger*

Table 115. Attributes for FieldGroup (continued)

Attribute	Use	Description	Valid Values
countedValue	optional		*string*
displayLabel	optional		*string*
groupType	**required**		**fieldGroup** **multiCategorySet** **multiDichotomySet**
name	**required**		

XML Representation

```
<xs:element name="FieldGroup" type="FIELD-GROUP-DECLARATION" minOccurs="0" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="FieldName">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="FIELD-GROUP-NAME" use="required"/>
  <xs:attribute name="displayLabel" type="xs:string"/>
  <xs:attribute name="groupType" type="FIELD-GROUP-TYPE" use="required">
    <xs:enumeration value="fieldGroup"/>
    <xs:enumeration value="multiCategorySet"/>
    <xs:enumeration value="multiDichotomySet"/>
  </xs:attribute>
  <xs:attribute name="countedValue" type="xs:string"/>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Parent Elements

FieldGroups

Child Elements

FieldName

FieldName Element:

Table 116. Attributes for FieldName

Attribute	Use	Description	Valid Values
name	**required**		

XML Representation

```
<xs:element name="FieldName">
  <xs:attribute name="name" type="FIELD-NAME" use="required"/>
</xs:element>
```

Parent Elements

FieldGroup

FileFormatType Element

Table 117. Attributes for FileFormatType

Attribute	Use	Description	Valid Values
name	optional		

XML Representation

```
<xs:element name="FileFormatType">
  <xs:sequence>
    <xs:group ref="FILE-FORMAT">
      <xs:choice>
        <xs:element ref="UTF8Format"/>
        <xs:element ref="BinaryFormat"/>
        <xs:element ref="DataFormat"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
  <xs:attribute name="name" type="EVALUATED-STRING" use="optional"/>
</xs:element>
```

Parent Elements

FileFormatTypes

Child Elements

BinaryFormat, DataFormat, UTF8Format

FileFormatTypes Element

XML Representation

```
<xs:element name="FileFormatTypes">
  <xs:sequence>
    <xs:element ref="FileFormatType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

CommonObjects

Child Elements

FileFormatType

ForEach Element

Table 118. Attributes for ForEach

Attribute	Use	Description	Valid Values
container	optional		*string*
from	optional		*string*
inFields	optional		*string*
inFieldValues	optional		*string*
inProperty	optional		*string*
step	optional		*string*
to	optional		*string*
var	**required**		*string*

XML Representation

```
<xs:element name="ForEach">
  <xs:sequence maxOccurs="unbounded">
    <xs:group ref="DATA-MODEL-EXPRESSION">
      <xs:choice>
        <xs:element ref="ForEach"/>
        <xs:element ref="AddField"/>
        <xs:element ref="ChangeField"/>
```

```
        <xs:element ref="RemoveField"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
  <xs:attribute name="var" type="xs:string" use="required"/>
  <xs:attribute name="inProperty" type="xs:string" use="optional"/>
  <xs:attribute name="inFields" type="xs:string" use="optional"/>
  <xs:attribute name="inFieldValues" type="xs:string" use="optional"/>
  <xs:attribute name="from" type="xs:string" use="optional"/>
  <xs:attribute name="to" type="xs:string" use="optional"/>
  <xs:attribute name="step" type="xs:string" use="optional"/>
  <xs:attribute name="container" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

ForEach, ModelFields

Child Elements

AddField, ChangeField, ForEach, RemoveField

Icon Element

Table 119. Attributes for Icon

Attribute	Use	Description	Valid Values
imagePath	**required**		*string*
resourceID	optional		*string*
type	**required**		**standardNode** **smallNode** **standardWindow**

XML Representation

```
<xs:element name="Icon">
  <xs:attribute name="type" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="standardNode"/>
        <xs:enumeration value="smallNode"/>
        <xs:enumeration value="standardWindow"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="imagePath" type="xs:string" use="required"/>
  <xs:attribute name="resourceID" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Icons, Palette

Icons Element

XML Representation

```
<xs:element name="Icons">
  <xs:sequence>
    <xs:element ref="Icon" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

UserInterface

Child Elements

Icon

InputFiles Element

XML Representation

```
<xs:element name="InputFiles">
  <xs:group ref="RUNTIME-FILES">
    <xs:sequence>
      <xs:element ref="DataFile"/>
      <xs:element ref="ContainerFile" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:group>
</xs:element>
```

Parent Elements

Execution, Module

Child Elements

ContainerFile, DataFile

InteractiveDocumentBuilder Element

Table 120. Attributes for InteractiveDocumentBuilder

Attribute	Use	Description	Valid Values
deprecatedScriptNames	optional		*string*
id	**required**		*string*
scriptName	optional		*string*

XML Representation

```
<xs:element name="InteractiveDocumentBuilder">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="Properties"/>
      <xs:element name="Containers" minOccurs="0">
        <xs:sequence maxOccurs="unbounded">
          <xs:element ref="Container"/>
        </xs:sequence>
      </xs:element>
      <xs:element ref="UserInterface"/>
      <xs:element ref="Constructors" minOccurs="0"/>
      <xs:element ref="ModelProvider" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Extension

Child Elements

Constructors, Containers, ModelProvider, Properties, UserInterface

Related Elements

DocumentOutput, InteractiveModelBuilder, ModelOutput, Node

Containers Element:

XML Representation

```
<xs:element name="Containers" minOccurs="0">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Container"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

Node

Child Elements

Container

InteractiveModelBuilder Element

Table 121. Attributes for InteractiveModelBuilder

Attribute	Use	Description	Valid Values
deprecatedScriptNames	optional		*string*
id	**required**		*string*
scriptName	optional		*string*

XML Representation

```
<xs:element name="InteractiveModelBuilder">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="Properties"/>
      <xs:element name="Containers" minOccurs="0">
        <xs:sequence maxOccurs="unbounded">
          <xs:element ref="Container"/>
        </xs:sequence>
      </xs:element>
      <xs:element ref="UserInterface"/>
      <xs:element ref="Constructors" minOccurs="0"/>
      <xs:element ref="ModelProvider" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Extension

Child Elements

Constructors, Containers, ModelProvider, Properties, UserInterface

Related Elements

DocumentOutput, InteractiveDocumentBuilder, ModelOutput, Node

Containers Element:

XML Representation

```
<xs:element name="Containers" minOccurs="0">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Container"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

Node

Child Elements

Container

Layout Element

Table 122. Attributes for Layout

Attribute	Use	Description	Valid Values
anchor	optional		**north** **northeast** **east** **southeast** **south** **southwest** **west** **northwest** **center**
columnWeight	optional		*double*
fill	optional		**horizontal** **vertical** **both** **none**
gridColumn	optional		*nonNegativeInteger*
gridHeight	optional		*nonNegativeInteger*
gridRow	optional		*nonNegativeInteger*
gridWidth	optional		*nonNegativeInteger*
leftIndent	optional		*nonNegativeInteger*
rowIncrement	optional		*nonNegativeInteger*
rowWeight	optional		*double*

XML Representation

```
<xs:element name="Layout">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="Cell">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="gridRow" type="xs:nonNegativeInteger" use="optional"/>
```

```
  <xs:attribute name="gridColumn" type="xs:nonNegativeInteger" use="optional"/>
  <xs:attribute name="rowIncrement" type="xs:nonNegativeInteger" use="optional"/>
  <xs:attribute name="gridWidth" type="xs:nonNegativeInteger" use="optional" default="1"/>
  <xs:attribute name="gridHeight" type="xs:nonNegativeInteger" use="optional" default="1"/>
  <xs:attribute name="rowWeight" type="xs:double" use="optional"/>
  <xs:attribute name="columnWeight" type="xs:double" use="optional"/>
  <xs:attribute name="fill" type="UI-COMPONENT-FILL" use="optional" default="none">
    <xs:enumeration value="horizontal"/>
    <xs:enumeration value="vertical"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
  </xs:attribute>
  <xs:attribute name="anchor" type="UI-COMPONENT-ANCHOR" use="optional" default="west">
    <xs:enumeration value="north"/>
    <xs:enumeration value="northeast"/>
    <xs:enumeration value="east"/>
    <xs:enumeration value="southeast"/>
    <xs:enumeration value="south"/>
    <xs:enumeration value="southwest"/>
    <xs:enumeration value="west"/>
    <xs:enumeration value="northwest"/>
    <xs:enumeration value="center"/>
  </xs:attribute>
  <xs:attribute name="leftIndent" type="xs:nonNegativeInteger" use="optional"/>
</xs:element>
```

Parent Elements

ActionButton, CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, ComboBoxControl, DBConnectionChooserControl, DBTableChooserControl, ExtensionObjectPanel, ItemChooserControl, ModelViewerPanel, MultiFieldChooserControl, MultiItemChooserControl, PasswordBoxControl, PropertiesPanel, PropertiesSubPanel, PropertyControl, RadioButtonGroupControl, SelectorPanel, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SingleItemChooserControl, SpinnerControl, StaticText, SystemControls, TabbedPanel, TableControl, TextAreaControl, TextBoxControl, TextBrowserPanel

Child Elements

Cell

Cell Element:

Table 123. Attributes for Cell

Attribute	Use	Description	Valid Values
column	**required**		*nonNegativeInteger*
row	**required**		*nonNegativeInteger*
width	**required**		*nonNegativeInteger*

XML Representation

```
<xs:element name="Cell">
  <xs:attribute name="row" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="column" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="width" type="xs:nonNegativeInteger" use="required"/>
</xs:element>
```

Parent Elements

Layout

License Element

Reserved for system use.

XML Representation

```
<xs:element name="License">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="OptionCode"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

Extension

Child Elements

OptionCode

ListValue Element

A sequence of values. All values must have the same content type but this is not checked.

XML Representation

```
<xs:element name="ListValue" type="LIST-VALUE">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
</xs:element>
```

Parent Elements

Attribute, Attribute, ListValue, ListValue, ListValue, Parameter

Child Elements

DatabaseConnectionValue, ListValue, MapValue, StructuredValue, Value

MapValue Element

A set of map entries, each consisting of a key and a value.

