**SAP**®

# Contents

# CHAPTER 1 **Getting Started with XML Modeling**

An *XML model (XSM)* helps you analyze an XML Schema Definition (.XSD), Document Type Definition (.DTD) or XML-Data Reduced (.XDR) file. You can model, reverse-engineer, and generate each of these file formats.

Since XML structures can be very complex, it can be easier to visualize them through diagrams. With its Browser tree view and diagram, an SAP® Sybase® PowerDesigner® XSM gives you a global and schematic view of all the elements composing your XSD, DTD, or XDR:
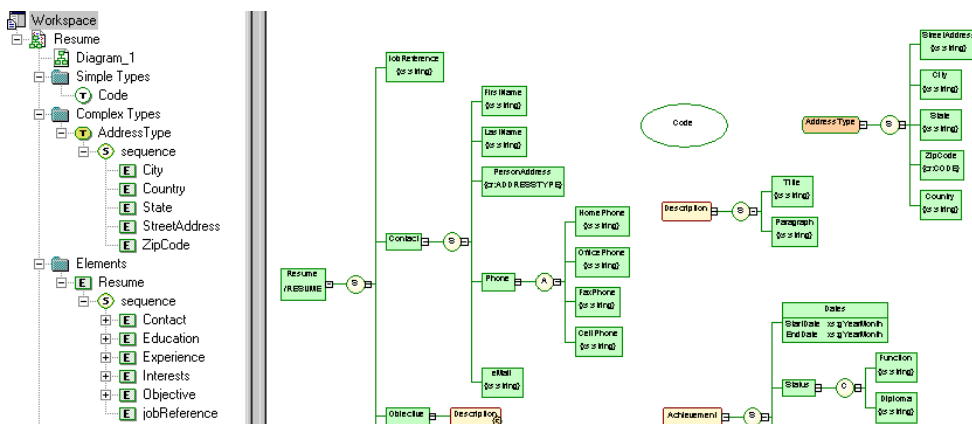


A PowerDesigner XSM allows you to generate and reverse engineer XSD, DTD and XDR files and also generate an XML model from a Physical Data Model (PDM), Object Oriented Model (OOM), or another XSM:

R.E: Reverse Engineering
G: Generation

*DTD, XSD or XDR*

The structure of an XSM is described by a DTD, an XSD or an XDR file:

• A DTD file is a basic way to describe the structure of an XML document. It is a raw list of all the legal elements making up an XML document. An extract of a DTD file follows:

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

<!ELEMENT Database (DIVISION,EMPLOYEE,CUSTOMER,PROJECT,
TEAM,MATERIAL,PARTICIPATE,MEMBER,USED,COMPOSE)>
<!ELEMENT DIVISION EMPTY>
<!ATTLIST DIVISION
            DIVNUM                          CDATA
            DIVNAME                         CDATA
            DIVADDR                         CDATA>
<!ELEMENT EMPLOYEE EMPTY>
<!ATTLIST EMPLOYEE
            EMPNUM                          CDATA
            EMP_EMPNUM                      CDATA
            DIVNUM                          CDATA
            EMPFNAM                         CDATA
            EMPLNAM                         CDATA
            EMPFUNC                         CDATA
            EMPSAL                          CDATA>
```

• An XSD file (or schema) is an elaborated way to describe the structure of an XML document. It can support namespaces, derivations, keys, simple and complex user-defined data types and a robust collection of predefined data types. An extract of an XSD file follows:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!--
Project Management
-->
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="Database">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="DIVISION">
                    <xs:complexType>
                        <xs:attribute name="DIVNUM">
                            <xs:simpleType>
                              <xs:restriction base="ID">
                                  <xs:minInclusive value="1"/>
                                  <xs:pattern value="00000"/>
                              </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                        <xs:attribute name="DIVNAME" type="NAME">
                        </xs:attribute>
                        <xs:attribute name="DIVADDR" type="SHORT_TEXT">
                        </xs:attribute>
                    </xs:complexType>
                </xs:element>
```

An XSD file always starts with the <schema> tag (root element). All objects created in the model will appear in the XSD file between the schema start-tag and end-tag

- An XDR file is a simplified XSD file (or schema). It does not support simple and complex user-defined data types. An extract of an XDR file follows:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<Schema name="PROJECT"
xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <description>Project Management</description>
  <ElementType name="DIVISION" content="empty">
      <AttributeType name="DIVNUM"/>
      <attribute type="DIVNUM"/>
      <AttributeType name="DIVNAME" dt:type="string"/>
      <attribute type="DIVNAME"/>
      <AttributeType name="DIVADDR" dt:type="string"/>
      <attribute type="DIVADDR"/>
  </ElementType>
```

An XDR file always starts with the <schema> tag (root element). All objects created in the model will appear in the XDR file between the schema start-tag and end-tag

*Suggested Bibliography*

- W3C XML Recommendation – *http://www.w3.org/TR/REC-xml*
- W3C DTD Recommendation – *http://www.w3.org/TR/REC-xml#dt-doctype*
- W3C XML Schema Recommendation – *http://www.w3.org/XML/Schema#dev*
- W3C XML-Data Note – *http://www.w3.org/TR/1998/NOTE-XML-data-0105/*

# Creating an XSM

You create a new XML model by selecting **File > New Model**.

**Note:** In addition to creating an XSM from scratch with the following procedure, you can also reverse-engineer a model from an existing XSD, a DTD or an XDR file (see *Reverse Engineering an XML Schema into an XSM* on page 67).

The New Model dialog is highly configurable, and your administrator may hide options that are not relevant for your work or provide templates or predefined models to guide you through model creation. When you open the dialog, one or more of the following buttons will be available on the left hand side:

- **Categories** - which provides a set of predefined models and diagrams sorted in a configurable category structure.
- **Model types** - which provides the classic list of PowerDesigner model types and diagrams.
- **Template files** - which provides a set of model templates sorted by model type.
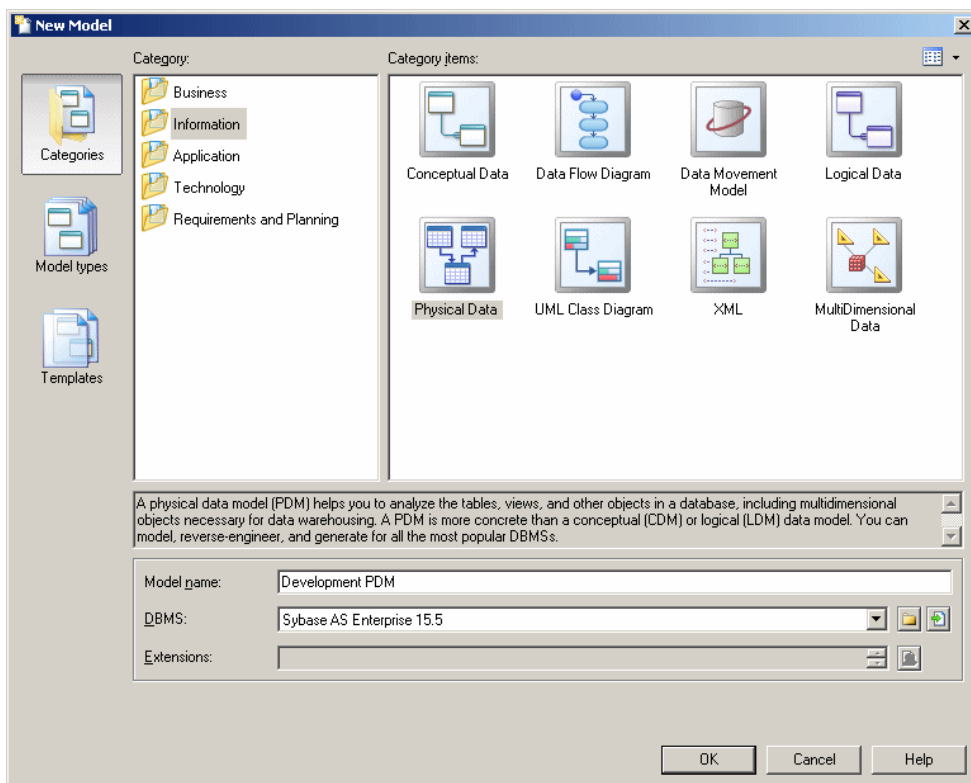
1. Select **File > New Model** to open the New Model dialog.

2. Click a button, and then select a category or model type ( **XML Model** ) in the left-hand pane.

3. Select an item in the right-hand pane. Depending on how your New Model dialog is configured, these items may be first diagrams or templates on which to base the creation of your model.

   Use the **Views** tool on the upper right hand side of the dialog to control the display of the items.

4. Enter a model name. The code of the model, which is used for script or code generation, is derived from this name using the model naming conventions.

5. Select a target XML language , which customizes PowerDesigner's default modifying environment with target-specific properties, objects, and generation templates.

   By default, PowerDesigner creates a link in the model to the specified file. To copy the contents of the resource and save it in your model file, click the **Embed Resource in Model** button to the right of this field. Embedding a file in this way enables you to make changes specific to your model without affecting any other models that reference the shared resource.

6. [optional] Click the **Select Extensions** button and attach one or more extensions to your model.

7. Click **OK** to create and open the XML model .

   **Note:** Sample XSMs are available in the Example Directory.

## XSM Properties

You open the model property sheet by right-clicking the model in the Browser and selecting **Properties**.

Each XML model has the following model properties:

| Property | Description |
|---|---|
| Name/Code/Comment | Identify the model. The name should clearly convey the model's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. You can optionally add a comment to provide more detailed information about the model. By default the code is auto-generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field. |
| Filename | Specifies the location of the model file. This box is empty if the model has never been saved. |

| Property | Description |
|---|---|
| Author | Specifies the author of the model. If you enter nothing, the Author field in diagram title boxes displays the user name from the model property sheet Version Info tab. If you enter a space, the Author field displays nothing. |
| Version | Specifies the version of the model. You can use this box to display the repository version or a user defined version of the model. This parameter is defined in the display preferences of the Title node. |
| XML language | Specifies the model target. |
| Default diagram | Specifies the diagram displayed by default when you open the model. |
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas. |

The following tabs are also available:

- **Detail** - [XSD only] Contains the following properties:

| Property | Description |
|---|---|
| Target Namespace | Specifies a URI as the namespace for all the model objects. All the schema elements with this prefix in their start-tag will be associated with the namespace. For example: http://www.mycompany.com/myproduct/XMLmodel |
| Language | Specifies the language used in the model. For example: en, en-GB, en-US, de, fr |
| ID | Specifies the ID of the model. Its value must be of type ID and unique within the file containing the model. For example: XMOD1 |
| Default | Specifies defaults for the **Form** and **Block** and **Final** model object properties. |

- **Items** - lists the model's global objects (which have no parent symbol in the diagram, and are directly linked to the <schema> tag). The list reflects the order in which global objects are declared in the schema. You can change the order of declaration by selecting an item in the list and using the arrowed buttons, at the bottom-left corner of the tab, to move it in the list. The following tools are available on this tab:

| Tool | Description |
|---|---|
|  | Add Element |
|  | Add Group |
|  | Add Attribute |

| Tool | Description |
|------|-------------|
| | Add Attribute Group |
| | Add Simple Type [XSD only] |
| | Add Complex Type [XSD only] |
| | Add Notation |
| | Add Annotation [XSD only] |

- **External Schemas** - [XSD only] Allows you to link to and reuse in your model global objects from other schemas. The following tools are available on this tab:

| Tool | Description |
|------|-------------|
| | Add Include |
| | Add Import |
| | Add Redefine |
| | Add Annotation |

- **Namespaces** - [XSD and XDR only] Lists the namespaces used to declare objects used in the model.
- **Preview** - Displays a preview of the XSD, DTD or XDR file generated from the XSM.

# Previewing XML Code

Click the **Preview** tab in the property sheet of the model, elements, and various other model objects in order to view the code that will be generated for it.
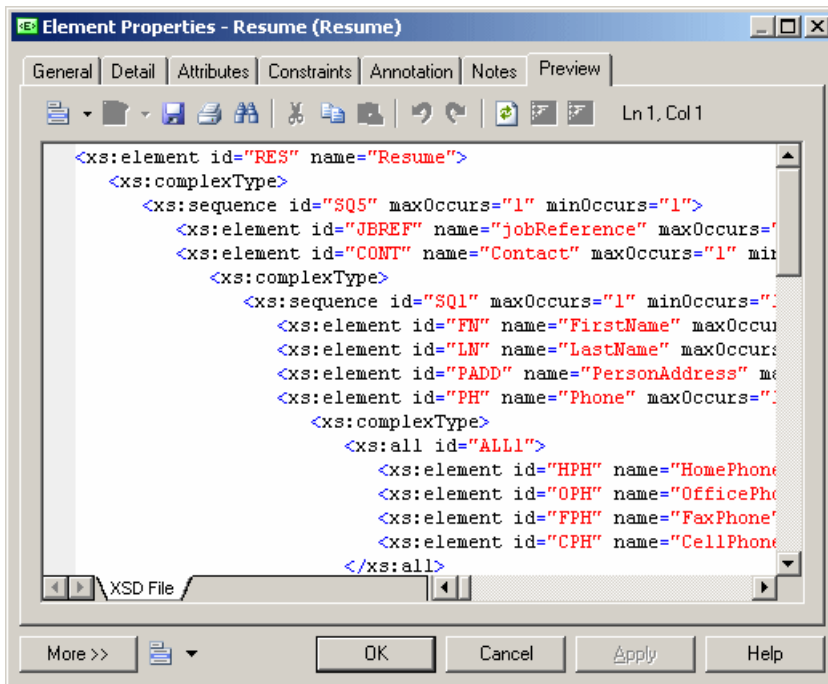


The following tools are available on the **Preview** tab toolbar:

| Tools | Description |
|---|---|
|  | **Editor Menu [Shift+F11]** - Contains the following commands:<br><br>• **New [Ctrl+N]** - Reinitializes the field by removing all the existing content.<br>• **Open... [Ctrl+O]** - Replaces the content of the field with the content of the selected file.<br>• **Insert... [Ctrl+I]** - Inserts the content of the selected file at the cursor.<br>• **Save [Ctrl+S]** - Saves the content of the field to the specified file.<br>• **Save As...** - Saves the content of the field to a new file.<br>• **Select All [Ctrl+A]** - Selects all the content of the field.<br>• **Find... [Ctrl+F]** - Opens a dialog to search for text in the field.<br>• **Find Next... [F3]** - Finds the next occurence of the searched for text.<br>• **Find Previous... [Shift+F3]** - Finds the previous occurence of the searched for text.<br>• **Replace... [Ctrl+H]** - Opens a dialog to replace text in the field.<br>• **Go To Line... [Ctrl+G]** - Opens a dialog to go to the specified line.<br>• **Toggle Bookmark [Ctrl+F2]** Inserts or removes a bookmark (a blue box) at the cursor position. Note that bookmarks are not printable and are lost if you refresh the tab<br>• **Next Bookmark [F2]** - Jumps to the next bookmark.<br>• **Previous Bookmark [Shift+F2]** - Jumps to the previous bookmark. |
|  | **Edit With [Ctrl+E]** - Opens the previewed code in an external editor. Click the down arrow to select a particular editor or **Choose Program** to specify a new editor. Editors specified here are added to the list of editors available at **Tools > General Options > Editors**. |
|  | **Save [Ctrl+S]** - Saves the content of the field to the specified file. |
|  | **Print [Ctrl+P]** - Prints the content of the field. |
|  | **Find [Ctrl+F]** - Opens a dialog to search for text. |
|  | **Cut [Ctrl+X]**, **Copy [Ctrl+C]**, and **Paste [Ctrl+V]** - Perform the standard clipboard actions. |
|  | **Undo [Ctrl+Z]** and **Redo [Ctrl+Y]** - Move backward or forward through edits. |
|  | **Refresh [F5]** - Refreshes the Preview tab.<br><br>You can debug the GTL templates that generate the code shown in the Preview tab. To do so, open the target or extension resource file, select the **Enable Trace Mode** option, and click **OK** to return to your model. You may need to click the **Refresh** tool to display the templates. |

| Tools | Description |
|---|---|
|  | **Select Generation Targets [Ctrl+F6]** - Lets you select additional generation targets (defined in extensions), and adds a sub-tab for each selected target. For information about generation targets, see *Customizing and Extending PowerDesigner > Extension Files > Generated Files (Profile) > Generating Your Files in a Standard or Extended Generation*. |

# Customizing your Modeling Environment

The PowerDesigner XML model provides various means for customizing and controlling your modeling environment.

## Setting Model Options

You can set XSM model options by selecting **Tools > Model Options** or right-clicking the diagram background and selecting **Model Options**.

You can set the following options on the Model Settings page:

| Option | Description |
|---|---|
| Name/Code case sensitive | Specifies that the names and codes for all objects are case sensitive, allowing you to have two objects with identical names or codes but different cases in the same model. If you change case sensitivity during the design process, we recommend that you check your model to verify that your model does not contain any duplicate objects. |
| Enable links to requirements | Displays a Requirements tab in the property sheet of every object in the model, which allows you to attach requirements to objects (see *Requirements Modeling*). |
| Generate tables as | Specifies how PDM tables are generated to the XSM during model-to-model generation or when creating objects through drag-and-drop in the Mapping Editor. You can choose between:<br><br>• Elements - [default] each table is generated as an untyped element directly linked to its columns generated as attributes or sub-elements.<br>• Elements with complex types - each table is generated as an element typed by a complex type, generated in parallel, to contain the columns. |

| Option | Description |
|---|---|
| Generate columns as | Specifies how PDM columns are generated to the XSM during model-to-model generation or when creating objects through drag-and-drop in the Mapping Editor. You can choose between:<br><br>• Elements - [default] each column is generated as an sub-element of its table element or complex type.<br>• Attributes - each column is generated as an attribute of its table element or complex type. |

For information about controlling the naming conventions of your models, see *Core Features Guide > Modeling with PowerDesigner > Objects > Naming Conventions*.

## Setting XSM Display Preferences

PowerDesigner display preferences allow you to customize the format of object symbols, and the information that is displayed on them. To set XML model display preferences, select **Tools > Display Preferences** or right-click the diagram background and select **Display Preferences** from the contextual menu.

For detailed information about customizing and controlling the attributes and collections displayed on object symbols, see *Core Features Guide > Modeling with PowerDesigner > Diagrams, Matrices, and Symbols > Display Preferences*.

## Viewing and Editing the XML Language Definition File

Each XSM is linked to a definition file that extends the standard PowerDesigner metamodel to provide objects, properties, data types, and generation parameters and templates specific to the language being modeled. Definition files and other resource files are XML files located in the Resource Files directory inside your installation directory, and can be opened and edited in the PowerDesigner Resource Editor.

**Warning!** The resource files provided with PowerDesigner inside the Program Files folder cannot be modified directly. To create a copy for editing, use the **New** tool on the resource file list, and save it in another location. To include resource files from different locations for use in your models, use the **Path** tool on the resource file list.

To open your model's definition file and review its extensions, select **Language > Edit Current Language**.

For detailed information about the format of these files, see *Customizing and Extending PowerDesigner > Object, Process, and XML Language Definition Files*.

**Note:** Some resource files are delivered with "Not Certified" in their names. Sybase® will perform all possible validation checks, however Sybase does not maintain specific environments to fully certify these resource files. Sybase will support the definition by accepting bug reports and will provide fixes as per standard policy, with the exception that

there will be no final environmental validation of the fix. Users are invited to assist Sybase by testing fixes of the definition provided by Sybase and report any continuing inconsistencies.

## Changing the XML Language

You can change the **XML language** being modeled in your XSM at any time.

Simple types and complex types are only supported by XSDs (schemas). When changing an XSD into a DTD or an XDR, simple types and global complex types (directly linked to the <schema> tag) disappear from the diagram and the Browser tree view. Local complex types (within an element) are expanded in the diagram, beneath their containing element. In this example, HighDefinition is a global complex type, reused as data type for the deluxeTV element:

• In the model with target XSD:
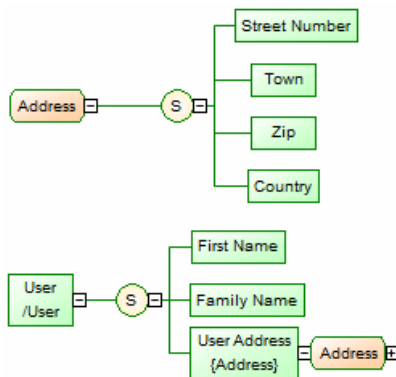


• The model target is changed to DTD or XDR:
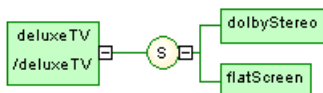


**Note:** You may be required to change the XML language if you open a model and the associated definition file is unavailable.

1. Select **Language > Change Current Language**:
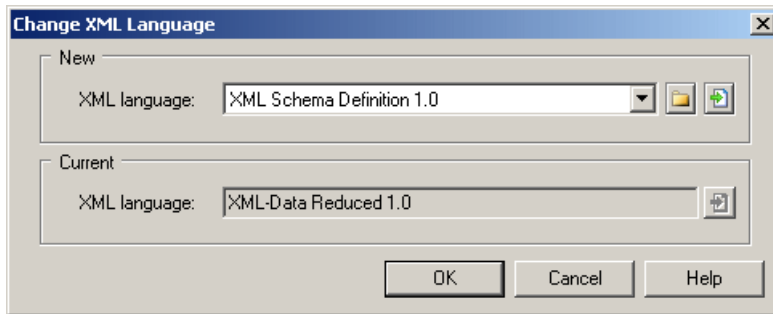
2. Select a **XML language** from the list.

   By default, PowerDesigner creates a link in the model to the specified file. To copy the contents of the resource and save it in your model file, click the **Embed Resource in Model** button to the right of this field. Embedding a file in this way enables you to make changes specific to your model without affecting any other models that reference the shared resource.

3. Click **OK**.

   A message box opens to tell you that the XML language has been changed.

4. Click **OK** to return to the model.

## Extending your Modeling Environment

You can customize and extend PowerDesigner metaclasses, parameters, and file generation with extensions, which can be stored as part of your model or in separate extension files (*.xem) for reuse with other models.

To access extension defined in a `*.xem` file, simply attach the file to your model. You can do this when creating a new model by clicking the **Select Extensions** button at the bottom of the New Model dialog, or at any time by selecting **Model > Extensions** to open the List of Extensions and clicking the **Attach an Extension** tool.

In each case, you arrive at the Select Extensions dialog, which lists the extensions available, sorted on sub-tabs appropriate to the type of model you are working with:
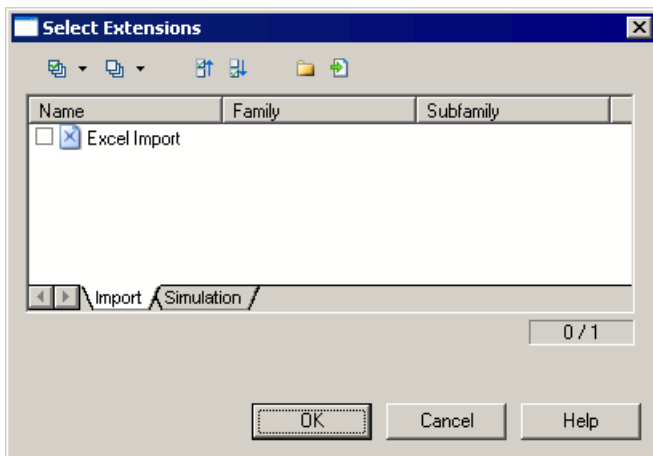
To get started extending objects, see *Core Features Guide > Modeling with PowerDesigner > Objects > Extending Objects*. For detailed information about working with extensions, see *Customizing and Extending PowerDesigner > Extension Files*.

## Linking Objects with Traceability Links

You can create traceability links to show any kind of relationship between two model objects (including between objects in different models) via the **Traceability Links** tab of the object's property sheet. These links are used for documentation purposes only, and are not interpreted or checked by PowerDesigner.

For more information about traceability links, see *Core Features Guide > Linking and Synchronizing Models > Getting Started with Linking and Syncing > Creating Traceability Links*.

CHAPTER 2 **XML Diagrams**

An *XML diagram* provides a graphical view of the elements that comprise an XML schema definition in a tree format.

**Note:** To create an XML diagram in an existing XSM, right-click the model in the Browser and select **New > XML Model Diagram**. To create a new model, select **File > New Model**, choose XML Model as the model type and **XML Model Diagram** as the first diagram, and then click **OK**.

The following example shows the diagram of an XSM which models an XML schema for Resume documents:
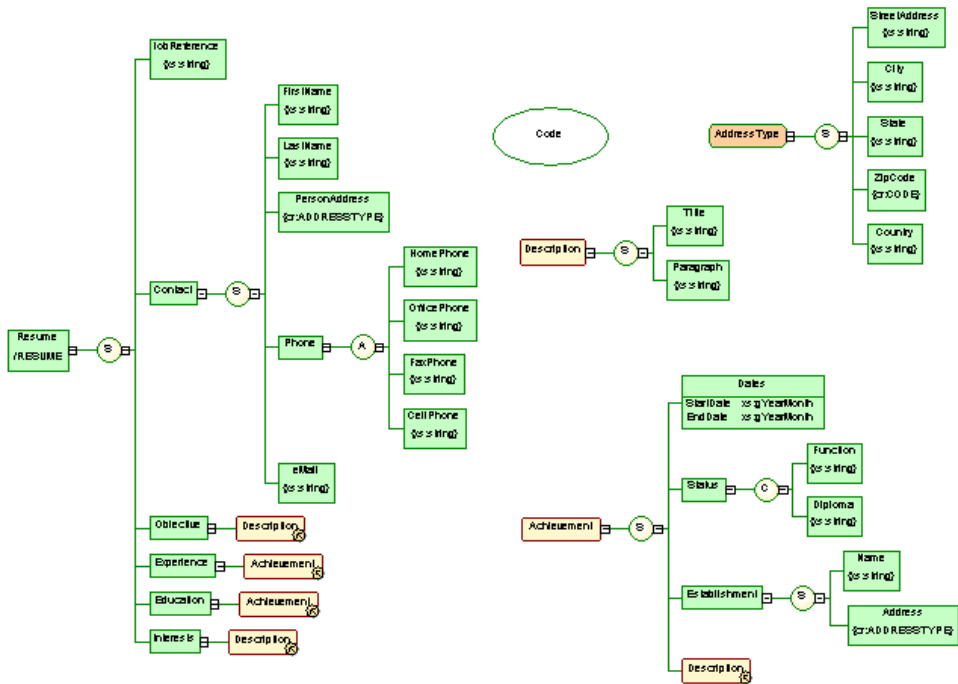


Right-click a symbol in an XML diagram and select one of these features:

- **Expand** - to expand the first level of the hierarchy below the symbol.
- **Expand All** - to expand the whole hierarchy below the symbol.
- **Collapse** - to hide the hierarchy below the symbol.
- **Arrange Symbols**- to reorganize the hierarchy below the symbol.

**Note:** The **Symbol > Group Symbols** command only acts on free symbols in an XML diagram.

If an XML model is too large or too complex, you can create several diagrams to have partial views of the model and focus on certain objects.

For example, the original Resume diagram could be split into five diagrams, corresponding to the five main objects of the model (Main, Contact, Achievement, Description and Address).

The following illustration shows the Achievement sub-diagram:
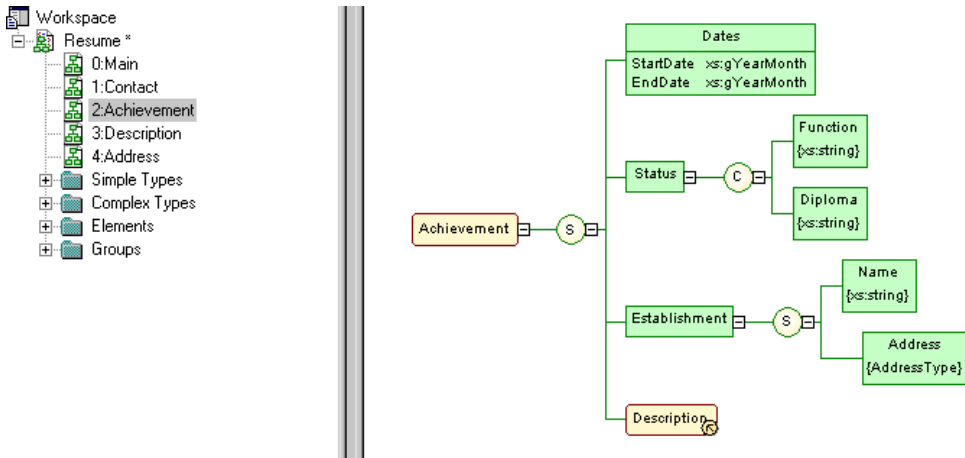


# XML Diagram Objects

An XML model represents the structure of a potential or existing XSD, DTD, or XDR through a tree structure of elements. PowerDesigner supports all the objects necessary to build XML diagrams.

| Object | Tool | Symbol | Description |
|---|---|---|---|
| Element |  |  | The basic object of an XML model. An element can contain other elements or attributes. See *Elements (XSM)* on page 19. |
| Attribute | N/A | N/A | Additional information about an element or a complex type. Defined by a built-in data type or a simple data type. See *Attributes (XSM)* on page 24. |
| Sequence<br>Choice<br>All |  |  | Group particles arranges child elements, so that all must appear at least once in the order of their declaration, only one must be chosen, or any can appear in any order. See *Group Particles (XSM)* on page 29. |

| Object | Tool | Symbol | Description |
|---|---|---|---|
| Simple Type | N/A | N/A | [XSD only] Used in the case of elements or attributes with text-only content. See *Simple Types (XSM)* on page 31. |
| Complex Type |  | Complex | [XSD only] Used to introduce elements or attributes within an element declaration. See *Complex Types (XSM)* on page 33. |
| Group |  | Group | A group of elements arranged by a group particle. Defined once and reused through references. See *Groups (XSM)* on page 37. |
| Attribute Group | N/A | N/A | A group of attributes, defined once and reused in the model through references. See *Attribute Groups (XSM)* on page 41. |
| Any |  | Any | Any type of object. Can only be attached to a sequence or a choice group particle. See *Any Elements (XSM)* on page 43. |
| Constraint | N/A | N/A | [XSD only] Specifies uniqueness of element values. See *Constraints: Keys, Uniques, and KeyRefs (XSM)* on page 45. |
| Derivation | N/A | N/A | Extends or restricts the values of elements and simple and complex types. See *Derivations: Extensions, Restrictions, Lists and Unions (XSM)* on page 50. |
| Instruction | N/A | N/A | An import, include, or redefine instruction. See *Instructions: Import, Include and Redefine (XSM)* on page 61. |
| Annotation | N/A | N/A | Provides documentation or application information. See *Annotations (XSM)* on page 56. |
| Entity | N/A | N/A | [DTD only] Specifies a predefined value or external XML or non-XML file. See *Entities (XSM)* on page 59. |
| Notation | N/A | N/A | Defines and processes non-XML objects within an XML model. See *Notations (XSM)* on page 58. |

## Constructing Schemas in an XSL

You construct a schema in an XSL by building a tree structure of elements and group particles. To link a child object to a parent object, click the child object tool in the Toolbox and then click the symbol of the parent object in the diagram. You can reuse structures of elements in your schema by creating a group or complex type and referencing them.
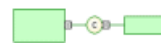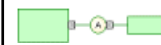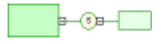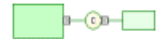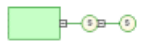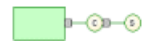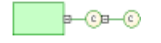
The following tables list the allowed links:

| Tool | Element symbol | Group symbol | Complex type symbol |
|------|---------------|--------------|---------------------|
|  |  |  |  |
|  (Any) |  |  |  |
|  (creates a reference to a group) |  |  |  |
|  | No link | No link | No link |
|  |  |  |  |
|  |  |  |  |
|  (All) |  |  |  |

**Note:** If the tool cannot be used in the current point in the diagram, the cusror displays a forbidden sign. If an object can be created as a sibling or a child to the object under the cursor, it displays an arrow indicating the corresponding direction.

The following table lists the children that you can create under sequence, choice, and all group particles:

| Tool | Sequence symbol | Choice symbol | All symbol |
|------|----------------|---------------|------------|
|  |  |  |  |
|  (Any) |  |  | No link |
|  |  |  | No link |
|  | No link | No link | No link |
|  |  |  | No link |
|  |  |  | No link |
|  (All) | No link | No link | No link |

**Warning!** A group particle (sequence, choice, all) cannot be created from scratch in a diagram. It must be the child element of an element, a group or a complex type.

# Elements (XSM)

Elements are the basic building blocks of an XML model, which organizes them into a tree structure. Elements can contain other elements (via group particles) and attributes, and can reference groups, attribute groups and simple and complex types.

Elements can be either global or local:

- Global elements - have no parent element, and are directly linked to the `<schema>` root element. They can be reused in the model through referencing elements.
- Local elements - have a parent element in a diagram, and are unique within their parent scope. They reference (and, thus, be defined by) a global element by selecting the global element in the **Reference** property on the **General** tab of their property sheet.

**Note:** In a model targeted with the XML-Data Reduced language, local elements are first declared separately, like global elements (with the `<ElementType>` tag and a name attribute), then within their parent element (with the `<element>` tag and a type attribute). For example:

```
<ElementType name="localElement"
<ElementType name="globalElement"
    <element type="localElement"/>
</ElementType>
```

Parent elements are linked to their child elements through group particles (sequence, choice or all), which contain a group of child elements (see the **Group type** property in *Element Properties* on page 20).
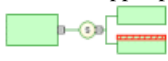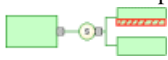
You can derive an XSD element data type to extend or restrict its values (see the **Derivation** property in *Element Properties* on page 20).

## Creating an Element

You can create an element from the Toolbox, Browser, or **Model** menu. Elements can be created as root elements anywhere in the diagram and, via group particles, as children of elements, complex types, and groups.

- Use the **Element** tool in the Toolbox:

| Tool | Action |
|------|--------|
|  | Click in empty space in the diagram to create a root element. |
|  | Click any part of a top-level element symbol to create a sequence group particle (see *Group Particles (XSM)* on page 29) and a child element:  |

| Tool | Action |
|---|---|
|  | Click the upper part of a child element symbol to create a sibling element above it:<br> |
|  | Click the lower part of a child element symbol to create a sibling element below it:<br> |
|  | Click the middle part of a child element symbol to create a sequence group particle and child element to the child element:<br> |

- Select **Model > Elements** to access the List of Elements, and click the **Add a Row** tool.
- Right-click the model or package in the Browser, and select **New > Element**.
- Open the property sheet of a group particle (see *Group Particles (XSM)* on page 29), click the **Items** tab, and use the **Add Element** tool.

For information about creating children under the element, see *Constructing Schemas in an XSL* on page 17.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Element Properties

To view or edit an element's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The General tab of an XSD or DTD element property sheet displays the following properties (for XDR element properties, see the subsequent table):

| Property | Description |
|---|---|
| Name/Code/ Comment | Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. Neither the name nor code should contain colons. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field. |
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |

| Property | Description |
|---|---|
| Reference | Specifies a global element to reuse. Select an element in the current model from the list or click the **Browse** tool to select an element from any model open in the workspace.<br><br>Disables all other properties. To locate the referenced element in the diagram, right-click the referencing element and select **Find Referenced Element**. |
| Group type | Specifies that the object has child elements, and how they are used (see *Group Particles (XSM)* on page 29). You can choose between:<br><br>• all – Each child element can occur 0 or 1 times.<br>• choice – Only one child must be present.<br>• group – Reference to a predefined group (see *Groups (XSM)* on page 37)<br>• sequence – All children must be present in order. |
| Type / IDREF type | Specifies the data type. Select a built-in data type from the list or click the **Browse** tool to select a simple type defined in any model open in the workspace. If you select IDREF or IDREFS, the **IDREF type** property is displayed, allowing you to select the element to reference for documentation purposes.<br><br>For an XSD element, selecting a data type will delete any group particle or attribute previously defined. Do not select a data type if you want to define attributes or child elements within the current element. |
| Embedded type | [XSD only] Specifies a locally defined data type, which applies to the current element only. Automatically set to Complex if you define a derivation for the element data type. |
| Content | [XSD only] Specifies the type of content of the object. You can select:<br><br>• Complex – elements or elements and character data. Click the **Properties** tool to specify an ID (unique within the model) for the complex content, and select the **Mixed** check box if character data can appear between child elements.<br>• Simple – character data or a simple type (but no elements). Click the **Properties** tool to specify an ID for the simple content. |
| Derivation | [XSD only] Specifies a derivation method for the data type to extend or restrict its values. Resets the Type property to <None>. Click the **Properties** tool to further define the derivation (see *Derivations: Extensions, Restrictions, Lists and Unions (XSM)* on page 50). |
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas. |

*Detail Tab*

The Detail tab contains the following properties:

| Property | Description |
| --- | --- |
| Persistent | Specifies how the element will be generated to a PDM. You can choose between:<br><br>• Generate table - the element will be generated as a table (with a reference to its parent, if appropriate).<br>• Migrate columns - the attributes and child elements of the element will be migrated to its parent element. |
| Minimum | Specifies the minimum number of times the object can occur. Enter zero to specify that it is optional. |
| Maximum | Specifies the maximum number of times the object can occur. Select unbounded to specify unlimited instances. |
| Substitution group | Species a global element for which the current element can be substituted. The substitute must have the the same type or derived type. |
| Default | Specifies a default value for the object. Mutually exclusive with Fixed. |
| Fixed | Specifies a fixed value for the object. Mutually exclusive with Default. |
| Block | Specifies that another object with the same type of derivation cannot be used in place of the current one. |
| Final | [global elements] Prevents derivation of the object. |
| Form | Specifies whether or not the object name must be qualified by the target namespace of the schema. |
| ID | Specifies the ID of the object, which must be unique within the model. |
| Abstract | Specifies that the object cannot be used in the instance document. |
| Nillable | Specifies that the element may be null. |

**Note:** In the case of a model targeted with XDR, the Detail tab is only available for local elements.

The following tabs are also available:

• Attributes - lists the attributes and attribute groups associated with the element (see *Attributes (XSM)* on page 24).
• Constraints - lists the constraints associated with the element (see *Constraints: Keys, Uniques, and KeyRefs (XSM)* on page 45).

- Mappings - lists the mappings to objects in other models associated with the element (see *Core Features Guide > Linking and Synchronizing Models > Object Mappings*).

### XDR Element Properties

In a model targeted with the XML-Data Reduced language, elements are defined as follows:

| XDR Element Attribute | Description |
|---|---|
| Model | Specifies if an element can contain new local elements. Possible values are:<br><br>• closed – [default]<br>• open - if an "Any" element is attached to the element. See *Any Elements (XSM)* on page 43 |
| Content | Specifies the content type. Possible values are:<br><br>• mixed - a group particle and a data type are defined<br>• eltOnly - a group particle is defined without a data type<br>• textOnly - a data type is defined without a group particle<br>• empty – neither group particle nor data type are defined<br><br>**General** tab: **Group type**/**Type** |
| Order | Specifies how child elements are organized within a parent element. Possible values are:<br><br>• seq - sequence group particle<br>• one - choice group particle<br>• many - all group particle<br><br>**General** tab: **Group type** |
| dt:type | Specifies a data type.<br><br>**General** tab: **Type** |
| dt:values | Specifies a list of possible element values.<br><br>**Values** tab |
| type | [local elements only] Specifies the name of a global element as reference for the local element<br><br>**General** tab: **Reference** |
| minOccurs | [local elements only] To specify the minimum number of occurrences for a local element. Usually set to *0* or *1*<br><br>**Detail** tab: **Minimum** |

| XDR Element Attribute | Description |
|---|---|
| maxOccurs | [local elements only] To specify the maximum number of occurrences for a local element. Usually set to *1* or *\** (unbounded)<br><br>**Detail** tab: **Maximum** |

# Attributes (XSM)

Attributes can be created under elements or complex types or directly at the root or in an attribute group for reuse.

There are global and local attributes:

- Global attributes are defined with the Model menu. In a schema, they are directly linked to the <schema> tag (root element). They can be reused for any element in the model through references (See "NUMBER" attribute in the generated schema)
- Local attributes only apply to the elements in which they are created. They can be defined by reference to a global attribute (See Reference property)

**Note:** In a model targeted with the XML-Data Reduced language, local attributes are first declared separately, like global attributes (with the <AttributeType> tag and a name attribute), then within their parent element (with the <attribute> tag and a type attribute).

Extract of an XDR file:

```
<AttributeType name="globalAttribute"/>
<ElementType name="parentElement" content="empty">
    <AttributeType name="localAttribute" default="0"
    <attribute default="0" type="localAttribute"/>
</ElementType>
```

In a model targeting XSD, you can derive an attribute data type to extend or restrict its values. For example:



Generated schema:

---

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="COMPANY">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="EMPLOYEES">
                    <xs:complexType>
                        <xs:attribute ref="NUMBER">
                        </xs:attribute>
                    </xs:complexType>
                </xs:element>
                <xs:element name="PRODUCTS">
                    <xs:complexType>
                        <xs:attribute name="REFERENCES" type="xs:string">
                        </xs:attribute>
                        <xs:attribute name="QUANTITIES" type="xs:unsignedLong">
                        </xs:attribute>
                    </xs:complexType>
                </xs:element>
                <xs:element name="CLIENTS">
                    <xs:complexType>
                        <xs:attribute name="FIDELITY" type="xs:string">
                        </xs:attribute>
                        <xs:attribute name="SOLVENCY" type="xs:string">
                        </xs:attribute>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
            <xs:attribute name="FOUNDATION" type="xs:date">
            </xs:attribute>
            <xs:attribute name="STATUS" type="xs:token">
            </xs:attribute>
        </xs:complexType>
    </xs:element>
    <xs:attribute name="NUMBER" type="xs:positiveInteger">
    </xs:attribute>
</xs:schema>
```

## Creating an Attribute

You can create attributes on the **Attributes** tab of an element, complex type, or attribute group property sheet.

The **Attributes** tab contains the following tools:

| Tool | Description |
|------|-------------|
|  | **Add Attribute** - Creates a local attribute. |
|  | **Add Undefined Reference to Attribute Group** - Adds a reference to an attribute group defined in the current model. Select a name from the **Reference** list or enter a name for a new attribute group to be defined later. |
|  | **Add Reference to Attribute** - Adds references to global attributes defined in the current model that you choose from a Selection dialog. |
|  | **Add Reference to Attribute Group** - Adds references to attribute groups defined in the current model that you choose from a Selection dialog. |

---

| Tool | Description |
|------|-------------|
| ☑ | **Any Attribute** - Adds "any" attribute of a specified namespace ( see *Any Attributes* on page 28. |

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Attribute Properties

To view or edit an attribute's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

| Property | Description |
|----------|-------------|
| Name/Code/ Comment | Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. Neither the name nor code should contain colons. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field. |
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |
| Reference | Specifies a global attribute to reuse. Select a global attribute in the current model from the list or click the **Browse** tool to select an attribute from any model open in the workspace. Disables the Name, Code, Type, Default, and Fixed properties. |
| Type / IDREF type | Specifies the data type. Select a built-in data type from the list or click the **Browse** tool to select a simple type defined in any model open in the workspace. If you select IDREF or IDREFS, the **IDREF type** property is displayed, allowing you to select the element to reference for documentation purposes. |
| Embedded Type | [XSD only] Creates a `<simple type>` tag in the schema within the `<attribute>` tag. Resets the Type property to `<None>`. |
| Derivation | [XSD only] Specifies a derivation method for the data type to extend or restrict its values. Resets the Type property to `<None>`. Click the **Properties** tool to further define the derivation (see *Derivations: Extensions, Restrictions, Lists and Unions (XSM)* on page 50). |
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas. |

*Detail Tab*

The Detail tab of an attribute property sheet displays the following properties:

| Property | Description |
|---|---|
| Default | Specifies a default value for the object. Mutually exclusive with Fixed. |
| Fixed | Specifies a fixed value for the object. Mutually exclusive with Default. |
| Use | Specifies how the attribute can be used. Select from:<br><br>• `Optional` - the attribute is optional and may have any value.<br>• `Prohibited` - the attribute cannot be used. Use this value to prohibit the use of an existing attribute in the restriction of another complex type.<br>• `Required` - the attribute must appear at least once and may have any value matching its data type. |
| Form | Specifies whether or not the object name must be qualified by the target namespace of the schema. |
| ID | Specifies the ID of the object, which must be unique within the model. |

*Attribute Property Sheet Values Tab*

The Values tab of an attribute property sheet is only available in a model targeted with DTD or XDR. You can set a list of predefined values for an attribute.

**Note:** In a model targeted with the XML-Data Reduced language, there is also a Values tab in the element property sheet.

*XDR Attribute Properties*

In an XML-Data Reduced language model, attributes tags are defined as follows:

| XDR Attribute Attribute | Description |
|---|---|
| *name* | Specifies the name of the attribute.<br><br>**General** tab: **Name** |
| *default* | Specifies a default value for the attribute.<br><br>**Detail** tab: **Default** |
| *dt:type* | Specifies a type for the attribute.<br><br>**General** tab: **Type** |

| XDR Attribute Attribute | Description |
|---|---|
| *dt:values* | To specify a list of available values for a global attribute<br><br>**Values** tab |
| *type* | Specifies the name of a global attribute as a reference for a local attribute.<br><br>**General** tab: **Reference** |

## Any Attributes

The **Any Attribute** check box in the bottom-left corner of the **Attributes** tab allows you to specify that any attribute of the specified namespaces can be inserted into an element, a complex type or an attribute group declaration. It is only available in a model targeted with XSD.

For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="PRODUCT">
        <xs:complexType>
            <xs:attribute name="EXPIRYDATE" type="xs:date">
            </xs:attribute>
            <xs:anyAttribute namespace="##local" processContents="skip"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Click the **Properties** tool to display the Any Attribute property sheet. The **General** tab contains the following properties:

| Property | Description |
|---|---|
| ID | Specifies the ID of the object, which must be unique within the model. |
| Namespace | Specifies the namespaces containing the attributes that can be used. You can enter a white space delimited list with URI references or choose from:<br><br>• `##any` - attributes from any namespace can be used.<br>• `##other` - attributes from any namespace other than the target namespace of the schema can be used.<br>• `##local` - attributes that are not qualified with a namespace can be used.<br>• `##targetNamespace` - attributes from the target namespace of the schema can be used. |

| Property | Description |
|---|---|
| Process con-tents | Specifies how an XML processor should handle validation of XML documents containing the attributes specified by the Any Attribute. You can choose from:<br><br>• *Lax* - the processor will try to obtain the schema and validate any attribute of the specified namespaces. If the schema cannot be found, no error will occur.<br>• *Skip* - the processor will not try to validate the attributes.<br>• *Strict* - the processor must obtain the schema and validate any attribute of the specified namespaces. |

## Group Particles (XSM)

An element composed of other elements is a parent element with child elements.

Child elements are linked to their parent element through a group particle.
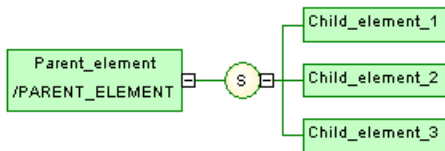


There are three kinds of group particles:

| Tool | Symbol | Description |
|---|---|---|
| | S | *Sequence* - Child elements must appear at least once in the order of their declaration |
| | C | *Choice* - Only one child element can be linked to the parent element |
| | A | *All* - Child elements can appear in any order and each of them once or not at all |

These particles translate to the following tags in each of the supported languages:

| Group Par-ticle | XSD | XDR (order attribute) | DTD (separator) |
|---|---|---|---|
| Sequence | <sequence> | seq | , (comma) |
| Choice | <choice> | one | \| (bar) |
| All | <all> | many | , (comma) |

## Creating a Group Particle

You can create a group particle from the Toolbox or from the property sheet of an element, group, or complex type.

- Select the **Sequence**, **Choice**, or **All** tool in the Toolbox, and click an element, complex type, group, or group particle.

  > **Note:** A sequence particle is automatically created if you click on an element symbol with the **Element** or **Any** tool (see *Creating an Element* on page 19).

- Open the property sheet of an element, group, or complex type, select a **Group type** on the **General** tab, and click **OK**. The element symbol displays a plus sign on its right side that you can click to reveal the group particle.
- Open the property sheet of a group particle, click the **Items** tab, and use the **Add Group Particle** tool.

For information about creating children under the group particle, see *Constructing Schemas in an XSL* on page 17.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Group Particle Properties

To view or edit a group particle's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

| Property | Description |
|----------|-------------|
| Type | Specifies the type of the group particle. |
| Minimum | Specifies the minimum number of times the object can occur. Enter zero to specify that it is optional. |
| Maximum | Specifies the maximum number of times the object can occur. Select unbounded to specify unlimited instances. |
| ID | Specifies the ID of the object, which must be unique within the model. |
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas. |

*Items Tab*
This tab list the child objects associated with the group particle. You can add additional children directly on this tab using the following tools:

| Tool | Description |
|------|-------------|
| ![Add Element icon] | *Add Element* - Adds an element to the list |
| ![Add Any icon] | *Add Any* - [choice or sequence] Adds an any element. |
| ![Add Group Particle icon] | *Add Group Particle* - Adds a group particle. |
| ![Add Reference to Element icon] | *Add Reference to Element* - Adds a reference to a global element. |
| ![Add Reference to Group icon] | *Add Reference to Group* - Adds a reference to a group. |

## Simple Types (XSM)

A simple type is a data type definition defined by derivation of an existing simple type (built-in data type or derived simple type). It can be used by elements or attributes with text-only content; it cannot contain elements or attributes. You can only create simple types in a model targeting XSD.

There are three kinds of derivation for a simple type:

- List - contains a white space-separated list of values of an inherited simple type (see *Deriving by List* on page 55).
- Restriction - has a range of values restricted to a subset of those of an inherited simple type (see *Deriving by Restriction* on page 52).
- Union - contains a union of values of two or more inherited simple types *Deriving by Union* on page 56).

Once defined in a model, a simple type can be reused in the definition of an attribute, an element or a complex type.

Example of a simple type in a schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="BARCODE">
        <xs:restriction base="xs:nonNegativeInteger" id="STR1">
            <xs:length value="13"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="PRODUCTA" type="BARCODE"/>
    <xs:element name="PRODUCTB" type="BARCODE"/>
</xs:schema>
```

## Creating a Simple Type

You can create a simple type from the Browser or **Model** menu. Simple types can only be created at the root of the model.

- Select **Model > Simple Types** to access the List of Simple Types, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Simple Type**.

---

**Warning!** If the simple type symbol does not appear in the diagram, select **Symbol > Show Symbols**, click the **Simple Type** tab, select the simple types that you want to display, and click **OK**.

---

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Simple Type Properties

To view or edit a simple type's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

| Property | Description |
|---|---|
| Name/Code/ Comment | Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. Neither the name nor code should contain colons. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.<br><br>Names and codes must be unique among all simple and complex types. |
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |
| Derivation | [required] [XSD only] Specifies a derivation method for the data type to extend or restrict its values. Resets the Type property to <None>. Click the **Properties** tool to further define the derivation (see *Derivations: Extensions, Restrictions, Lists and Unions (XSM)* on page 50). |
| Final | Prevents derivation of the object. |
| ID | Specifies the ID of the object, which must be unique within the model. |
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas. |

# Complex Types (XSM)

A complex type is an element that contains other elements or attributes, and which is used to define a data type to be reused and derived by extension or restriction. You can only create complex types in a model targeted with XSD.

Complex types are generally created directly under the <schema> tag, to be reused or derived (by extension or restriction) in other parts of the schema. Such global complex types are listed in the Browser, and can have symbols in the diagram. Complex types can also be created within an element, by selecting Complex in the **Embedded Type** field of the element property sheet (see *Element Properties* on page 20). Such local complex types can only be seen as part of the schema in the **Preview** tab of the element property sheet.

In this example, the global Address complex type is selected as the type of the User Address element:
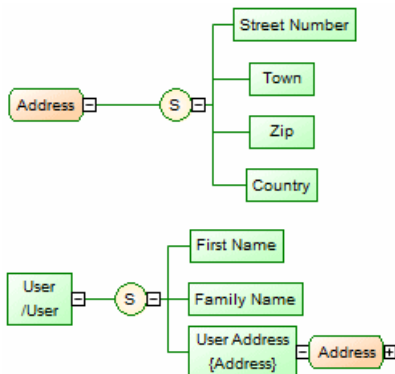


The generated schema is as follows:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="Address">
      <xs:sequence>
        <xs:element name="Street_Number"/>
        <xs:element name="Town"/>
        <xs:element name="Zip"/>
        <xs:element name="Country"/>
      </xs:sequence>
    </xs:complexType>
    <xs:element name="User">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="First_Name"/>
          <xs:element name="Family_Name"/>
          <xs:element name="User_Address" type="Address"/>
```

```
            </xs:sequence>
        </xs:complexType>
      </xs:element>
</xs:schema>
```

## Creating a Complex Type

You can create a complex type from the Toolbox, Browser, or **Model** menu. Complex types can only be created at the model root.