XML Representation

```
<xs:element name="MapValue" type="MAP-VALUE">
  <xs:sequence>
    <xs:element name="MapEntry" type="MAP-ENTRY" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="KeyValue" type="KEY-VALUE">
        </xs:element>
        <xs:element name="StructuredValue" type="STRUCTURED-VALUE">
          <xs:sequence>
            <xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
              <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
                <xs:choice>
                  <xs:element ref="MapValue"/>
                  <xs:element ref="StructuredValue"/>
                  <xs:element ref="ListValue"/>
                  <xs:element ref="Value"/>
                  <xs:element ref="DatabaseConnectionValue"/>
                </xs:choice>
              </xs:group>
              <xs:sequence>
                <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
                  <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
                    <xs:choice>
                      <xs:element ref="MapValue"/>
                      <xs:element ref="StructuredValue"/>
                      <xs:element ref="ListValue"/>
                      <xs:element ref="Value"/>
```

```
                    <xs:element ref="DatabaseConnectionValue"/>
                  </xs:choice>
                </xs:group>
              </xs:element>
            </xs:sequence>
          </xs:element>
        </xs:sequence>
      </xs:element>
    </xs:sequence>
  </xs:element>
</xs:element>
```

Parent Elements

Attribute, Attribute, ListValue, ListValue, ListValue, Parameter

Child Elements

MapEntry

MapEntry Element: An entry in a keyed property map. Each entry consists of a key and an associated value.

XML Representation

```
<xs:element name="MapEntry" type="MAP-ENTRY" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="KeyValue" type="KEY-VALUE">
    </xs:element>
    <xs:element name="StructuredValue" type="STRUCTURED-VALUE">
      <xs:sequence>
        <xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
          <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
            <xs:choice>
              <xs:element ref="MapValue"/>
              <xs:element ref="StructuredValue"/>
              <xs:element ref="ListValue"/>
              <xs:element ref="Value"/>
              <xs:element ref="DatabaseConnectionValue"/>
            </xs:choice>
          </xs:group>
          <xs:sequence>
            <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
              <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
                <xs:choice>
                  <xs:element ref="MapValue"/>
                  <xs:element ref="StructuredValue"/>
                  <xs:element ref="ListValue"/>
                  <xs:element ref="Value"/>
                  <xs:element ref="DatabaseConnectionValue"/>
                </xs:choice>
              </xs:group>
            </xs:element>
          </xs:sequence>
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
</xs:element>
```

Parent Elements

MapValue

Child Elements

KeyValue, StructuredValue

KeyValue Element: The key value in a map entry.

Table 124. Attributes for KeyValue

Attribute	Use	Description	Valid Values
value	**required**		*string*

XML Representation

```
<xs:element name="KeyValue" type="KEY-VALUE">
  <xs:attribute name="value" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

MapEntry

StructuredValue Element: A sequence of named values ("attributes").

XML Representation

```
<xs:element name="StructuredValue" type="STRUCTURED-VALUE">
  <xs:sequence>
    <xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
      <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
        <xs:choice>
          <xs:element ref="MapValue"/>
          <xs:element ref="StructuredValue"/>
          <xs:element ref="ListValue"/>
          <xs:element ref="Value"/>
          <xs:element ref="DatabaseConnectionValue"/>
        </xs:choice>
      </xs:group>
      <xs:sequence>
        <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
          <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
            <xs:choice>
              <xs:element ref="MapValue"/>
              <xs:element ref="StructuredValue"/>
              <xs:element ref="ListValue"/>
              <xs:element ref="Value"/>
              <xs:element ref="DatabaseConnectionValue"/>
            </xs:choice>
          </xs:group>
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
</xs:element>
```

Parent Elements

MapEntry

Child Elements

Attribute

Attribute Element:

Table 125. Attributes for Attribute

Attribute	Use	Description	Valid Values
name	**required**		*string*
value	optional		*string*

XML Representation

```
<xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
  <xs:sequence>
    <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
      <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
        <xs:choice>
          <xs:element ref="MapValue"/>
          <xs:element ref="StructuredValue"/>
          <xs:element ref="ListValue"/>
          <xs:element ref="Value"/>
          <xs:element ref="DatabaseConnectionValue"/>
        </xs:choice>
      </xs:group>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="value" type="xs:string"/>
</xs:element>
```

Parent Elements

StructuredValue

Child Elements

DatabaseConnectionValue, ListValue, ListValue, MapValue, StructuredValue, Value

ListValue Element: A sequence of values. All values must have the same content type but this is not checked.

XML Representation

```
<xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
</xs:element>
```

Parent Elements

Attribute

Child Elements

DatabaseConnectionValue, ListValue, MapValue, StructuredValue, Value

Menu Element

Table 126. Attributes for Menu

Attribute	Use	Description	Valid Values
id	**required**		*string*
label	**required**		*string*
labelKey	optional		*string*

Table 126. Attributes for Menu (continued)

Attribute	Use	Description	Valid Values
mnemonic	optional		*string*
mnemonicKey	optional		*string*
offset	optional		*nonNegativeInteger*
separatorAfter	optional		*boolean*
separatorBefore	optional		*boolean*
showIcon	optional		*boolean*
showLabel	optional		*boolean*
systemMenu	**required**		**file** **edit** **insert** **view** **tools** **window** **help** **generate** **file.project** **file.outputs** **file.models** **edit.stream** **edit.node** **edit.outputs** **edit.models** **edit.project** **tools.repository** **tools.options** **tools.streamProperties**

XML Representation

```
<xs:element name="Menu">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="systemMenu" type="STANDARD-MENU" use="required">
    <xs:enumeration value="file"/>
    <xs:enumeration value="edit"/>
    <xs:enumeration value="insert"/>
    <xs:enumeration value="view"/>
    <xs:enumeration value="tools"/>
    <xs:enumeration value="window"/>
    <xs:enumeration value="help"/>
    <xs:enumeration value="generate"/>
    <xs:enumeration value="file.project"/>
    <xs:enumeration value="file.outputs"/>
    <xs:enumeration value="file.models"/>
    <xs:enumeration value="edit.stream"/>
    <xs:enumeration value="edit.node"/>
    <xs:enumeration value="edit.outputs"/>
    <xs:enumeration value="edit.models"/>
    <xs:enumeration value="edit.project"/>
    <xs:enumeration value="tools.repository"/>
    <xs:enumeration value="tools.options"/>
    <xs:enumeration value="tools.streamProperties"/>
  </xs:attribute>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="showIcon" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="separatorBefore" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="separatorAfter" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="offset" type="xs:nonNegativeInteger" use="optional" default="0"/>
</xs:element>
```

Parent Elements

Controls

MenuItem Element

Table 127. Attributes for MenuItem

Attribute	Use	Description	Valid Values
action	**required**		*string*
customMenu	optional		*string*
offset	optional		*nonNegativeInteger*
separatorAfter	optional		*boolean*
separatorBefore	optional		*boolean*
showIcon	optional		*boolean*
showLabel	optional		*boolean*
systemMenu	optional		**file** **edit** **insert** **view** **tools** **window** **help** **generate** **file.project** **file.outputs** **file.models** **edit.stream** **edit.node** **edit.outputs** **edit.models** **edit.project** **tools.repository** **tools.options** **tools.streamProperties**

XML Representation

```
<xs:element name="MenuItem">
  <xs:attribute name="action" type="xs:string" use="required"/>
  <xs:attribute name="systemMenu" type="STANDARD-MENU" use="optional">
    <xs:enumeration value="file"/>
    <xs:enumeration value="edit"/>
    <xs:enumeration value="insert"/>
    <xs:enumeration value="view"/>
    <xs:enumeration value="tools"/>
    <xs:enumeration value="window"/>
    <xs:enumeration value="help"/>
    <xs:enumeration value="generate"/>
    <xs:enumeration value="file.project"/>
    <xs:enumeration value="file.outputs"/>
    <xs:enumeration value="file.models"/>
    <xs:enumeration value="edit.stream"/>
    <xs:enumeration value="edit.node"/>
    <xs:enumeration value="edit.outputs"/>
    <xs:enumeration value="edit.models"/>
    <xs:enumeration value="edit.project"/>
    <xs:enumeration value="tools.repository"/>
    <xs:enumeration value="tools.options"/>
    <xs:enumeration value="tools.streamProperties"/>
  </xs:attribute>
  <xs:attribute name="customMenu" type="xs:string" use="optional"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="showIcon" type="xs:boolean" use="optional" default="false"/>
```

```
  <xs:attribute name="separatorBefore" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="separatorAfter" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="offset" type="xs:nonNegativeInteger" use="optional" default="0"/>
</xs:element>
```

Parent Elements

Controls

MissingValues Element

Table 128. Attributes for MissingValues

Attribute	Use	Description	Valid Values
treatNullAsMissing	optional		*boolean*
treatWhitespaceAsMissing	optional		*boolean*

XML Representation

```
<xs:element name="MissingValues" type="MISSING-VALUES" minOccurs="0">
  <xs:sequence>
    <xs:element name="Range" type="RANGE">
    </xs:element>
    <xs:element name="Values" type="FIELD-VALUES">
      <xs:sequence>
        <xs:element name="Value" type="FIELD-VALUE" minOccurs="0" maxOccurs="unbounded">
          <xs:sequence>
            <xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
            </xs:element>
          </xs:sequence>
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="treatNullAsMissing" type="xs:boolean" default="true"/>
  <xs:attribute name="treatWhitespaceAsMissing" type="xs:boolean" default="false"/>
</xs:element>
```

Parent Elements

Field

Child Elements

Range, Values

Range Element:

Table 129. Attributes for Range

Attribute	Use	Description	Valid Values
maxValue	**required**		*string*
minValue	**required**		*string*

XML Representation

```
<xs:element name="Range" type="RANGE">
  <xs:attribute name="minValue" type="xs:string" use="required"/>
  <xs:attribute name="maxValue" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

MissingValues

Values Element:

Table 130. Attributes for Values

Attribute	Use	Description	Valid Values
count	optional		*nonNegativeInteger*

XML Representation

```
<xs:element name="Values" type="FIELD-VALUES">
  <xs:sequence>
    <xs:element name="Value" type="FIELD-VALUE" minOccurs="0" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Parent Elements

MissingValues

Child Elements

Value

Value Element:

Table 131. Attributes for Value

Attribute	Use	Description	Valid Values
code	**required**		*integer*
displayLabel	optional		*string*
flagValue	optional		*boolean*
value	**required**		*string*

XML Representation

```
<xs:element name="Value" type="FIELD-VALUE" minOccurs="0" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="code" type="xs:integer" use="required"/>
  <xs:attribute name="flagValue" type="xs:boolean"/>
  <xs:attribute name="displayLabel" type="xs:string"/>
</xs:element>
```

Parent Elements

Values

Child Elements

DisplayLabel

DisplayLabel Element: A display label for a field or value in a specified language. The displayLabel attribute can be used where there is no label for a particular language.

Table 132. Attributes for DisplayLabel

Attribute	Use	Description	Valid Values
lang	optional		*NMTOKEN*
value	**required**		*string*

XML Representation

```
<xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="lang" type="xs:NMTOKEN" default="en"/>
</xs:element>
```

Parent Elements

Values

ModelBuilder Element

Table 133. Attributes for ModelBuilder

Attribute	Use	Description	Valid Values
allowNoInputs	optional		*boolean*
allowNoOutputs	optional		*boolean*
miningFunctions	**required**		*any*
nullifyBlanks	optional		*boolean*

XML Representation