- Use the **Complex Type** tool in the Toolbox.
- Select **Model > Complex Types** to access the List of Complex Types, and click the **Add a Row** tool.
- Right-click the model or package in the Browser, and select **New > Complex Type**.

For information about creating children under the complex type, see *Constructing Schemas in an XSL* on page 17.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Complex Type Properties

To view or edit a complex type's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

| Property | Description |
|---|---|
| Name/Code/ Comment | Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. Neither the name nor code should contain colons. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.<br><br>Names and codes must be unique among all simple and complex types. |
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |

| Property | Description |
|---|---|
| Group type | Specifies that the object has child elements, and how they are used (see *Group Particles (XSM)* on page 29). You can choose between:<br><br>• all – Each child element can occur 0 or 1 times.<br>• choice – Only one child must be present.<br>• group – Reference to a predefined group (see *Groups (XSM)* on page 37)<br>• sequence – All children must be present in order. |
| Content | [XSD only] Specifies the type of content of the object. You can select:<br><br>• Complex – elements or elements and character data. Click the **Properties** tool to specify an ID (unique within the model) for the complex content, and select the **Mixed** check box if character data can appear between child elements.<br>• Simple – character data or a simple type (but no elements). Click the **Properties** tool to specify an ID for the simple content. |
| Derivation | [XSD only] Specifies a derivation method for the data type to extend or restrict its values. Resets the Type property to <None>. Click the **Properties** tool to further define the derivation (see *Derivations: Extensions, Restrictions, Lists and Unions (XSM)* on page 50). |

### Detail Tab

The Detail tab contains the following properties:

| Property | Description |
|---|---|
| Final | Prevents derivation of the object. |
| Block | Specifies that another object with the same type of derivation cannot be used in place of the current one. |
| Mixed | Specifies that character data can appear between child elements. Only appropriate if the complex type has complex content. |
| Abstract | Specifies that the object cannot be used in the instance document. |
| ID | Specifies the ID of the object, which must be unique within the model. |
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas. |

The following tabs are also available:

• Attributes - lists the attributes and attribute groups associated with the complex type (see *Attributes (XSM)* on page 24).

- Mappings - lists the mappings to objects in other models associated with the element (see *Core Features Guide > Linking and Synchronizing Models > Object Mappings*).

## Applying a Complex Type to an Element

You apply a complex type to an element by selecting it in the **Type** list on the **General** tab of the element property sheet.

1. Open the property sheet of the element to which you want to apply the type.
2. If the type is present in the model, even if only in shortcut form, you can select it directly in the **Type** list. If the type is defined in another model open in the workspace, click the **Select Object** tool to the right of this field to select it and create a shortcut in the model.
3. Click **OK** to apply the type and return to the diagram. The complex type symbol appears to the right of the element

   In the following example, the `Name` is a complex type defined in the model and applied to the `Employee/Name` element, while `Address` is a shortcut to a complex type defined in another model and applied to the `Employee/Address` element:
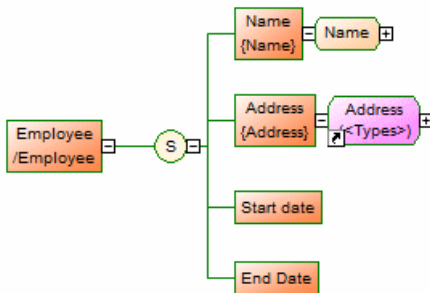
   

4. [optional] Click the plus sign to the right of the complex type to display its child elements. You may need to move the complex type symbol or other symbols to allow all of them to display without overlap:
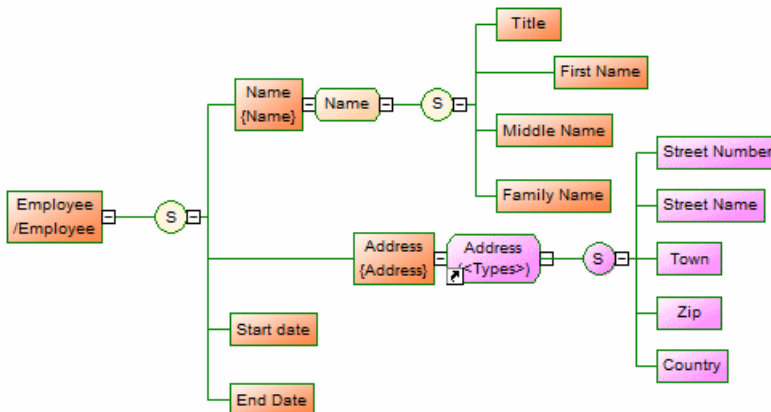
You can edit the properties and child elements of any instance of a complex type defined in the model, and these changes will be propagated to all other instances.

When working with shortcuts to complex types defined in other models, the model containing the complex type must be open for you to be able to display its children. When the model is open and the children displayed, you can move them around temporarily in the diagram, but their positions will not be persisted after you save and close the model.

Complex types and complex type shortcuts and their children are displayed in the Mapping Editor (see *Core Features Guide > Linking and Synchronizing Models > Object Mappings*) under each element to which they are applied, and each instance of the complex type can be mapped independently through generation (see *Generating Other Models from an XSM* on page 69) or manually:
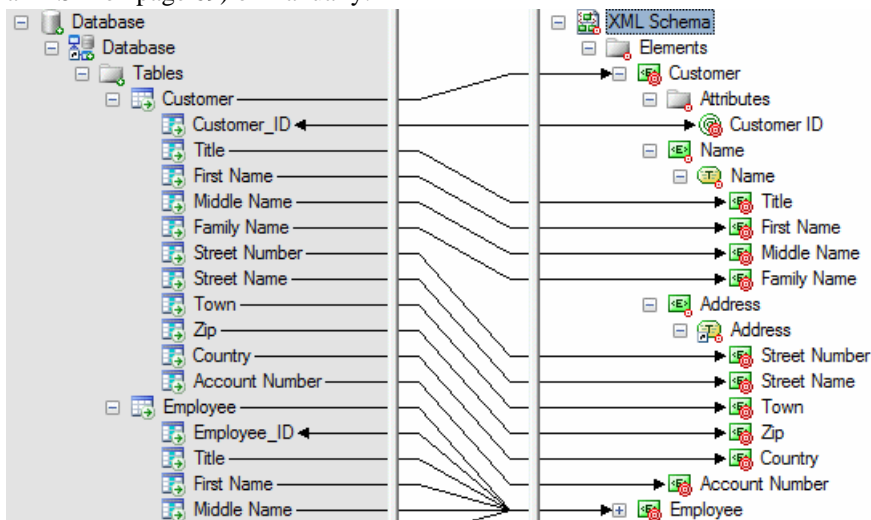


# Groups (XSM)

A group of elements is a set of elements arranged by a group particle (all, choice or sequence), which is then referenced in the model by various elements.

- A *group* - is created independently, without a parent element, and can be reused multiple times by elements, complex types or other global groups, through references. In a schema, it is directly linked to the <schema> tag (root element). See *Creating a group* on page 39.
- A *reference to a group* - is created within an element, complex type or global group, and makes the referenced group available to its parent. See *Creating a reference to a group* on page 39.

For example:
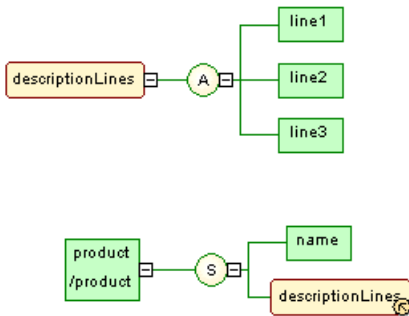
The descriptionLines group is reused in the definition of the product element by clicking the sequence group particle (S) with the Toolbox **Group** tool. The Reference property of the referencing group property sheet is then set to descriptionLines.

In the generated XSD file, the group is first declared with the <group> tag and then reused through a reference (ref) set to descriptionLines:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:group name="descriptionLines">
        <xs:all>
            <xs:element name="line1"/>
            <xs:element name="line2"/>
            <xs:element name="line3"/>
        </xs:all>
    </xs:group>
    <xs:element name="product">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name"/>
                <xs:group ref="descriptionLines">
                </xs:group>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

• In the generated DTD file, the group is expanded directly within its parent element:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT line1 EMPTY>
<!ELEMENT line2 EMPTY>
<!ELEMENT line3 EMPTY>
<!ELEMENT product (name,line1,line2,line3)>
<!ELEMENT name EMPTY>
```

• In the generated XDR file, the group is declared through a <group> tag, within an <ElementType> tag with its order attribute set to seq:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Schema name="XDR_group"
xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="line1" content="empty"/>
  <ElementType name="line2" content="empty"/>
  <ElementType name="line3" content="empty"/>
  <ElementType name="name" content="empty"/>
  <ElementType name="product" model="closed" content="eltOnly" order="seq">
      <element type="name"/>
      <group>
          <element type="line1"/>
          <element type="line2"/>
          <element type="line3"/>
      </group>
  </ElementType>
</Schema>
```

**Note:** In a model targeted with DTD or XDR language, there are no global or referencing groups, although they appear on the diagram. Groups are expanded within their parent element and their child elements are declared individually as global elements. (See generated DTD and XDR files in *Groups (XSM)* on page 37)

## Creating a Group

You can create a group from the Toolbox, Browser, or **Model** menu. Groups are created at the model root to be referenced by other elements.

- Use the **Group** tool in the Toolbox.
- Select **Model > Groups** to access the List of Groups, and click the **Add a Row** tool.
- Right-click the model or package in the Browser, and select **New > Group**.

For information about creating children under the group, see *Constructing Schemas in an XSL* on page 17.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Creating a Reference to a Group

A reference to a group is created as a child of an element, group or complex type, and makes the referenced group available to its parent.

You can create a referencing group in any of the following ways:

- Select the **Group** tool in the Toolbox, and click on an element, group, or complex type symbol.
- On the **Items** tab of the property sheet of a group particle, click the **Add Reference to Group** tool (see *Group Particle Properties* on page 30).

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Group Properties

To view or edit a group's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

| Property | Description |
|---|---|
| Name/Code/ Comment | Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. Neither the name nor code should contain colons. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field.<br><br>Name and Code are read-only for references to groups. |
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |
| Reference | [for references to groups] Specifies the group being referenced. Select a group in the current model from the list or use the **Browse** tool to select a group from any model opened in the workspace.<br><br>Disables all other properties. To locate the referenced group in the diagram, right-click the referencing element and select **Find Referenced Group**. |
| Group type | Specifies that the object has child elements, and how they are used (see *Group Particles (XSM)* on page 29). You can choose between:<br><br>• all – Each child element can occur 0 or 1 times.<br>• choice – Only one child must be present.<br>• sequence – All children must be present in order. |
| Minimum | Specifies the minimum number of times the object can occur. Enter zero to specify that it is optional. |
| Maximum | Specifies the maximum number of times the object can occur. Select unbounded to specify unlimited instances. |
| ID | Specifies the ID of the object, which must be unique within the model. |
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple key-words, separate them with commas. |

# Attribute Groups (XSM)

An attribute group is a set of attributes, which is referenced in the model by various elements. It is created independently, without a parent element, and can be reused multiple times by elements, complex types or other global attribute groups, through references. In a schema, it is directly linked to the `<schema>` tag (root element). Attribute groups are not supported by XDR.

For example:
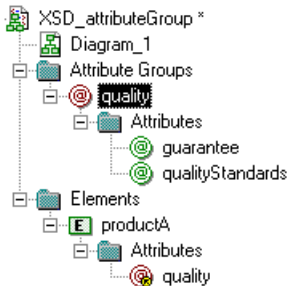


The `quality` attribute group is composed of the `guarantee` and `qualityStandards` attributes. The `productA` element reuses the `quality` attribute group via the **Attributes** tab of its property sheet.