```
<xs:element name="ModelBuilder">
  <xs:sequence>
    <xs:element name="Algorithm">
    </xs:element>
    <xs:element name="ModelingFields" minOccurs="0">
      <xs:sequence>
        <xs:element name="InputFields" minOccurs="0">
        </xs:element>
        <xs:element name="OutputFields" minOccurs="0">
        </xs:element>
      </xs:sequence>
    </xs:element>
    <xs:element name="ModelGeneration">
    </xs:element>
    <xs:element name="ModelFields">
      <xs:sequence minOccurs="0" maxOccurs="unbounded">
        <xs:group ref="DATA-MODEL-EXPRESSION">
          <xs:choice>
            <xs:element ref="ForEach"/>
            <xs:element ref="AddField"/>
            <xs:element ref="ChangeField"/>
            <xs:element ref="RemoveField"/>
          </xs:choice>
        </xs:group>
      </xs:sequence>
    </xs:element>
    <xs:element name="ModelEvaluation" minOccurs="0">
      <xs:sequence>
        <xs:element name="RawPropensity" minOccurs="0">
        </xs:element>
        <xs:element name="AdjustedPropensity" minOccurs="0">
        </xs:element>
        <xs:element name="VariableImportance" minOccurs="0">
        </xs:element>
      </xs:sequence>
    </xs:element>
    <xs:element name="AutoModeling" minOccurs="0">
      <xs:sequence>
```

```
        <xs:element name="SimpleSettings">
          <xs:sequence>
            <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:element>
        <xs:element name="ExpertSettings" minOccurs="0">
          <xs:sequence>
            <xs:element ref="Condition" minOccurs="0"/>
            <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:element>
        <xs:element name="PropertyMap" minOccurs="0">
          <xs:sequence>
            <xs:element name="PropertyMapping" maxOccurs="unbounded">
            </xs:element>
          </xs:sequence>
        </xs:element>
        <xs:element ref="Constraint" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="miningFunctions" use="required"/>
  <xs:attribute name="allowNoInputs" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="allowNoOutputs" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="nullifyBlanks" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Parent Elements

Node

Child Elements

Algorithm, AutoModeling, ModelEvaluation, ModelFields, ModelGeneration, ModelingFields

Algorithm Element:

Table 134. Attributes for Algorithm

Attribute	Use	Description	Valid Values
label	**required**		*string*
labelKey	optional		*string*
value	**required**		*string*

XML Representation

```
<xs:element name="Algorithm">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

ModelBuilder

ModelingFields Element:

Table 135. Attributes for ModelingFields

Attribute	Use	Description	Valid Values
controlsId	**required**		*string*
ignoreBOTH	optional		*boolean*

XML Representation

```
<xs:element name="ModelingFields" minOccurs="0">
  <xs:attribute name="controlsId" type="xs:string" use="required"/>
  <xs:sequence>
    <xs:element name="InputFields" minOccurs="0">
    </xs:element>
    <xs:element name="OutputFields" minOccurs="0">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="ignoreBOTH" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Parent Elements

ModelBuilder

Child Elements

InputFields, OutputFields

InputFields Element:

Table 136. Attributes for InputFields

Attribute	Use	Description	Valid Values
label	**required**		*string*
labelKey	optional		*string*
multiple	**required**		*boolean*
onlyDatetime	optional		*boolean*
onlyDiscrete	optional		*boolean*
onlyNumeric	optional		*boolean*
onlyRanges	optional		*boolean*
onlySymbolic	optional		*boolean*
property	**required**		*string*
storage	optional		*string*
types	optional		*string*

XML Representation

```
<xs:element name="InputFields" minOccurs="0">
  <xs:attribute name="storage" type="xs:string" use="optional"/>
  <xs:attribute name="onlyNumeric" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlySymbolic" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDatetime" type="xs:boolean" use="optional"/>
  <xs:attribute name="types" type="xs:string" use="optional"/>
  <xs:attribute name="onlyRanges" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDiscrete" type="xs:boolean" use="optional"/>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="multiple" type="xs:boolean" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

ModelingFields

OutputFields Element:

Table 137. Attributes for OutputFields

Attribute	Use	Description	Valid Values
label	**required**		*string*
labelKey	optional		*string*
multiple	**required**		*boolean*
onlyDatetime	optional		*boolean*
onlyDiscrete	optional		*boolean*
onlyNumeric	optional		*boolean*
onlyRanges	optional		*boolean*
onlySymbolic	optional		*boolean*
property	**required**		*string*
storage	optional		*string*
types	optional		*string*

XML Representation

```
<xs:element name="OutputFields" minOccurs="0">
  <xs:attribute name="storage" type="xs:string" use="optional"/>
  <xs:attribute name="onlyNumeric" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlySymbolic" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDatetime" type="xs:boolean" use="optional"/>
  <xs:attribute name="types" type="xs:string" use="optional"/>
  <xs:attribute name="onlyRanges" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDiscrete" type="xs:boolean" use="optional"/>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="multiple" type="xs:boolean" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

ModelingFields

ModelGeneration Element:

Table 138. Attributes for ModelGeneration

Attribute	Use	Description	Valid Values
controlsId	**required**		*string*

XML Representation

```
<xs:element name="ModelGeneration">
  <xs:attribute name="controlsId" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

ModelBuilder

ModelFields Element:

XML Representation

```
<xs:element name="ModelFields">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:group ref="DATA-MODEL-EXPRESSION">
      <xs:choice>
        <xs:element ref="ForEach"/>
```

```
        <xs:element ref="AddField"/>
        <xs:element ref="ChangeField"/>
        <xs:element ref="RemoveField"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Parent Elements

ModelBuilder

Child Elements

AddField, ChangeField, ForEach, RemoveField

ModelEvaluation Element:

Table 139. Attributes for ModelEvaluation

Attribute	Use	Description	Valid Values
container	**required**		*any*
controlsId	**required**		*string*
outputContainer	optional		*any*

XML Representation

```
<xs:element name="ModelEvaluation" minOccurs="0">
  <xs:attribute name="controlsId" type="xs:string" use="required"/>
  <xs:sequence>
    <xs:element name="RawPropensity" minOccurs="0">
    </xs:element>
    <xs:element name="AdjustedPropensity" minOccurs="0">
    </xs:element>
    <xs:element name="VariableImportance" minOccurs="0">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="container" use="required"/>
  <xs:attribute name="outputContainer" use="optional"/>
</xs:element>
```

Parent Elements

ModelBuilder

Child Elements

AdjustedPropensity, RawPropensity, VariableImportance

RawPropensity Element:

Table 140. Attributes for RawPropensity

Attribute	Use	Description	Valid Values
availabilityProperty	optional		*string*
defaultValue	optional		*boolean*
property	optional		*string*

XML Representation

```
<xs:element name="RawPropensity" minOccurs="0">
  <xs:attribute name="property" type="xs:string" use="optional"/>
  <xs:attribute name="availabilityProperty" type="xs:string" use="optional"/>
  <xs:attribute name="defaultValue" type="xs:boolean" use="optional"/>
</xs:element>
```

Parent Elements

ModelEvaluation

AdjustedPropensity Element:

Table 141. Attributes for AdjustedPropensity

Attribute	Use	Description	Valid Values
availabilityProperty	optional		*string*
defaultValue	optional		*boolean*
property	optional		*string*

XML Representation

```
<xs:element name="AdjustedPropensity" minOccurs="0">
  <xs:attribute name="property" type="xs:string" use="optional"/>
  <xs:attribute name="availabilityProperty" type="xs:string" use="optional"/>
  <xs:attribute name="defaultValue" type="xs:boolean" use="optional"/>
</xs:element>
```

Parent Elements

ModelEvaluation

VariableImportance Element:

Table 142. Attributes for VariableImportance

Attribute	Use	Description	Valid Values
availabilityProperty	optional		*string*
defaultValue	optional		*boolean*
property	optional		*string*

XML Representation

```
<xs:element name="VariableImportance" minOccurs="0">
  <xs:attribute name="property" type="xs:string" use="optional"/>
  <xs:attribute name="availabilityProperty" type="xs:string" use="optional"/>
  <xs:attribute name="defaultValue" type="xs:boolean" use="optional"/>
</xs:element>
```

Parent Elements

ModelEvaluation

AutoModeling Element:

Table 143. Attributes for AutoModeling

Attribute	Use	Description	Valid Values
enabledByDefault	optional		*any*

XML Representation

```
<xs:element name="AutoModeling" minOccurs="0">
  <xs:sequence>
    <xs:element name="SimpleSettings">
      <xs:sequence>
        <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:element>
    <xs:element name="ExpertSettings" minOccurs="0">
      <xs:sequence>
        <xs:element ref="Condition" minOccurs="0"/>
        <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:element>
    <xs:element name="PropertyMap" minOccurs="0">
      <xs:sequence>
        <xs:element name="PropertyMapping" maxOccurs="unbounded">
        </xs:element>
      </xs:sequence>
    </xs:element>
    <xs:element ref="Constraint" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="enabledByDefault" use="optional"/>
</xs:element>
```

Parent Elements

ModelBuilder

Child Elements

Constraint, ExpertSettings, PropertyMap, SimpleSettings

SimpleSettings Element:

XML Representation

```
<xs:element name="SimpleSettings">
  <xs:sequence>
    <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

AutoModeling

Child Elements

PropertyGroup

ExpertSettings Element:

XML Representation

```
<xs:element name="ExpertSettings" minOccurs="0">
  <xs:sequence>
    <xs:element ref="Condition" minOccurs="0"/>
    <xs:element ref="PropertyGroup" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

AutoModeling

Child Elements

Condition, PropertyGroup

PropertyMap Element:

XML Representation

```
<xs:element name="PropertyMap" minOccurs="0">
  <xs:sequence>
    <xs:element name="PropertyMapping" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:element>
```

Parent Elements

AutoModeling

Child Elements

PropertyMapping

PropertyMapping Element:

Table 144. Attributes for PropertyMapping

Attribute	Use	Description	Valid Values
property	**required**		*string*
systemProperty	**required**		*string*

XML Representation

```
<xs:element name="PropertyMapping" maxOccurs="unbounded">
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="systemProperty" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

PropertyMap

ModelOutput Element

Table 145. Attributes for ModelOutput

Attribute	Use	Description	Valid Values
deprecatedScriptNames	optional		*string*
helpLink	optional		*string*
id	**required**		*string*
label	optional		*string*
labelKey	optional		*string*
scriptName	optional		*string*

XML Representation

```
<xs:element name="ModelOutput">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="Properties"/>
      <xs:element name="Containers" minOccurs="0">
```

```
        <xs:sequence maxOccurs="unbounded">
          <xs:element ref="Container"/>
        </xs:sequence>
      </xs:element>
      <xs:element ref="UserInterface"/>
      <xs:element ref="Constructors" minOccurs="0"/>
      <xs:element ref="ModelProvider" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="helpLink" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Extension

Child Elements

Constructors, Containers, ModelProvider, Properties, UserInterface

Related Elements

DocumentOutput, InteractiveDocumentBuilder, InteractiveModelBuilder, Node

Containers Element:

XML Representation

```
<xs:element name="Containers" minOccurs="0">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Container"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

Node

Child Elements

Container

ModelProvider Element

Table 146. Attributes for ModelProvider

Attribute	Use	Description	Valid Values
container	**required**		*string*
isPMML	optional		*boolean*

XML Representation

```
<xs:element name="ModelProvider">
  <xs:attribute name="container" type="xs:string" use="required"/>
  <xs:attribute name="isPMML" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Parent Elements

DocumentOutput, InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput, Node

ModelType Element

Defines a new model type

Table 147. Attributes for ModelType

Attribute	Use	Description	Valid Values
format	**required**		**utf8** **binary**
id	**required**		*string*
inputDirection	optional		*string*
outputDirection	optional		*string*

XML Representation

```
<xs:element name="ModelType">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="format" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="utf8"/>
        <xs:enumeration value="binary"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="inputDirection" type="xs:string" use="optional" default="[in]"/>
  <xs:attribute name="outputDirection" type="xs:string" use="optional" default="[out]"/>
</xs:element>
```

Parent Elements

ContainerTypes

Related Elements

DocumentType

ModelViewerPanel Element

Table 148. Attributes for ModelViewerPanel

Attribute	Use	Description	Valid Values
container	**required**		*string*

XML Representation

```
<xs:element name="ModelViewerPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="container" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

Tab

Child Elements

Enabled, Layout, Visible

Related Elements

ActionButton, ComboBoxControl, ExtensionObjectPanel, SelectorPanel, StaticText, SystemControls, TabbedPanel, TextBrowserPanel

Module Element

Table 149. Attributes for Module

Attribute	Use	Description	Valid Values
libraryId	optional		*string*
name	optional		*string*
systemModule	optional		**SmartScore**

XML Representation

```
<xs:element name="Module">
  <xs:sequence>
    <xs:element ref="InputFiles"/>
    <xs:element ref="OutputFiles"/>
    <xs:element ref="StatusCodes" minOccurs="0" maxOccurs="1"/>
  </xs:sequence>
  <xs:attribute name="systemModule" use="optional" default="SmartScore">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="SmartScore"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="libraryId" type="xs:string" use="optional"/>
  <xs:attribute name="name" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Execution

Child Elements

InputFiles, OutputFiles, StatusCodes

MultiFieldChooserControl Element

Table 150. Attributes for MultiFieldChooserControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
onlyDatetime	optional		*boolean*
onlyDiscrete	optional		*boolean*
onlyNumeric	optional		*boolean*

Table 150. Attributes for MultiFieldChooserControl (continued)

Attribute	Use	Description	Valid Values
onlyRanges	optional		*boolean*
onlySymbolic	optional		*boolean*
property	**required**		*string*
showLabel	optional		*boolean*
storage	optional		*string*
types	optional		*string*

XML Representation

```
<xs:element name="MultiFieldChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="storage" type="xs:string" use="optional"/>
  <xs:attribute name="onlyNumeric" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlySymbolic" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDatetime" type="xs:boolean" use="optional"/>
  <xs:attribute name="types" type="xs:string" use="optional"/>
  <xs:attribute name="onlyRanges" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDiscrete" type="xs:boolean" use="optional"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

MultiItemChooserControl Element

Table 151. Attributes for MultiItemChooserControl

Attribute	Use	Description	Valid Values
catalog	**required**		*string*
description	optional		*string*
descriptionKey	optional		*string*

Table 151. Attributes for MultiItemChooserControl (continued)

Attribute	Use	Description	Valid Values
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="MultiItemChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="catalog" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

SingleItemChooserControl

Node Element

Table 152. Attributes for Node

Attribute	Use	Description	Valid Values
deprecatedScriptNames	optional		*string*
description	optional		*string*
descriptionKey	optional		*string*
helpLink	optional		*string*
id	**required**		*string*
label	**required**		*string*
labelKey	optional		*string*

Table 152. Attributes for Node (continued)

Attribute	Use	Description	Valid Values
palette	optional		**import** **fieldOp** **recordOp** **modeling** **dbModeling** **graph** **output** **export** **modeling.classification** **modeling.association** **modeling.segmentation** **modeling.auto**
relativePosition	optional		*string*
relativeTo	optional		*string*
scriptName	optional		*string*
type	**required**		**dataReader** **dataWriter** **dataTransformer** **modelApplier** **modelBuilder** **documentBuilder**

XML Representation

```
<xs:element name="Node">
  <xs:sequence maxOccurs="unbounded">
    <xs:choice maxOccurs="unbounded">
      <xs:element ref="Properties"/>
      <xs:element name="Containers" minOccurs="0">
        <xs:sequence maxOccurs="unbounded">
          <xs:element ref="Container"/>
        </xs:sequence>
      </xs:element>
      <xs:element ref="UserInterface"/>
      <xs:element ref="Constructors" minOccurs="0"/>
      <xs:element ref="ModelProvider" minOccurs="0"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
  <xs:sequence>
    <xs:element ref="ModelBuilder" minOccurs="0"/>
    <xs:element ref="DocumentBuilder" minOccurs="0"/>
    <xs:element ref="Execution"/>
    <xs:element ref="OutputDataModel" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="type" type="NODE-TYPE" use="required">
    <xs:enumeration value="dataReader"/>
    <xs:enumeration value="dataWriter"/>
    <xs:enumeration value="dataTransformer"/>
    <xs:enumeration value="modelApplier"/>
    <xs:enumeration value="modelBuilder"/>
    <xs:enumeration value="documentBuilder"/>
  </xs:attribute>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="palette" type="SYSTEM-PALETTE" use="optional">
    <xs:enumeration value="import"/>
    <xs:enumeration value="fieldOp"/>
    <xs:enumeration value="recordOp"/>
    <xs:enumeration value="modeling"/>
    <xs:enumeration value="dbModeling"/>
```

```
    <xs:enumeration value="graph"/>
    <xs:enumeration value="output"/>
    <xs:enumeration value="export"/>
    <xs:enumeration value="modeling.classification"/>
    <xs:enumeration value="modeling.association"/>
    <xs:enumeration value="modeling.segmentation"/>
    <xs:enumeration value="modeling.auto"/>
  </xs:attribute>
  <xs:attribute name="helpLink" type="xs:string" use="optional"/>
  <xs:attribute name="relativeTo" type="xs:string" use="optional"/>
  <xs:attribute name="relativePosition" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Extension

Child Elements

Constructors, Containers, DocumentBuilder, Execution, ModelBuilder, ModelProvider, OutputDataModel, Properties, UserInterface

Related Elements

DocumentOutput, InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput

Containers Element:

XML Representation

```
<xs:element name="Containers" minOccurs="0">
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Container"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

Node

Child Elements

Container

Not Element

XML Representation

```
<xs:element name="Not">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Parent Elements

And, Command, Constraint, CreateDocument, CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModel, CreateModelApplier, CreateModelOutput, Enabled, Not, Option, Or, Run, Visible

Child Elements

And, Condition, Not, Or

NumberFormat Element

Defines format information for a numeric field.

Table 153. Attributes for NumberFormat

Attribute	Use	Description	Valid Values
decimalPlaces	**required**		*nonNegativeInteger*
decimalSymbol	**required**		**period** **comma**
formatType	**required**		**standard** **scientific** **currency**
groupingSymbol	**required**		**none** **period** **comma** **space**
name	**required**		*string*

XML Representation

```
<xs:element name="NumberFormat" type="NUMBER-FORMAT-DECLARATION">
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="formatType" type="NUMBER-FORMAT-TYPE" use="required">
    <xs:enumeration value="standard"/>
    <xs:enumeration value="scientific"/>
    <xs:enumeration value="currency"/>
  </xs:attribute>
  <xs:attribute name="decimalPlaces" type="xs:nonNegativeInteger" use="required"/>
  <xs:attribute name="decimalSymbol" type="DECIMAL-SYMBOL" use="required">
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
  </xs:attribute>
  <xs:attribute name="groupingSymbol" type="NUMBER-GROUPING-SYMBOL" use="required">
    <xs:enumeration value="none"/>
    <xs:enumeration value="period"/>
    <xs:enumeration value="comma"/>
    <xs:enumeration value="space"/>
  </xs:attribute>
</xs:element>
```

NumericInfo Element

Table 154. Attributes for NumericInfo

Attribute	Use	Description	Valid Values
mean	optional		*double*
standardDeviation	optional		*double*

XML Representation

```
<xs:element name="NumericInfo">
  <xs:attribute name="mean" type="xs:double"/>
  <xs:attribute name="standardDeviation" type="xs:double"/>
</xs:element>
```

Parent Elements

AddField, ChangeField, Field

Option Element

Table 155. Attributes for Option

Attribute	Use	Description	Valid Values
ifProperty	optional		*string*
unlessProperty	optional		*string*
value	**required**		

XML Representation

```
<xs:element name="Option">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
  <xs:attribute name="value" type="EVALUATED-STRING" use="required"/>
  <xs:attribute name="ifProperty" type="xs:string" use="optional"/>
  <xs:attribute name="unlessProperty" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Run

Child Elements

And, Condition, Not, Or

OptionCode Element

Table 156. Attributes for OptionCode

Attribute	Use	Description	Valid Values
code	optional		*long*
description	optional		*string*
type	optional		**mandatory** **optional**

XML Representation

```
<xs:element name="OptionCode">
  <xs:attribute name="code" type="xs:long"/>
  <xs:attribute name="type" type="LicenseType">
    <xs:enumeration value="mandatory"/>
    <xs:enumeration value="optional"/>
  </xs:attribute>
  <xs:attribute name="description" type="xs:string"/>
</xs:element>
```

Parent Elements

License

Or Element

XML Representation

```
<xs:element name="Or">
  <xs:sequence minOccurs="2" maxOccurs="unbounded">
    <xs:group ref="CONDITION-EXPRESSION">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Parent Elements

And, Command, Constraint, CreateDocument, CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModel, CreateModelApplier, CreateModelOutput, Enabled, Not, Option, Or, Run, Visible