- Generated XSD file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:attributeGroup name="quality">
        <xs:attribute name="guarantee">
        </xs:attribute>
        <xs:attribute name="qualityStandards">
        </xs:attribute>
    </xs:attributeGroup>
    <xs:element name="productA">
        <xs:complexType>
            <xs:attributeGroup ref="quality">
            </xs:attributeGroup>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

In a schema, a group of attributes is declared with the `<attributeGroup>` tag and can contain the `<attribute>`, `<attributeGroup>`, and `<anyAttribute>` tags:

- Generated DTD file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT productA EMPTY>
<!ATTLIST productA
          guarantee           CDATA
          qualityStandards    CDATA>
```

## Creating an Attribute Group

You can create an attribute group from the Browser or **Model** menu. Attribute groups are created at the model root to be referenced by other elements.

You can create an attribute group in any of the following ways:

- Select **Model > Attribute Groups** to access the List of Attribute Groups, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Attribute Group**.

To reference an attribute group, open the property sheet of an element, complex type, or attribute group, click the **Attributes** tab, and then click the **Add Group with Reference to Group** tool (see *Element Properties* on page 20).

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Attribute Group Properties

To view or edit attribute group's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

| Property | Description |
|---|---|
| Name/Code/ Comment | Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. Neither the name nor code should contain colons. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field. Name and Code are read-only for references to attribute groups. |
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |
| Reference | [for references to attribute groups] Specifies the group being referenced. Select a group in the current model from the list or use the **Browse** tool to select a group from any model opened in the workspace. Disables the name and code properties. |
| ID | Specifies the ID of the object, which must be unique within the model. |

| Property | Description |
|----------|-------------|
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple key-words, separate them with commas. |

*Attributes Tab*
This tab lists the attributes and attribute groups associated with the attribute group. For information about the tools available on this tab, see *Creating an Attribute* on page 25.

# Any Elements (XSM)

Any elements allow you to attach any type of object to a choice or a sequence group particle.

The following illustration shows an Any in a diagram:
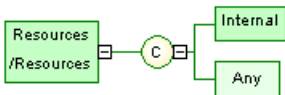


- In an XSD file, Any is declared with the <any> tag:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="RESOURCES">
        <xs:complexType>
            <xs:choice>
                <xs:element name="INTERNAL"/>
                <xs:any namespace="##local" processContents="lax"/>
            </xs:choice>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

- In a DTD file, Any is declared within an <!ELEMENT> tag with the keyword "ANY":

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT Resources ANY>
<!ELEMENT Internal EMPTY>
```

- In an XDR file, Any is declared through of an <ElementType> tag (resources in the example) with its *model* attribute set to "*open*". Although it is displayed in a diagram, Any is not considered as an object in an XDR file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Schema name="XDR_Any"
xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <ElementType name="internal" content="empty"/>
    <ElementType name="resources" model="open" content="eltonly" order="one">
        <element type="internal"/>
    </ElementType>
</Schema>
```

## Creating an Any Element

You can create an Any element from the Toolbox or from a group particle property sheet.

- Use the **Any** tool in the Toolbox.
- Open the **Items** tab in the property sheet of a group particle, and click the **Add Any** tool.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Any Element Properties

To view or edit an any element's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

| Property | Description |
|---|---|
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |
| Minimum | Specifies the minimum number of times the object can occur. Enter zero to specify that it is optional. |
| Maximum | Specifies the maximum number of times the object can occur. Select unbounded to specify unlimited instances. |
| ID | Specifies the ID of the object, which must be unique within the model. |
| Namespace | Specifies the namespaces containing the elements that can be used. You can enter a white space delimited list with URI references or choose from:<br><br>• ##any - elements from any namespace can be used.<br>• ##other - elements from any namespace other than the target namespace of the schema can be used.<br>• ##local - elements that are not qualified with a namespace can be used.<br>• ##targetNamespace - elements from the target namespace of the schema can be used. |
| Process contents | Specifies how an XML processor should handle validation of XML documents containing the elements specified by the Any element. You can choose from:<br><br>• *Lax* - the processor will try to obtain the schema and validate any element of the specified namespaces. If the schema cannot be found, no error will occur.<br>• *Skip* - the processor will not try to validate the elements.<br>• *Strict* - the processor must obtain the schema and validate any element of the specified namespaces. |

| Property | Description |
|---|---|
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple key-words, separate them with commas. |

# Constraints: Keys, Uniques, and KeyRefs (XSM)

Constraints indicate that element values must be unique within their specified scope. In a schema, a constraint is declared with the `<unique>`, `<key>`, or `<keyRef>` tag. Constraints are only available in a model targeted with XSD.

There are three kinds of identity constraints, each with a selector and field:

• A *Unique* constraint - specifies that an element or an attribute value (or set of values) must be unique or null within a specified scope. For example:
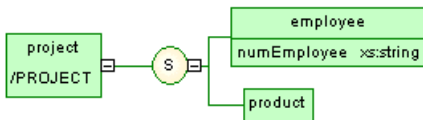


Generated schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="PROJECT">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="EMPLOYEE">
                    <xs:complexType>
                        <xs:attribute name="NUMEMPLOYEE" type="xs:string">
                        </xs:attribute>
                    </xs:complexType>
                </xs:element>
                <xs:element name="PRODUCT"/>
            </xs:sequence>
        </xs:complexType>
        <xs:unique name="UNIQUENUM">
            <xs:selector xpath="employee"/>
            <xs:field xpath="@numEmployee"/>
        </xs:unique>
    </xs:element>
</xs:schema>
```

The `UNIQUENUM` unique constraint, defined on the `project` element, specifies that the `numEmployee` attribute must be unique or null within the `employee` element

• A *Key* constraint - specifies that an element or an attribute value (or set of values) must be a key within a specified scope; the data must be unique, not null, and always present within a specified scope. For example:
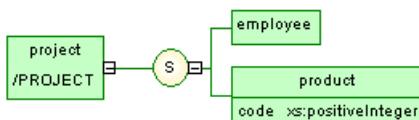
Generated schema:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="PROJECT">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="EMPLOYEE"/>
                <xs:element name="PRODUCT">
                    <xs:complexType>
                        <xs:attribute name="CODE" type="xs:positiveInteger">
                        </xs:attribute>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
        <xs:key name="KEYCODE">
            <xs:selector xpath="product"/>
            <xs:field xpath="@code"/>
        </xs:key>
    </xs:element>
</xs:schema>
```

The KEYCODE key constraint, defined on the project element, specifies that the code attribute must be unique, not null and always present within the product element.

- A *KeyRef* constraint - specifies that an element or attribute value (or set of values) corresponds to the value of a specified key or unique constraint. A keyRef is a reference to a key or a unique constraint. For example:
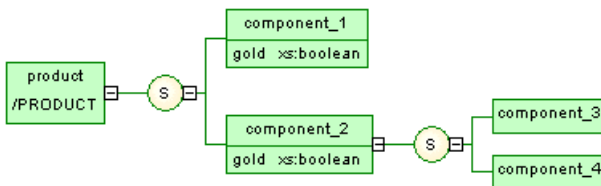


Generated schema:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:k="keyRef.namespace">
    <xs:element name="PRODUCT">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="COMPONENT_1">
                    <xs:complexType>
                        <xs:attribute name="GOLD" type="xs:boolean">
                        </xs:attribute>
                    </xs:complexType>
                </xs:element>
                <xs:element name="COMPONENT_2">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="COMPONENT_3"/>
                            <xs:element name="COMPONENT_4"/>
                        </xs:sequence>
                        <xs:attribute name="GOLD" type="xs:boolean">
                        </xs:attribute>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
        <xs:unique name="UNIGOLD">
            <xs:selector xpath="k:COMPONENT_1"/>
            <xs:field xpath="@GOLD"/>
        </xs:unique>
        <xs:keyref name="KEYREF_UNIGOLD" refer="k:UNIGOLD">
            <xs:selector xpath="k:COMPONENT_2"/>
            <xs:field xpath="@GOLD"/>
        </xs:keyref>
    </xs:element>
</xs:schema>
```

The KEYREF_UNIGOLD keyRef, defined on the product element, by reference to the UNIGOLD unique constraint, specifies that the gold attribute must be unique or null within the component_2 element, and unique or null within the component_1 element.

## Creating a Constraint

You create a constraint on the **Constraints** tab of an element property sheet.

The **Constraints** tab contains the following tools:

| Tool | Description |
|------|-------------|
| 🔧 | *Add Key Constraint* - The element value must be a key within the specified scope. The scope of a key is the containing element in an instance document. A key must be unique, not null, and always present. |
| 🔧 | *Add Unique Constraint* - The element value must be unique or null within the specified scope. |
| 🔧 | *Add KeyRef Constraint* - The element value corresponds to those of the specified key or unique constraint |

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Constraint Properties

To view or edit a constraint's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

| Property | Description |
|---|---|
| Name/Code/ Comment | Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. Neither the name nor code should contain colons. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field. |
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |
| ID | Specifies the ID of the object, which must be unique within the model. |
| Reference | [keyrefs] Specifies the key or unique constraint being referenced. Select a constraint defined in the current model (or another model with a specified namespace). |
| Selector (XPath) | Enter an XPath expression that selects a set of elements across which the values specified in the Fields tab must be unique. There can only be one selector. |
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple key-words, separate them with commas. |

### Fields Tab

The Fields tab lists XPath expressions used to define the constraint. If more than one field is listed, the combination of fields must be unique.

The XPath expressions permitted to define constraint selectors and fields are limited to a subset of the full XPath language defined in the W3C Recommendation XML Path Language 1.0:

| Syntax | Description |
|---|---|
| / | Root node of the XML document. It is the root element with its ramifications |

| Syntax | Description |
|---|---|
| . | Selects the context node. It is the current element (on which an identity constraint is defined) with its ramifications |
| .. | Selects the context node parent |
| * | Selects all the child elements of the context node |
| employee | Selects all the employee child elements of the context node |
| s:employee | Selects all the employee child elements of the context node, defined in the namespace with the "s" prefix |
| @numEmployee | Selects the numEmployee attribute of the context node |
| @* | Selects all the attributes of the context node |
| ../@numEmployee | Selects the numEmployee attribute of the context node parent |
| employee[1] | Selects the first employee child element of the context node |
| employee[last()] | Selects the last employee child element of the context node |
| */employee | Selects all the employee grandchildren of the context node |
| //employee | Selects all the employee descendants of the root node |
| .//employee | Selects the employee descendants of the context node |
| company//employee | Selects the employee descendants of the company child elements of the context node |
| //company/employee | Selects all the employee elements with company as parent element in the context node |
| /book/chapter[2]/section[3] | Selects the third section in the second chapter of the book |
| employee[@dept="doc"] | Selects all the employee child elements of the context node with a dept attribute set to doc |
| employee[@dept="doc"][3] | Selects the third employee child element of the context node with a dept attribute set to doc |
| employee[3][@dept="doc"] | Selects the third employee child element of the context node only if it has a dept attribute set to doc |
| chapter[title] | Selects the chapter child elements of the context node with at least one title child element |

| Syntax | Description |
|---|---|
| chapter[title="About this book"] | Selects the chapter child elements of the context node with at least one title child element with a text content set to About this book |
| employee[@numEmployee and @dept] | Selects all the employee child elements of the context node with the numEmployee and dept attributes |
| text() | Selects all the child nodes of the text context node |

*Field and Selector Properties*
The General tab of a selector or field property sheet contains the following properties:

| Property | Description |
|---|---|
| XPath | For a selector: An XPath expression relative to the parent element being declared. It identifies the child elements to which the identity applies<br><br>For a field: An XPath expression relative to each element selected by the selector of the constraint. It identifies a single element (with a simple type) whose content or value is used for the constraint |
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |
| ID | Specifies the ID of the object, which must be unique within the model. |

# Derivations: Extensions, Restrictions, Lists and Unions (XSM)

You can use derivations to extend or restrict the values of elements and of simple and complex types.

An XML model allows you to derive:

- Elements by extension, restriction, list or union
- Simple types by restriction, list or union
- Complex types by extension or restriction

**Note:** When you define a derivation in an element property sheet, a simple or a complex type is automatically created within the element declaration (See Preview tab). The Embedded type property is automatically set to Simple or Complex, and the Content property to Simple or Complex in the case of an embedded complex type.

## Deriving by Extension

You can derive an element or complex type by extension to extend the values of its base type.

For example:
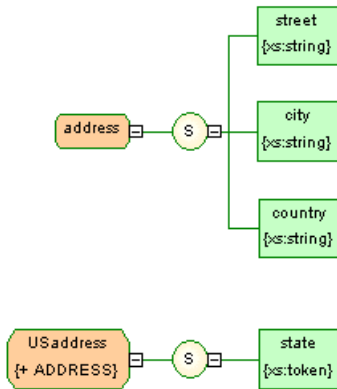


USaddress is a derivation by extension of the address complex type.

The generated schema is the following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="ADDRESS">
        <xs:sequence>
            <xs:element name="STREET" type="xs:string"/>
            <xs:element name="CITY" type="xs:string"/>
            <xs:element name="COUNTRY" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="USADDRESS">
        <xs:complexContent>
            <xs:extension base="ADDRESS">
              <xs:sequence>
                  <xs:element name="STATE" type="xs:token"/>
              </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

1. Open the property sheet of an element or complex type and select Extension in the Derivation list.

   The Content field (and, in the case of an element, the Embedded type field) is set to Complex.

2. Click the Properties tool to the right of the Derivation box to open the property sheet of the extension and complete the following properties:

| Property | Description |
| --- | --- |
| ID | ID of the extension. Its value must be of type ID and unique within the model containing the extension |
| Base Type | Data type on which the extension is based |

**3.** Specify an ID, select a base type, and then click OK to return to the element or complex type.

## Deriving by Restriction

You can derive an element, simple type, or complex type by restriction to restrict the values of their base type.

**1.** Open the property sheet of an element, simple type, or complex type, and select Restriction in the Derivation list.



For elements and complex types, the Content field (and, in the case of an element, the Embedded type field) is set to Complex.