Child Elements

And, Condition, Not, Or

OutputDataModel Element

Table 157. Attributes for OutputDataModel

Attribute	Use	Description	Valid Values
libraryId	optional		*string*
method	optional		**xml** **dataModelProvider** **sharedLibrary**
mode	optional		**fixed** **modify** **extend** **replace**
providerClass	optional		*string*

XML Representation

```
<xs:element name="OutputDataModel">
  <xs:attribute name="mode" use="optional" default="fixed">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="fixed"/>
        <xs:enumeration value="modify"/>
        <xs:enumeration value="extend"/>
        <xs:enumeration value="replace"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="method" use="optional" default="xml">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="xml"/>
        <xs:enumeration value="dataModelProvider"/>
        <xs:enumeration value="sharedLibrary"/>
      </xs:restriction>
    </xs:simpleType>
```

```
  </xs:attribute>
  <xs:attribute name="providerClass" type="xs:string" use="optional"/>
  <xs:attribute name="libraryId" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Node

OutputFiles Element

XML Representation

```
<xs:element name="OutputFiles">
  <xs:group ref="RUNTIME-FILES">
    <xs:sequence>
      <xs:element ref="DataFile"/>
      <xs:element ref="ContainerFile" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:group>
</xs:element>
```

Parent Elements

Execution, Module

Child Elements

ContainerFile, DataFile

Palette Element

Table 158. Attributes for Palette

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
id	**required**		*string*
label	**required**		*string*
labelKey	optional		*string*
position	optional		**atStart** **atEnd** **before** **after**
systemPalette	optional		**import** **fieldOp** **recordOp** **modeling** **dbModeling** **graph** **output** **export** **modeling.classification** **modeling.association** **modeling.segmentation** **modeling.auto**

XML Representation

```
<xs:element name="Palette">
  <xs:sequence>
    <xs:element ref="Icon"/>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="position" use="optional">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="atStart"/>
        <xs:enumeration value="atEnd"/>
        <xs:enumeration value="before"/>
        <xs:enumeration value="after"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="systemPalette" type="SYSTEM-PALETTE" use="optional">
    <xs:enumeration value="import"/>
    <xs:enumeration value="fieldOp"/>
    <xs:enumeration value="recordOp"/>
    <xs:enumeration value="modeling"/>
    <xs:enumeration value="dbModeling"/>
    <xs:enumeration value="graph"/>
    <xs:enumeration value="output"/>
    <xs:enumeration value="export"/>
    <xs:enumeration value="modeling.classification"/>
    <xs:enumeration value="modeling.association"/>
    <xs:enumeration value="modeling.segmentation"/>
    <xs:enumeration value="modeling.auto"/>
  </xs:attribute>
</xs:element>
```

Child Elements

Icon

Parameters Element

Configuration parameters from the extension node.

Table 159. Attributes for Parameters

Attribute	Use	Description	Valid Values
count	optional		*nonNegativeInteger*

XML Representation

```
<xs:element name="Parameters" type="PARAMETERS">
  <xs:sequence>
    <xs:element name="Parameter" type="PARAMETER" minOccurs="0" maxOccurs="unbounded">
      <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
        <xs:choice>
          <xs:element ref="MapValue"/>
          <xs:element ref="StructuredValue"/>
          <xs:element ref="ListValue"/>
          <xs:element ref="Value"/>
          <xs:element ref="DatabaseConnectionValue"/>
        </xs:choice>
      </xs:group>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="count" type="xs:nonNegativeInteger"/>
</xs:element>
```

Child Elements

Parameter

Parameter Element: A parameter has a name and a value. A simple value can be expressed with the value attribute; a compound value uses the content model described by ParameterContent. This combination of attribute and content is repeated for nested values.

Table 160. Attributes for Parameter

Attribute	Use	Description	Valid Values
name	**required**		*string*
value	optional		*string*

XML Representation

```
<xs:element name="Parameter" type="PARAMETER" minOccurs="0" maxOccurs="unbounded">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="value" type="xs:string"/>
</xs:element>
```

Parent Elements

Parameters

Child Elements

DatabaseConnectionValue, ListValue, MapValue, StructuredValue, Value

PasswordBoxControl Element

Table 161. Attributes for PasswordBoxControl

Attribute	Use	Description	Valid Values
columns	optional		*positiveInteger*
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="PasswordBoxControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
```

```
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="columns" type="xs:positiveInteger" use="optional" default="20"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

Properties Element

XML Representation

```
<xs:element name="Properties">
  <xs:sequence>
    <xs:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

DocumentOutput, Execution, InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput, Node

Child Elements

Property

PropertiesPanel Element

Table 162. Attributes for PropertiesPanel

Attribute	Use	Description	Valid Values
id	optional		*string*
label	optional		*string*
labelKey	optional		*string*

XML Representation

```
<xs:element name="PropertiesPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
```

```
    </xs:choice>
  </xs:sequence>
  <xs:sequence maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="CheckBoxControl"/>
      <xs:element ref="TextBoxControl"/>
      <xs:element ref="PasswordBoxControl"/>
      <xs:element ref="TextAreaControl"/>
      <xs:element ref="RadioButtonGroupControl"/>
      <xs:element ref="CheckBoxGroupControl"/>
      <xs:element ref="ComboBoxControl"/>
      <xs:element ref="SpinnerControl"/>
      <xs:element ref="ServerFileChooserControl"/>
      <xs:element ref="ServerDirectoryChooserControl"/>
      <xs:element ref="ClientFileChooserControl"/>
      <xs:element ref="ClientDirectoryChooserControl"/>
      <xs:element ref="TableControl"/>
      <xs:element ref="SingleFieldChooserControl"/>
      <xs:element ref="MultiFieldChooserControl"/>
      <xs:element ref="SingleFieldValueChooserControl"/>
      <xs:element ref="SingleItemChooserControl"/>
      <xs:element ref="MultiItemChooserControl"/>
      <xs:element ref="DBConnectionChooserControl"/>
      <xs:element ref="DBTableChooserControl"/>
      <xs:element ref="PropertyControl"/>
      <xs:element ref="StaticText"/>
      <xs:element ref="SystemControls"/>
      <xs:element ref="ActionButton"/>
      <xs:element ref="PropertiesPanel"/>
      <xs:element ref="PropertiesSubPanel"/>
      <xs:element ref="SelectorPanel"/>
      <xs:element ref="ExtensionObjectPanel"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="optional"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel, Tab

Child Elements

ActionButton, CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, ComboBoxControl, DBConnectionChooserControl, DBTableChooserControl, Enabled, ExtensionObjectPanel, Layout, MultiFieldChooserControl, MultiItemChooserControl, PasswordBoxControl, PropertiesPanel, PropertiesSubPanel, PropertyControl, RadioButtonGroupControl, SelectorPanel, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SingleItemChooserControl, SpinnerControl, StaticText, SystemControls, TableControl, TextAreaControl, TextBoxControl, Visible

Related Elements

PropertiesSubPanel

PropertiesSubPanel Element

Table 163. Attributes for PropertiesSubPanel

Attribute	Use	Description	Valid Values
buttonLabel	optional		*string*
buttonLabelKey	optional		*string*
dialogTitle	optional		*string*
dialogTitleKey	optional		*string*
helpLink	optional		*string*

Table 163. Attributes for PropertiesSubPanel (continued)

Attribute	Use	Description	Valid Values
mnemonic	optional		*string*
mnemonicKey	optional		*string*

XML Representation

```
<xs:element name="PropertiesSubPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:sequence maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="CheckBoxControl"/>
      <xs:element ref="TextBoxControl"/>
      <xs:element ref="PasswordBoxControl"/>
      <xs:element ref="TextAreaControl"/>
      <xs:element ref="RadioButtonGroupControl"/>
      <xs:element ref="CheckBoxGroupControl"/>
      <xs:element ref="ComboBoxControl"/>
      <xs:element ref="SpinnerControl"/>
      <xs:element ref="ServerFileChooserControl"/>
      <xs:element ref="ServerDirectoryChooserControl"/>
      <xs:element ref="ClientFileChooserControl"/>
      <xs:element ref="ClientDirectoryChooserControl"/>
      <xs:element ref="TableControl"/>
      <xs:element ref="SingleFieldChooserControl"/>
      <xs:element ref="MultiFieldChooserControl"/>
      <xs:element ref="SingleFieldValueChooserControl"/>
      <xs:element ref="SingleItemChooserControl"/>
      <xs:element ref="MultiItemChooserControl"/>
      <xs:element ref="DBConnectionChooserControl"/>
      <xs:element ref="DBTableChooserControl"/>
      <xs:element ref="PropertyControl"/>
      <xs:element ref="StaticText"/>
      <xs:element ref="SystemControls"/>
      <xs:element ref="ActionButton"/>
      <xs:element ref="PropertiesPanel"/>
      <xs:element ref="PropertiesSubPanel"/>
      <xs:element ref="SelectorPanel"/>
      <xs:element ref="ExtensionObjectPanel"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="buttonLabel" type="xs:string" use="optional"/>
  <xs:attribute name="buttonLabelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="dialogTitle" type="xs:string" use="optional"/>
  <xs:attribute name="dialogTitleKey" type="xs:string" use="optional"/>
  <xs:attribute name="helpLink" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

ActionButton, CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, ComboBoxControl, DBConnectionChooserControl, DBTableChooserControl, Enabled, ExtensionObjectPanel, Layout, MultiFieldChooserControl, MultiItemChooserControl, PasswordBoxControl, PropertiesPanel, PropertiesSubPanel, PropertyControl, RadioButtonGroupControl, SelectorPanel, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SingleItemChooserControl, SpinnerControl, StaticText, SystemControls, TableControl, TextAreaControl, TextBoxControl, Visible

Related Elements

PropertiesPanel

Property Element

Table 164. Attributes for Property

Attribute	Use	Description	Valid Values
defaultValue	optional		
defaultValueKey	optional		*string*
deprecatedScriptNames	optional		*string*
description	optional		*string*
descriptionKey	optional		*string*
isList	optional		*boolean*
label	optional		*string*
labelKey	optional		*string*
max	optional		*string*
min	optional		*string*
name	**required**		*string*
scriptName	optional		*string*
type	optional		*string*
valueType	optional		**string** **encryptedString** **fieldName** **integer** **double** **boolean** **date** **enum** **structure** **databaseConnection**

XML Representation

```
<xs:element name="Property">
  <xs:choice>
    <xs:element ref="DefaultValue" minOccurs="0"/>
  </xs:choice>
  <xs:attribute name="valueType" type="PROPERTY-VALUE-TYPE">
    <xs:enumeration value="string"/>
    <xs:enumeration value="encryptedString"/>
    <xs:enumeration value="fieldName"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="double"/>
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="enum"/>
    <xs:enumeration value="structure"/>
    <xs:enumeration value="databaseConnection"/>
  </xs:attribute>
  <xs:attribute name="isList" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="min" type="xs:string" use="optional"/>
  <xs:attribute name="max" type="xs:string" use="optional"/>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="scriptName" type="xs:string" use="optional"/>
  <xs:attribute name="deprecatedScriptNames" type="xs:string" use="optional"/>
  <xs:attribute name="type" type="xs:string" use="optional"/>
  <xs:attribute name="defaultValue" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="defaultValueKey" type="xs:string" use="optional"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
```

```
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Properties, PropertySets

Child Elements

DefaultValue

Related Elements

PropertyType

PropertyControl Element

Table 165. Attributes for PropertyControl

Attribute	Use	Description	Valid Values
controlClass	**required**		*string*
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="PropertyControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="controlClass" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

PropertyGroup Element

Table 166. Attributes for PropertyGroup

Attribute	Use	Description	Valid Values
label	optional		*string*
labelKey	optional		*string*
properties	**required**		*string*

XML Representation

```
<xs:element name="PropertyGroup">
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="properties" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

ExpertSettings, SimpleSettings

PropertySets Element

XML Representation

```
<xs:element name="PropertySets">
  <xs:sequence>
    <xs:element ref="Property" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

CommonObjects

Child Elements

Property

PropertyType Element

Table 167. Attributes for PropertyType

Attribute	Use	Description	Valid Values
id	**required**		*string*
isKeyed	optional		*boolean*
isList	optional		*boolean*
max	optional		*string*

Table 167. Attributes for PropertyType (continued)

Attribute	Use	Description	Valid Values
min	optional		*string*
valueType	optional		**string** **encryptedString** **fieldName** **integer** **double** **boolean** **date** **enum** **structure** **databaseConnection**

XML Representation

```
<xs:element name="PropertyType">
  <xs:choice>
    <xs:element ref="DefaultValue" minOccurs="0"/>
  </xs:choice>
  <xs:attribute name="valueType" type="PROPERTY-VALUE-TYPE">
    <xs:enumeration value="string"/>
    <xs:enumeration value="encryptedString"/>
    <xs:enumeration value="fieldName"/>
    <xs:enumeration value="integer"/>
    <xs:enumeration value="double"/>
    <xs:enumeration value="boolean"/>
    <xs:enumeration value="date"/>
    <xs:enumeration value="enum"/>
    <xs:enumeration value="structure"/>
    <xs:enumeration value="databaseConnection"/>
  </xs:attribute>
  <xs:attribute name="isList" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="min" type="xs:string" use="optional"/>
  <xs:attribute name="max" type="xs:string" use="optional"/>
  <xs:choice>
    <xs:element ref="Enumeration" minOccurs="0"/>
    <xs:element ref="Structure" minOccurs="0"/>
  </xs:choice>
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="isKeyed" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Parent Elements

PropertyTypes

Child Elements

DefaultValue, Enumeration, Structure

Related Elements

Property

PropertyTypes Element

XML Representation

```
<xs:element name="PropertyTypes">
  <xs:sequence>
    <xs:element ref="PropertyType" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

CommonObjects

Child Elements

PropertyType

RadioButtonGroupControl Element

Table 168. Attributes for RadioButtonGroupControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
falseLabel	optional		*string*
falseLabelKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
layoutByRow	optional		*boolean*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
rows	optional		*positiveInteger*
showLabel	optional		*boolean*
trueFirst	optional		*boolean*
trueLabel	optional		*string*
trueLabelKey	optional		*string*
useSubPanel	optional		*boolean*

XML Representation

```
<xs:element name="RadioButtonGroupControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="rows" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="layoutByRow" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="useSubPanel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="falseLabel" type="xs:string" use="optional"/>
  <xs:attribute name="falseLabelKey" type="xs:string" use="optional"/>
```

```
  <xs:attribute name="trueLabel" type="xs:string" use="optional"/>
  <xs:attribute name="trueLabelKey" type="xs:string" use="optional"/>
  <xs:attribute name="trueFirst" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

Range Element

Table 169. Attributes for Range

Attribute	Use	Description	Valid Values
max	optional		*string*
min	optional		*string*

XML Representation

```
<xs:element name="Range">
  <xs:attribute name="min" type="xs:string"/>
  <xs:attribute name="max" type="xs:string"/>
</xs:element>
```

Parent Elements

AddField, ChangeField, Field, Field, MissingValues, MissingValues, MissingValues

Range Element

Table 170. Attributes for Range

Attribute	Use	Description	Valid Values
maxValue	**required**		*string*
minValue	**required**		*string*

XML Representation

```
<xs:element name="Range" type="RANGE">
  <xs:attribute name="minValue" type="xs:string" use="required"/>
  <xs:attribute name="maxValue" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

AddField, ChangeField, Field, Field, MissingValues, MissingValues, MissingValues

RemoveField Element

Table 171. Attributes for RemoveField

Attribute	Use	Description	Valid Values
fieldRef	**required**		

XML Representation

```
<xs:element name="RemoveField">
  <xs:attribute name="fieldRef" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Parent Elements

ForEach, ModelFields

Resources Element

Defines common resources such as client-side libraries and resource bundles, and server-side libraries.

XML Representation

```
<xs:element name="Resources">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element name="Bundle" minOccurs="0">
      </xs:element>
      <xs:element name="JarFile" minOccurs="0">
      </xs:element>
      <xs:element name="SharedLibrary" minOccurs="0">
      </xs:element>
      <xs:element name="HelpInfo" minOccurs="0">
      </xs:element>
    </xs:choice>
  </xs:sequence>
</xs:element>
```

Parent Elements

Extension

Child Elements

Bundle, HelpInfo, JarFile, SharedLibrary

Bundle Element:

Table 172. Attributes for Bundle

Attribute	Use	Description	Valid Values
id	**required**		*string*
path	**required**		
type	**required**		**list** **properties**

XML Representation

```
<xs:element name="Bundle" minOccurs="0">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="type" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="list"/>
        <xs:enumeration value="properties"/>
      </xs:restriction>
```

```
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Parent Elements

Resources

JarFile Element:

Table 173. Attributes for JarFile

Attribute	Use	Description	Valid Values
id	**required**		*string*
path	**required**		

XML Representation

```
<xs:element name="JarFile" minOccurs="0">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Parent Elements

Resources

SharedLibrary Element:

Table 174. Attributes for SharedLibrary

Attribute	Use	Description	Valid Values
id	**required**		*string*
path	**required**		

XML Representation

```
<xs:element name="SharedLibrary" minOccurs="0">
  <xs:attribute name="id" type="xs:string" use="required"/>
  <xs:attribute name="path" type="EVALUATED-STRING" use="required"/>
</xs:element>
```

Parent Elements

Resources

HelpInfo Element:

Table 175. Attributes for HelpInfo

Attribute	Use	Description	Valid Values
default	optional		*string*
helpset	optional		
id	optional		*string*
missing	optional		*string*
path	optional		

Table 175. Attributes for HelpInfo (continued)

Attribute	Use	Description	Valid Values
type	**required**		**native** **javahelp** **html**

XML Representation

```
<xs:element name="HelpInfo" minOccurs="0">
  <xs:attribute name="id" type="xs:string" use="optional"/>
  <xs:attribute name="type" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="native"/>
        <xs:enumeration value="javahelp"/>
        <xs:enumeration value="html"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="path" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="helpset" type="EVALUATED-STRING" use="optional"/>
  <xs:attribute name="default" type="xs:string" use="optional"/>
  <xs:attribute name="missing" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Resources

Run Element

XML Representation

```
<xs:element name="Run">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
    <xs:element ref="Command" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="Option" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="StatusCodes" minOccurs="0"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

Executable

Child Elements

And, Command, Condition, Not, Option, Or, StatusCodes

SelectorPanel Element

Table 176. Attributes for SelectorPanel

Attribute	Use	Description	Valid Values
control	**required**		*string*

XML Representation

```
<xs:element name="SelectorPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="Selector">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="control" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Selector, Visible

Related Elements

ActionButton, ComboBoxControl, ExtensionObjectPanel, ModelViewerPanel, StaticText, SystemControls, TabbedPanel, TextBrowserPanel

Selector Element:

Table 177. Attributes for Selector

Attribute	Use	Description	Valid Values
controlValue	**required**		*string*
panelId	**required**		*string*

XML Representation

```
<xs:element name="Selector">
  <xs:attribute name="panelId" type="xs:string" use="required"/>
  <xs:attribute name="controlValue" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

SelectorPanel

ServerDirectoryChooserControl Element

Table 178. Attributes for ServerDirectoryChooserControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*

Table 178. Attributes for ServerDirectoryChooserControl (continued)

Attribute	Use	Description	Valid Values
mnemonicKey	optional		*string*
mode	**required**		**open** **save** **import** **export**
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="ServerDirectoryChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="mode" type="FILE-CHOOSER-MODE" use="required">
    <xs:enumeration value="open"/>
    <xs:enumeration value="save"/>
    <xs:enumeration value="import"/>
    <xs:enumeration value="export"/>
  </xs:attribute>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

ServerFileChooserControl Element

Table 179. Attributes for ServerFileChooserControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*

Table 179. Attributes for ServerFileChooserControl (continued)

Attribute	Use	Description	Valid Values
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
mode	**required**		**open** **save** **import** **export**
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="ServerFileChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="mode" type="FILE-CHOOSER-MODE" use="required">
    <xs:enumeration value="open"/>
    <xs:enumeration value="save"/>
    <xs:enumeration value="import"/>
    <xs:enumeration value="export"/>
  </xs:attribute>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

SetContainer Element

Table 180. Attributes for SetContainer

Attribute	Use	Description	Valid Values
source	**required**		*string*
target	**required**		*string*

XML Representation

```
<xs:element name="SetContainer">
  <xs:attribute name="source" type="xs:string" use="required"/>
  <xs:attribute name="target" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

Related Elements

SetProperty

SetProperty Element

Table 181. Attributes for SetProperty

Attribute	Use	Description	Valid Values
source	**required**		*string*
target	**required**		*string*

XML Representation

```
<xs:element name="SetProperty">
  <xs:attribute name="source" type="xs:string" use="required"/>
  <xs:attribute name="target" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

CreateDocumentOutput, CreateInteractiveDocumentBuilder, CreateInteractiveModelBuilder, CreateModelApplier, CreateModelOutput

Related Elements

SetContainer

SingleFieldChooserControl Element

Table 182. Attributes for SingleFieldChooserControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*

Table 182. Attributes for SingleFieldChooserControl (continued)

Attribute	Use	Description	Valid Values
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
onlyDatetime	optional		*boolean*
onlyDiscrete	optional		*boolean*
onlyNumeric	optional		*boolean*
onlyRanges	optional		*boolean*
onlySymbolic	optional		*boolean*
property	**required**		*string*
showLabel	optional		*boolean*
storage	optional		*string*
types	optional		*string*