**2.** Click the **Properties** tool to the right of the **Derivation** field to open the restriction property sheet, and complete the following fields on the **General** tab:

---

| Property | Description |
|---|---|
| ID | ID of the simple type restriction. Its value must be of type ID and unique within the model containing the simple type restriction |
| Base type | Data type on which the restriction is based. Select a data type in the Base type list or with the Browse tool |
| Embedded type [simple types only] | If selected, the base type disappears and a simple type is created in the schema within the current simple type. Click Apply, and then the Properties tool beside the Embedded type box, to define a derivation and a base type for the embedded simple type. |

**3.** [optional - simple type restrictions only] Click the **Detail** tab and enter appropriate facets (constraints on the set of values of a simple type) for the restriction:

| Icon | Facet |
|---|---|
|  | *Length* - Exact number of characters or list items allowed. It must be equal to or greater than zero |
|  | *Whitespace* - Way of handling white spaces. You can choose from the following:<br>• *Preserve* - white spaces are unchanged.<br>• *Replace* - Tabs, line feeds and carriage returns are replaced with spaces.<br>• *Collapse* - Contiguous sequences of spaces are collapsed to a single space. Leading and trailing spaces are removed. |
|  | *Minimum length* - Minimum number of characters or list items allowed. It must be equal to or greater than zero |
|  | *Maximum length* - Maximum number of characters or list items allowed. It must be equal to or greater than zero |
|  | *Minimum exclusive* - Lower bound for numeric values. All values are greater than this value |
|  | *Maximum exclusive* - Upper bound for numeric values. All values are lower than this value |
|  | *Minimum inclusive* - Minimum value allowed for data type |
|  | *Maximum inclusive* - Maximum value allowed for data type |
|  | *Total digits* - Exact number of decimal digits allowed. It must be greater than zero |
|  | *Fraction digits* - Maximum number of decimal digits in the fractional part |

You can optionally click the **Properties** tool to the right of each field to open the property sheet of the facet and enter the following properties:

| Property | Description |
|----------|-------------|
| ID | ID of the facet. Its value must be of type ID and unique within the model containing the facet |
| Value | Value(s) of the facet |
| Fixed | To prevent a modification of the facet value(s), select the Fixed property |

**4.** [optional - simple type restrictions only] Click the **Enumerations** tab and enter a set of acceptable values. Select the **F**[ixed] check box to prevent the modification of a value.

For example: the meetings simple type, based on the xs:gMonthDay data type, is restricted to the following dates: 01/20, 03/20, 05/20 and 07/20.
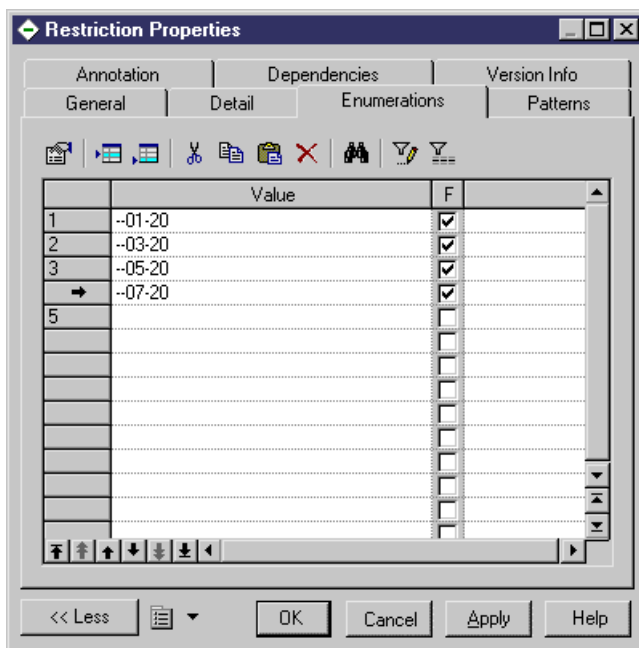


Generated schema:

```
<xs:simpleType name="MEETINGS">
  <xs:restriction base="xs:gMonthDay">
    <xs:enumeration value="--01-20"/>
    <xs:enumeration value="--03-20"/>
    <xs:enumeration value="--05-20"/>
    <xs:enumeration value="--07-20"/>
    <xs:enumeration/>
  </xs:restriction>
</xs:simpleType>
```

**5.** [optional - simple type restrictions only] Click the **Patterns** tab and enter one or more sequences of acceptable values. Select the **F**[ixed] check box to prevent the modification of a value.

For example: the zipCode simple type, based on the xs:string data type, is restricted to the following pattern: two uppercase letters, from A to Z, followed by a five-digit number, each digit ranging from 0 to 9.
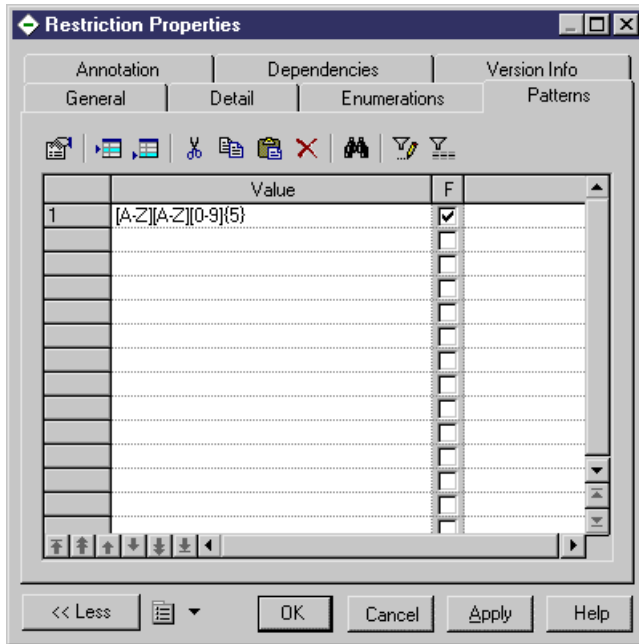


Generated schema:

```
<xs:simpleType name="ZIPCODE">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Z][A-Z][0-9]{5}"/>
  </xs:restriction>
</xs:simpleType>
```

6. Click **OK** to close the restriction property sheet and return to the element, simple type, or complex type.

## Deriving by List

You can derive an element or simple type by list to define it as a list of values of a specified data type.

1. Open the property sheet of an element or simple type, and select List in the Derivation list.

   For elements, the Embedded type field is set to Simple.

2. Click the Properties tool to the right of the Derivation box to open the list property sheet and complete the following properties:

| Property | Description |
|----------|-------------|
| ID | ID of the simple type list. Its value must be of type ID and unique within the model. |
| Type | Data type for the list of values |
| Embedded Type | If selected, the type disappears and a simple type is created in the schema within the current simple type or element. Click Apply, and then the Properties tool beside the Embedded type box, to define a derivation and a type for the embedded simple type. |

**3.** Click OK to close the list property sheet and return to the element or simple type.

## Deriving by Union

You can derive an element or simple type by union to define it as a collection of built-in and simple data types.

**1.** Open the property sheet of an element or simple type, and select Union in the Derivation list.

For elements, the Embedded type field is set to Simple.

**2.** Click the Properties tool to the right of the Derivation box to open the union property sheet and complete the following properties:

| Property | Description |
|----------|-------------|
| ID | ID of the simple type union. Its value must be of type ID and unique within the model containing the simple type union. |
| Member Types | White space separated list of built-in data types. Values must be qualified names. |

**3.** [optional] Click the Union Types tab and add appropriate simple types to the union.

**4.** Click OK to close the union property sheet and return to the element or simple type.

# Annotations (XSM)

Annotations allow you to add information about an XSD model. Annotations can be added at the schema level or on any element or other object within an XML model targeted with XSD.

*Creating an Annotation*
To create an annotation:

• On the schema element - Open the **Items** or **External Schemas** tab in the property sheet of the model, and click the **Add Annotation** tool or right-click the model or package in the

Browser, and select **New > Annotation**. You can add documentation and application information as necessary on the **Items** tab.

• On any other object - Open the **Annotations** tab and click one of the following tools:

| Tool | Description |
|---|---|
| [icon] | **Add Documentation** – to contain an URI reference or any well-formed XML content that gives extra information about XML objects or documents. |
| [icon] | **Add Application Information** - to contain an URI reference or any well-formed XML content that is used by applications for processing instructions. |

*Annotation Properties*

The **General** tab contains the following properties:

| Property | Description |
|---|---|
| ID | [annotation] Specifies the ID of the object, which must be unique within the model. |
| Source | [documentation and application information] Specifies the source of the content as a URI. |
| Language | [documentation] Specifies the language used in the documentation. For example: en, en-GB, en-US, de, fr. |

The **Content** tab allows you to enter any well-formed XML content and the **Preview** tab (of the annotation or parent object) allows you to review the content within the appropriate tags. The following schema `annotation` contains a `documentation` element with well-formed XML content (an extract of a DTD file), and an `appinfo` element with a source URI:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:annotation id="ANNOT1">
        <xs:documentation source="attributes.dtd" xml:lang="en-US">
        <?xml version="1.0" encoding="UTF-8" ?>
        <!ELEMENT COMPANY (EMPLOYEES,PRODUCTS,CLIENTS)>
        <!--  -->
        <!ATTLIST COMPANY
            FOUNDATION                       CDATA
            STATUS                           CDATA>
        <!ELEMENT EMPLOYEES EMPTY>
        <!--  -->
        <!ATTLIST EMPLOYEES
            NUMBER                           CDATA>
        <!ELEMENT PRODUCTS EMPTY>
        <!--  -->
        <!ATTLIST PRODUCTS
            REFERENCES                       CDATA
            QUANTITIES                       CDATA>
        <!ELEMENT CLIENTS EMPTY>
        <!--  -->
        <!ATTLIST CLIENTS
            FIDELITY                         CDATA
            SOLVENCY                         CDATA>
        </xs:documentation>
        <xs:appinfo source="http://www.parsersandco.com"></xs:appinfo>
    </xs:annotation>
</xs:schema>
```

# Notations (XSM)

Notations allow you to define and process non-XML objects within an XML model.

The following example shows the generated schema for a notation:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <!--
    Integrating GIF files in your XML model
    -->
    <xs:notation name="PICTURES" public="pictures/gif"
 system="user/local/pictureViewer"/>
</xs:schema>
```

Notations are not available on models targeted with XDR.

## Creating a Notation

You can create a notation from the Browser or **Model** menu.

• Select **Model > Notations** to access the List of Notations, and click the Add a Row tool.
• Right-click the model or package in the Browser, and select **New > Notation**.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Notation Properties

To view or edit a notation's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

| Property | Description |
|---|---|
| Name/Code/ Comment | Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. Neither the name nor code should contain colons. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field. |
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |
| Public | URI reference identifying the non-XML object. For example: pictures/gif. |
| System | URI reference identifying the application that will process the non-XML object. For example: user/local/pictureViewer. |
| ID | ID of the notation. Its value must be of type ID and unique within the model containing the notation. |
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas. |

# Entities (XSM)

Entities enable you to include predefined values, external XML or non-XML files in an XML model targeted with a DTD.

When an XML processor reads an entity reference in an XML document, it will replace this entity reference by its value defined in the DTD file of the XML document.

An entity reference is the entity name preceded by an ampersand and followed by a semicolon.

For example: `&furtherinfo;` will be replaced by `For further information, see`.

The W3C has predefined five entities for XML tags:

| Entity name | Reference | Value |
|---|---|---|
| Less than | &lt; | < |

| Entity name | Reference | Value |
|---|---|---|
| Greater than | &gt; | > |
| Ampersand | &amp; | & |
| Apostrophe | &apos; | ' |
| Quotation | &quot; | " |

In an XML model, you just need to type the name and the value of an entity.

## Creating an Entity

You can create an entity from the Browser or **Model** menu.

- Select **Model > Entities** to access the List of Entities, and click the Add a Row tool.
- Right-click the model or package in the Browser, and select **New > Entity**.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Entity Properties

To view or edit an entity's properties, double-click its diagram symbol or Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

| Property | Description |
|---|---|
| Name/Code/ Comment | Identify the object. The name should clearly convey the object's purpose to non-technical users, while the code, which is used for generating code or scripts, may be abbreviated, and should not normally include spaces. Neither the name nor code should contain colons. You can optionally add a comment to provide more detailed information about the object. By default the code is generated from the name by applying the naming conventions specified in the model options. To decouple name-code synchronization, click to release the = button to the right of the **Code** field. |
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |
| Value | Value of the entity. A string of characters in the case of a predefined value. A URI in the case of an XML or a non-XML file. For example: http://something.com/pictures/logo.gif. |
| Public | URI reference identifying the non-XML object. For example: pictures/gif. |
| System | URI reference identifying the application that will process the non-XML object. For example: user/local/pictureViewer. |

| Property | Description |
|---|---|
| Notation | Used to define and process non-XML objects within an XML model. |
| Parameter | If selected, the entity is parsed within the DTD, and not within the XML document as for a general entity. A parameter entity allows you to predefine a value within a DTD. This predefined value can then be easily changed within the DTD. |
| Keywords | Provide a way of loosely grouping objects through tagging. To enter multiple keywords, separate them with commas. |

# Instructions: Import, Include and Redefine (XSM)

Import, Include and Redefine allow you to enrich your XML model with external namespaces, schema files or schema components.

These instructions are only available in a model targeted with XSD.

*Imports*
An import identifies a namespace whose schema components are referenced by the current schema, allowing you to use components from any schema with different target namespace than the current schema.