XML Representation

```
<xs:element name="SingleFieldChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="storage" type="xs:string" use="optional"/>
  <xs:attribute name="onlyNumeric" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlySymbolic" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDatetime" type="xs:boolean" use="optional"/>
  <xs:attribute name="types" type="xs:string" use="optional"/>
  <xs:attribute name="onlyRanges" type="xs:boolean" use="optional"/>
  <xs:attribute name="onlyDiscrete" type="xs:boolean" use="optional"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

SingleFieldValueChooserControl Element

Table 183. Attributes for SingleFieldValueChooserControl

Attribute	Use	Description	Valid Values
description	optional		*string*
descriptionKey	optional		*string*
fieldControl	optional		*string*
fieldDirection	optional		**in** **out** **both** **none** **partition**
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="SingleFieldValueChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="fieldControl" type="xs:string" use="optional"/>
  <xs:attribute name="fieldDirection" type="FIELD-DIRECTION" use="optional">
    <xs:enumeration value="in"/>
    <xs:enumeration value="out"/>
    <xs:enumeration value="both"/>
    <xs:enumeration value="none"/>
    <xs:enumeration value="partition"/>
  </xs:attribute>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SpinnerControl, TableControl, TextAreaControl, TextBoxControl

SingleItemChooserControl Element

Table 184. Attributes for SingleItemChooserControl

Attribute	Use	Description	Valid Values
catalog	**required**		*string*
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="SingleItemChooserControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="catalog" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

MultiItemChooserControl

SpinnerControl Element

Table 185. Attributes for SpinnerControl

Attribute	Use	Description	Valid Values
columns	optional		*positiveInteger*
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
maxDecimalDigits	optional		*positiveInteger*
minDecimalDigits	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*
stepSize	optional		*decimal*

XML Representation

```
<xs:element name="SpinnerControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="columns" type="xs:positiveInteger" use="optional" default="5"/>
  <xs:attribute name="stepSize" type="xs:decimal" use="optional" default="1.0"/>
  <xs:attribute name="minDecimalDigits" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="maxDecimalDigits" type="xs:positiveInteger" use="optional"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl,

SingleFieldChooserControl, SingleFieldValueChooserControl, TableControl, TextAreaControl, TextBoxControl

SPSSDataFormat Element

XML Representation

```
<xs:element name="SPSSDataFormat"/>
```

Parent Elements

DataFormat

StaticText Element

Table 186. Attributes for StaticText

Attribute	Use	Description	Valid Values
text	optional		*string*
textKey	optional		*string*

XML Representation

```
<xs:element name="StaticText">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="text" type="xs:string" use="optional"/>
  <xs:attribute name="textKey" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

ActionButton, ComboBoxControl, ExtensionObjectPanel, ModelViewerPanel, SelectorPanel, SystemControls, TabbedPanel, TextBrowserPanel

StatusCode Element

Table 187. Attributes for StatusCode

Attribute	Use	Description	Valid Values
code	**required**		*integer*
message	optional		*string*
messageKey	optional		*string*
status	optional		**success** **warning** **error**

XML Representation

```
<xs:element name="StatusCode">
  <xs:attribute name="code" type="xs:integer" use="required"/>
  <xs:attribute name="status" use="optional" default="success">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="success"/>
        <xs:enumeration value="warning"/>
        <xs:enumeration value="error"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="message" type="xs:string" use="optional"/>
  <xs:attribute name="messageKey" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

StatusCodes

StatusCodes Element

Table 188. Attributes for StatusCodes

Attribute	Use	Description	Valid Values
defaultMessage	optional		*string*
defaultMessageKey	optional		*string*

XML Representation

```
<xs:element name="StatusCodes">
  <xs:sequence>
    <xs:element ref="StatusCode" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="defaultMessage" type="xs:string" use="optional"/>
  <xs:attribute name="defaultMessageKey" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Module, Run

Child Elements

StatusCode

StatusDetail Element

Supplementary information about a progress or other conditions.

Table 189. Attributes for StatusDetail

Attribute	Use	Description	Valid Values
destination	optional		**client** **tracefile** **console**

XML Representation

```
<xs:element name="StatusDetail" type="STATUS-DETAIL">
  <xs:sequence>
    <xs:element name="Diagnostic" type="DIAGNOSTIC" minOccurs="0" maxOccurs="unbounded">
      <xs:sequence>
        <xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
        </xs:element>
        <xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
```

```
      </xs:sequence>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="destination" type="STATUS-DESTINATION" default="client">
    <xs:enumeration value="client"/>
    <xs:enumeration value="tracefile"/>
    <xs:enumeration value="console"/>
  </xs:attribute>
</xs:element>
```

Child Elements

Diagnostic

Diagnostic Element:

Table 190. Attributes for Diagnostic

Attribute	Use	Description	Valid Values
code	**required**		*integer*
severity	optional		**unknown** **information** **warning** **error** **fatal**
source	optional		*string*
subCode	optional		*integer*

XML Representation

```
<xs:element name="Diagnostic" type="DIAGNOSTIC" minOccurs="0" maxOccurs="unbounded">
  <xs:sequence>
    <xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
    </xs:element>
    <xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="code" type="xs:integer" use="required"/>
  <xs:attribute name="subCode" type="xs:integer" default="0"/>
  <xs:attribute name="severity" type="DIAGNOSTIC-SEVERITY" default="error">
    <xs:enumeration value="unknown"/>
    <xs:enumeration value="information"/>
    <xs:enumeration value="warning"/>
    <xs:enumeration value="error"/>
    <xs:enumeration value="fatal"/>
  </xs:attribute>
  <xs:attribute name="source" type="xs:string"/>
</xs:element>
```

Parent Elements

StatusDetail

Child Elements

Message, Parameter

Message Element:

Table 191. Attributes for Message

Attribute	Use	Description	Valid Values
lang	optional		*NMTOKEN*

XML Representation

```
<xs:element name="Message" type="DIAGNOSTIC-MESSAGE" minOccurs="0">
  <xs:attribute name="lang" type="xs:NMTOKEN"/>
</xs:element>
```

Parent Elements

Diagnostic

Parameter Element:

XML Representation

```
<xs:element name="Parameter" type="xs:string" minOccurs="0" maxOccurs="unbounded"/>
```

Parent Elements

Diagnostic

Structure Element

XML Representation

```
<xs:element name="Structure">
  <xs:sequence>
    <xs:element ref="Attribute" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:element>
```

Parent Elements

PropertyType

Child Elements

Attribute

StructuredValue Element

A sequence of named values ("attributes").

XML Representation

```
<xs:element name="StructuredValue" type="STRUCTURED-VALUE">
  <xs:sequence>
    <xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
      <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
        <xs:choice>
          <xs:element ref="MapValue"/>
          <xs:element ref="StructuredValue"/>
          <xs:element ref="ListValue"/>
          <xs:element ref="Value"/>
          <xs:element ref="DatabaseConnectionValue"/>
        </xs:choice>
      </xs:group>
      <xs:sequence>
        <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
          <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
            <xs:choice>
              <xs:element ref="MapValue"/>
              <xs:element ref="StructuredValue"/>
              <xs:element ref="ListValue"/>
              <xs:element ref="Value"/>
              <xs:element ref="DatabaseConnectionValue"/>
            </xs:choice>
          </xs:group>
        </xs:element>
      </xs:sequence>
    </xs:element>
  </xs:sequence>
</xs:element>
```

Parent Elements

Attribute, Attribute, ListValue, ListValue, ListValue, Parameter

Child Elements

Attribute

Attribute Element:

Table 192. Attributes for Attribute

Attribute	Use	Description	Valid Values
name	**required**		*string*
value	optional		*string*

XML Representation

```
<xs:element name="Attribute" type="ATTRIBUTE" maxOccurs="unbounded">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
  <xs:sequence>
    <xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
      <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
        <xs:choice>
          <xs:element ref="MapValue"/>
          <xs:element ref="StructuredValue"/>
          <xs:element ref="ListValue"/>
          <xs:element ref="Value"/>
          <xs:element ref="DatabaseConnectionValue"/>
        </xs:choice>
      </xs:group>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required"/>
  <xs:attribute name="value" type="xs:string"/>
</xs:element>
```

Parent Elements

StructuredValue

Child Elements

DatabaseConnectionValue, ListValue, ListValue, MapValue, StructuredValue, Value

ListValue Element: A sequence of values. All values must have the same content type but this is not checked.

XML Representation

```
<xs:element name="ListValue" type="LIST-VALUE" minOccurs="0" maxOccurs="1">
  <xs:group ref="PARAMETER-CONTENT" minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="MapValue"/>
      <xs:element ref="StructuredValue"/>
      <xs:element ref="ListValue"/>
      <xs:element ref="Value"/>
      <xs:element ref="DatabaseConnectionValue"/>
    </xs:choice>
  </xs:group>
</xs:element>
```

Parent Elements

Attribute

Child Elements

DatabaseConnectionValue, ListValue, MapValue, StructuredValue, Value

SystemControls Element

Table 193. Attributes for SystemControls

Attribute	Use	Description	Valid Values
controlsId	**required**		*string*

XML Representation

```
<xs:element name="SystemControls">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="controlsId" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

ActionButton, ComboBoxControl, ExtensionObjectPanel, ModelViewerPanel, SelectorPanel, StaticText, TabbedPanel, TextBrowserPanel

Tab Element

Table 194. Attributes for Tab

Attribute	Use	Description	Valid Values
helpLink	optional		*string*
id	optional		*string*
label	**required**		*string*
labelKey	optional		*string*
mnemonic	optional		*string*
mnemonicKey	optional		*string*

XML Representation

```
<xs:element name="Tab">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:choice>
      <xs:element ref="PropertiesPanel"/>
      <xs:element ref="ExtensionObjectPanel"/>
      <xs:element ref="TextBrowserPanel"/>
```

```
      <xs:element ref="ModelViewerPanel"/>
      <xs:element ref="TabbedPanel"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="id" type="xs:string" use="optional"/>
  <xs:attribute name="label" type="xs:string" use="required"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="helpLink" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

Tabs

Child Elements

ExtensionObjectPanel, ModelViewerPanel, PropertiesPanel, TabbedPanel, TextBrowserPanel

TabbedPanel Element

Table 195. Attributes for TabbedPanel

Attribute	Use	Description	Valid Values
style	optional		**standard** **sidebar**

XML Representation

```
<xs:element name="TabbedPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:sequence maxOccurs="unbounded">
    <xs:element ref="Tabs"/>
  </xs:sequence>
  <xs:attribute name="style" use="optional">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="standard"/>
        <xs:enumeration value="sidebar"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>
```

Parent Elements

Tab

Child Elements

Enabled, Layout, Tabs, Visible

Related Elements

ActionButton, ComboBoxControl, ExtensionObjectPanel, ModelViewerPanel, SelectorPanel, StaticText, SystemControls, TextBrowserPanel

TableControl Element

Table 196. Attributes for TableControl

Attribute	Use	Description	Valid Values
columns	optional		*positiveInteger*
columnWidths	optional		*string*
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
rows	optional		*positiveInteger*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="TableControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="rows" type="xs:positiveInteger" use="optional" default="8"/>
  <xs:attribute name="columns" type="xs:positiveInteger" use="optional" default="20"/>
  <xs:attribute name="columnWidths" type="xs:string" use="optional"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TextAreaControl, TextBoxControl

Tabs Element

Table 197. Attributes for Tabs

Attribute	Use	Description	Valid Values
defaultTab	optional		*nonNegativeInteger*

XML Representation

```
<xs:element name="Tabs">
  <xs:sequence>
    <xs:element ref="Tab" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="defaultTab" type="xs:nonNegativeInteger" use="optional" default="0"/>
</xs:element>
```

Parent Elements

TabbedPanel, UserInterface

Child Elements

Tab

TextAreaControl Element

Table 198. Attributes for TextAreaControl

Attribute	Use	Description	Valid Values
columns	optional		*positiveInteger*
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
rows	optional		*positiveInteger*
showLabel	optional		*boolean*
wrapLines	optional		*boolean*

XML Representation

```
<xs:element name="TextAreaControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
```

```
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="rows" type="xs:positiveInteger" use="optional" default="8"/>
  <xs:attribute name="columns" type="xs:positiveInteger" use="optional" default="20"/>
  <xs:attribute name="wrapLines" type="xs:boolean" use="optional" default="true"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextBoxControl

TextBoxControl Element

Table 199. Attributes for TextBoxControl

Attribute	Use	Description	Valid Values
columns	optional		*positiveInteger*
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="TextBoxControl">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="property" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="label" type="xs:string" use="optional"/>
  <xs:attribute name="labelKey" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonic" type="xs:string" use="optional"/>
  <xs:attribute name="mnemonicKey" type="xs:string" use="optional"/>
  <xs:attribute name="labelWidth" type="xs:positiveInteger" use="optional" default="1"/>
  <xs:attribute name="labelAbove" type="xs:boolean" use="optional" default="false"/>
```

```
  <xs:attribute name="description" type="xs:string" use="optional"/>
  <xs:attribute name="descriptionKey" type="xs:string" use="optional"/>
  <xs:attribute name="columns" type="xs:positiveInteger" use="optional" default="20"/>
</xs:element>
```

Parent Elements

PropertiesPanel, PropertiesSubPanel

Child Elements

Enabled, Layout, Visible

Related Elements

CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, DBConnectionChooserControl, DBTableChooserControl, MultiFieldChooserControl, PasswordBoxControl, PropertyControl, RadioButtonGroupControl, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SpinnerControl, TableControl, TextAreaControl

TextBrowserPanel Element

Table 200. Attributes for TextBrowserPanel

Attribute	Use	Description	Valid Values
columns	optional		*string*
container	**required**		*string*
rows	optional		*string*
textFormat	**required**		**plainText** **html** **rtf**
wrapLines	optional		*boolean*

XML Representation

```
<xs:element name="TextBrowserPanel">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
  <xs:attribute name="container" type="xs:string" use="required"/>
  <xs:attribute name="textFormat" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="plainText"/>
        <xs:enumeration value="html"/>
        <xs:enumeration value="rtf"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="rows" type="xs:string" use="optional"/>
  <xs:attribute name="columns" type="xs:string" use="optional"/>
  <xs:attribute name="wrapLines" type="xs:boolean" use="optional" default="false"/>
</xs:element>
```

Parent Elements

Tab

Child Elements

Enabled, Layout, Visible

Related Elements

ActionButton, ComboBoxControl, ExtensionObjectPanel, ModelViewerPanel, SelectorPanel, StaticText, SystemControls, TabbedPanel

ToolbarItem Element

Table 201. Attributes for ToolbarItem

Attribute	Use	Description	Valid Values
action	**required**		*string*
offset	optional		*nonNegativeInteger*
separatorAfter	optional		*boolean*
separatorBefore	optional		*boolean*
showIcon	optional		*boolean*
showLabel	optional		*boolean*

XML Representation

```
<xs:element name="ToolbarItem">
  <xs:attribute name="action" type="xs:string" use="required"/>
  <xs:attribute name="showLabel" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="showIcon" type="xs:boolean" use="optional" default="true"/>
  <xs:attribute name="separatorBefore" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="separatorAfter" type="xs:boolean" use="optional" default="false"/>
  <xs:attribute name="offset" type="xs:nonNegativeInteger" use="optional" default="0"/>
</xs:element>
```

Parent Elements

Controls

UserInterface Element

Table 202. Attributes for UserInterface

Attribute	Use	Description	Valid Values
actionHandler	optional		*any*
frameClass	optional		*any*

XML Representation

```
<xs:element name="UserInterface">
  <xs:sequence>
    <xs:element ref="Icons" minOccurs="0"/>
    <xs:element ref="Controls" minOccurs="0"/>
    <xs:element ref="Tabs" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="frameClass" use="optional"/>
  <xs:attribute name="actionHandler" use="optional"/>
</xs:element>
```

Parent Elements

DocumentOutput, Extension, InteractiveDocumentBuilder, InteractiveModelBuilder, ModelOutput, Node

Child Elements

Controls, Icons, Tabs

UTF8Format Element

XML Representation

```
<xs:element name="UTF8Format"/>
```

Parent Elements

FileFormatType

Value Element

A simple value.

Table 203. Attributes for Value

Attribute	Use	Description	Valid Values
value	**required**		*string*

XML Representation

```
<xs:element name="Value" type="SIMPLE-VALUE">
  <xs:attribute name="value" type="xs:string" use="required"/>
</xs:element>
```

Parent Elements

Attribute, Attribute, ListValue, ListValue, ListValue, Parameter

Values Element

XML Representation

```
<xs:element name="Values">
  <xs:sequence>
    <xs:element name="Value" minOccurs="0" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
</xs:element>
```

Parent Elements

AddField, ChangeField, Field, Field, MissingValues, MissingValues, MissingValues

Child Elements

Value

Value Element:

Table 204. Attributes for Value

Attribute	Use	Description	Valid Values
flagProperty	optional		**trueValue** **falseValue**
value	**required**		*string*
valueLabel	optional		*string*

XML Representation

```
<xs:element name="Value" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="valueLabel" type="xs:string" use="optional"/>
  <xs:attribute name="flagProperty">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="trueValue"/>
        <xs:enumeration value="falseValue"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
</xs:element>
```

Parent Elements

Values

Values Element

Table 205. Attributes for Values

Attribute	Use	Description	Valid Values
code	**required**		*integer*
displayLabel	optional		*string*
flagValue	optional		*boolean*
value	**required**		*string*

XML Representation

```
<xs:element name="Values" type="FIELD-VALUE">
  <xs:sequence>
    <xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
    </xs:element>
  </xs:sequence>
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="code" type="xs:integer" use="required"/>
  <xs:attribute name="flagValue" type="xs:boolean"/>
  <xs:attribute name="displayLabel" type="xs:string"/>
</xs:element>
```

Parent Elements

AddField, ChangeField, Field, Field, MissingValues, MissingValues, MissingValues

Child Elements

DisplayLabel

DisplayLabel Element: A display label for a field or value in a specified language. The displayLabel attribute can be used where there is no label for a particular language.

Table 206. Attributes for DisplayLabel

Attribute	Use	Description	Valid Values
lang	optional		*NMTOKEN*
value	**required**		*string*

XML Representation

```
<xs:element name="DisplayLabel" type="DISPLAY-LABEL" minOccurs="0" maxOccurs="unbounded">
  <xs:attribute name="value" type="xs:string" use="required"/>
  <xs:attribute name="lang" type="xs:NMTOKEN" default="en"/>
</xs:element>
```

Parent Elements

Values

Visible Element

XML Representation

```
<xs:element name="Visible">
  <xs:sequence>
    <xs:group ref="CONDITION-EXPRESSION" minOccurs="0">
      <xs:choice>
        <xs:element ref="Condition"/>
        <xs:element ref="And"/>
        <xs:element ref="Or"/>
        <xs:element ref="Not"/>
      </xs:choice>
    </xs:group>
  </xs:sequence>
</xs:element>
```

Parent Elements

ActionButton, CheckBoxControl, CheckBoxGroupControl, ClientDirectoryChooserControl, ClientFileChooserControl, ComboBoxControl, DBConnectionChooserControl, DBTableChooserControl, ExtensionObjectPanel, ItemChooserControl, ModelViewerPanel, MultiFieldChooserControl, MultiItemChooserControl, PasswordBoxControl, PropertiesPanel, PropertiesSubPanel, PropertyControl, RadioButtonGroupControl, SelectorPanel, ServerDirectoryChooserControl, ServerFileChooserControl, SingleFieldChooserControl, SingleFieldValueChooserControl, SingleItemChooserControl, SpinnerControl, StaticText, SystemControls, TabbedPanel, TableControl, TextAreaControl, TextBoxControl, TextBrowserPanel

Child Elements

And, Condition, Not, Or

Extended Types

Extended types extend elements in an XML document by adding attributes and child elements. To use an extended type in an XML document, you specify the extended type with the xsi:type attribute for the element. Then you can use the attributes and elements defined by the extended type.

ItemChooserControl Type

Table 207. Attributes for ItemChooserControl

Attribute	Use	Description	Valid Values
catalog	**required**		*string*
description	optional		*string*
descriptionKey	optional		*string*
label	optional		*string*
labelAbove	optional		*boolean*
labelKey	optional		*string*
labelWidth	optional		*positiveInteger*
mnemonic	optional		*string*
mnemonicKey	optional		*string*
property	**required**		*string*
showLabel	optional		*boolean*

XML Representation

```
<xs:complexType name="ItemChooserControl" mixed="false">
  <xs:sequence>
    <xs:choice>
      <xs:element ref="Layout" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Enabled" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="Visible" minOccurs="0" maxOccurs="1"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

Extends

ComboBoxControl

Child Elements

Enabled, Layout, Visible

Related Types

ItemChooserControl

Notices

This information was developed for products and services offered worldwide.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Software Group
ATTN: Licensing
200 W. Madison St.
Chicago, IL; 60606
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other product and service names might be trademarks of IBM or other companies.

Index

D

E

F

G

H

P

R

S

T

U

V

X

Printed in USA