In a schema, an import is declared with the <import> tag. For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:import id="IMP1" namespace="xml.ordering"
 schemaLocation="ORDER.xsd"/>
</xs:schema>
```

*Includes*
An include allows you to include a specified schema file in the target namespace of the current schema, allowing you to use components from any schema with the same target namespace as the current schema or with no specified target namespace.

In a schema, an include is declared with the <include> tag. For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:include id="INC1" schemaLocation="PROFORMA.xsd"/>
</xs:schema>
```

*Redefines*
A redefine allows you to redefine simple and complex types, groups and attribute groups from an external schema file in the current schema, allowing you to use components from any schema with the same target namespace as the current schema or with no specified target namespace.

In a schema, a redefine is declared with the <redefine> tag. For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:redefine id="REDEF1" schemaLocation="CUSTOMERS.xsd">
        <xs:group name="PRIVILEGED_CUSTOMERS">
            <xs:all>
                <xs:element name="CUSTOMER_1"/>
                <xs:element name="CUSTOMER_2"/>
                <xs:element name="CUSTOMER_3"/>
            </xs:all>
        </xs:group>
        <xs:complexType name="PRIVILEGE">
        </xs:complexType>
    </xs:redefine>
</xs:schema>
```

## Creating an Import, Include, or Redefine Instruction

You can create an import, include, or redefine instruction from the model property sheet or from the Browser or **Model** menu.

- Select **Model > Import**, **Include**, or **Redefine** to access the relevant list, and click the Add a Row tool.
- Open the External Schemas tab in the property sheet of the model, and click the Add Import, Add Include, or Add Redefine tool.
- Right-click the model or package in the Browser, and select **New > Import**, Include, or Redefine.

For general information about creating objects, see *Core Features Guide > Modeling with PowerDesigner > Objects*.

## Import, Include, and Redefine Properties

To view or edit an instruction's properties, double-click its Browser or list entry. The property sheet tabs and fields listed here are those available by default, before any customization of the interface by you or an administrator.

The **General** tab contains the following properties:

| Property | Description |
|---|---|
| Schema location | URI reference for the location of a schema file with an external namespace. You can use the Browse tool beside the Properties tool to select a schema file among those opened in the current workspace. For example: ORDER.xsd. |
| ID | ID of the instruction. Its value must be of type ID and unique within the schema containing the instruction. |
| Namespace | [import only] URI reference for the namespace to import. For example: xml.ordering. |

| Property | Description |
|---|---|
| Comment | Descriptive label of the instruction. |
| Stereotype | Extends the semantics of the object. You can enter a stereotype directly in this field, or add stereotypes to the list by specifying them in an extension file. |

*Items Tab*

The **Items** tab, which is available for redefines only, lists the items to be redefined. The following tools are available:

| Tool | Description |
|---|---|
|  | *Add Group* - Adds a group of elements to be redefined. |
|  | *Add Attribute Group* - Adds a group of attributes to be redefined. |
|  | *Add Simple Type* - Adds a simple type to be redefined. |
|  | *Add Complex Type* - Adds a complex type to be redefined. |

# Business Rules (XSM)

A business rule is a rule that your business follows. It is a written statement specifying what an information system must do or how it must be structured. It could be a government-imposed law, a customer requirement, or an internal guideline.

You can attach business rules to your model objects to guide and document the creation of your model. For example, the rule "an employee belongs to only one division" can help you graphically build the link between an employee and a division.

For more information, see *Core Features Guide > Modeling with PowerDesigner > Objects > Business Rules*.

**Generating and Reverse Engineering XML Schemas and Other Models**

PowerDesigner supports the generation and reverse-engineering of XML Schema Definition files (.XSD), Document Type Definition files (.DTD) and XML-Data Reduced files (.XDR). You can also generate a physical data model (PDM) from an XSM or generate an XSM from a PDM

## Generating XML Schema Files

PowerDesigner provides a standard interface for generating all the supported XML schemas.

| Target Schema | Generated file |
|---|---|
| XML Schema Definition 1.0 | XSD |
| Document Type Definition 1.0 | DTD |
| | **Note:** Parameter entities are references to predefined values within a DTD file (See Parameter property in entity property sheet). During DTD generation, some object properties containing inadvertently parameter values will be generated with parameter references. If you are not satisfied with this default use of parameter entities, you should clear the Parameter property before generation. |
| XML-Data Reduced 1.0 | XDR |

You can preview the file to be generated by selecting the Preview tab of your XML model property sheet (see *Previewing XML Code* on page 8).

**Note:** The PowerDesigner generation system is extremely customizable through the use of extensions (see *Extending your Modeling Environment* on page 13). For detailed information about customizing generation, including adding generation targets, options, and tasks, see *Customizing and Extending PowerDesigner > Extension Files*.

**1.** Select **Language > Generate***schema* **File** to open the Generation dialog:
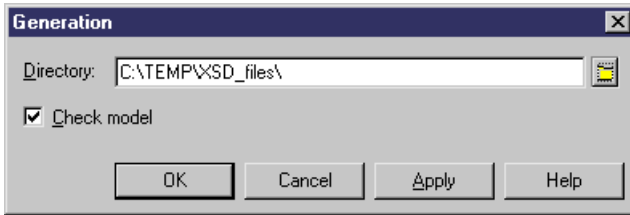
**2.** Enter a directory in which to generate the files and specify whether you want to perform a model check. For more information about checking your model, see *Chapter 4, Checking an XSM* on page 71.

**Note:** When generating an XDR file, the Generation dialog contains an Options tab, where you can specify whether or not to generate comments (within a <description> tag). This option is enabled by default.

**3.** Click OK to begin generation.

A Progress box is displayed. The Result list displays the files that you can edit. The result is also displayed in the Generation tab of the Output window, located in the bottom part of the main window.

**4.** Click Edit to edit the XSD, DTD or XDR file in your associated editor:
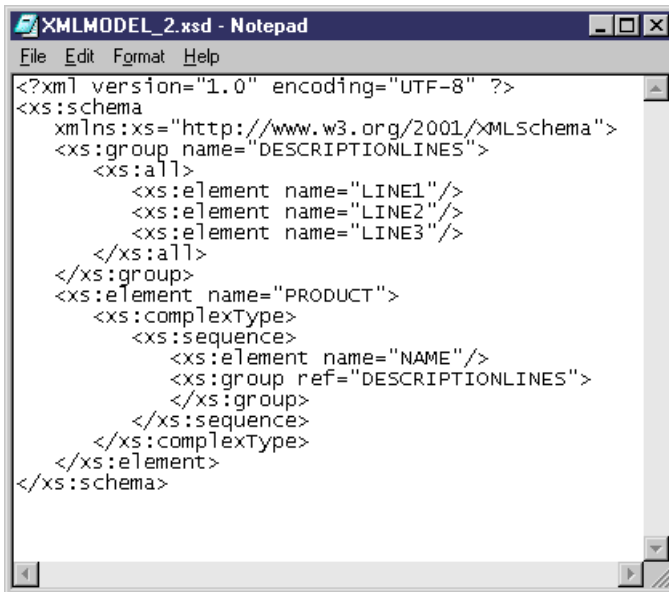
# Reverse Engineering an XML Schema into an XSM

Reverse engineering is the process of extracting an XML structure from an XML schema file into an XSM. You can reverse engineer XML schema files to create a new XSM or to add objects to an existing XSM.

**Note:** PowerDesigner uses parser software developed by the Apache Software Foundation (*http://www.apache.org*) for XML reverse engineering.

1. To reverse engineer a schema and create a new XSM, select **File > Reverse Engineer > XML Definition** to open the New XML Model dialog box. Specify a model name, choose an XML language from the list, and then click **OK**.

   **Note:** If your reverse-engineering will create multiple connected XSM models due to import and include statements, you may first want to create an empty project (**File > New Project**) and reverse-engineer your files into the project, which will act as a container for the files and allow you to check them into and out of the repository as a single unit.

   or

   To reverse engineer a schema into an existing XSM, open the model and select **Language > Reverse Engineer *schema* File**.

2. When the Reverse Engineering dialog opens, select the file or enter the URL you want to reverse-engineer (which can be of any type accessible via a browser except for ftp), and select any appropriate options:

   | Option | Description |
   | --- | --- |
   | Show symbols | Creates symbols for the reversed objects in the diagram. You can specify to expand all the nodes, and to display elements, groups, and complex and simple types. |
   | Convert unique references to elements | Transforms global objects that are referenced only once in the model into child objects. You can perform this conversion at any time by selecting **Tools > Convert Unique References** in the XML model. |

3. [XSD only] If the schema contains Import or Include elements, the schema files referenced are listed in a tree format in the dialog. Files not found display a red overlay. For each file in the tree (including those which are found), you can click on it to :

   • In the **File or URL** field, change the value to an appropriate local path or valid URL path.
   • In the **File or URL** field, browse to and select an XSM open in the workspace to stand in for the referenced file.
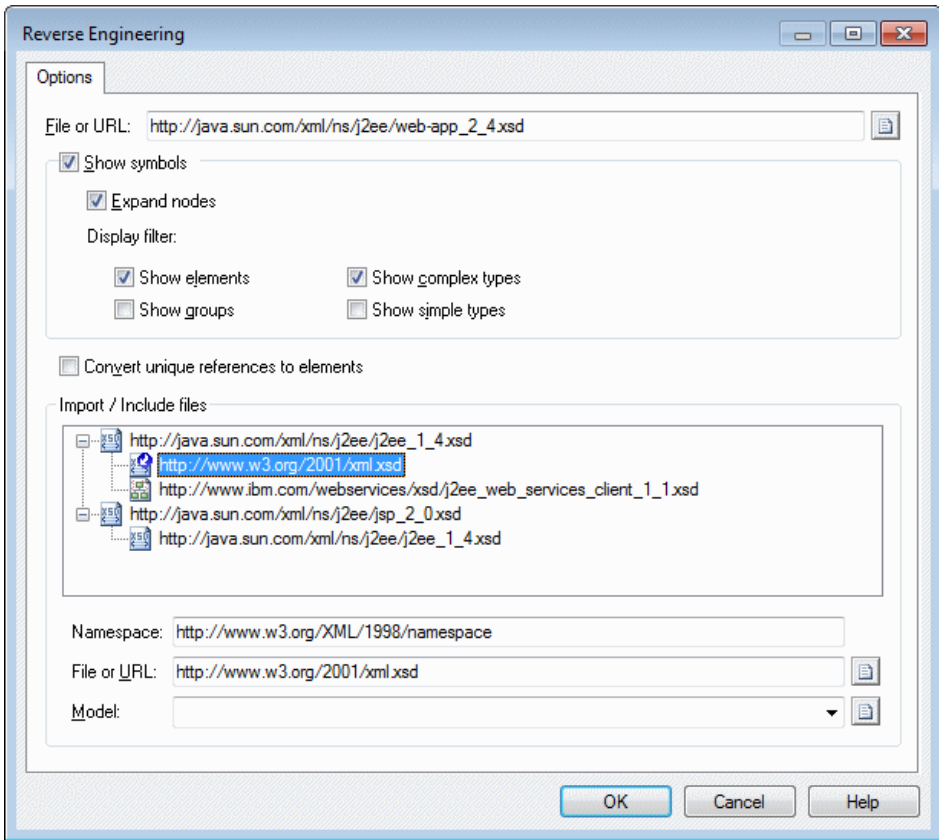
**4.** When you have resolved all references and are satisfied, click **OK** to begin reverse engineering.

If you are reverse engineering to an existing XSM, then the Merge Models dialog box opens to allow you to control the merging of the new objects into your XSM (see *Core Features Guide > Modeling with PowerDesigner > Comparing and Merging Models*).

When the process is complete, a confirmation message is given in the Output window. The principal XML file is created, and a separate XSM is created for each file that is included or imported. Each import and include is created as an object in the appropriate model (see *Instructions: Import, Include and Redefine (XSM)* on page 61), and shortcuts are created to reference the elements, types, or other objects defined in the imported or included schemas.

# Generating Other Models from an XSM

You can generate physical data models (PDMs) and other XSMs from an XSM.

1. Select **Tools**, and then one of the following commands to open the appropriate Model Generation Options window:
   - **Generate Physical Data Model... Ctrl+Shift+P**
   - **Generate XML Model... Ctrl+Shift+M**
2. On the **General** tab, select a radio button to generate a new or update an existing model, and complete the appropriate options.
3. [optional] Click the **Detail** tab and set any appropriate options. We recommend that you select the **Check model** checkbox to check the model for errors and warnings before generation.
4. [optional] Click the **Target Models** tab and specify the target models for any generated shortcuts.
5. [optional] Click the **Selection** tab and select or deselect objects to generate.
6. Click **OK** to begin generation.

   **Note:** For detailed information model generation, see *Core Features Guide > Linking and Synchronizing Models > Generating Models and Model Objects*.

The following table details how XSM objects are generated to PDM objects:

| XSM | PDM |
|---|---|
| Elements | Tables or columns:<br><br>• Root elements - are generated as tables.<br>• Non-root elements with complex types - are generated as tables or columns, depending on the option chosen in the **Persistent** groupbox on the **Detail** tab of the element property sheet.<br>• Non-root elements with primitive or simple types - are generated as table columns.<br><br>**Note:** Root elements with a primitive or simple type are not generated except where they are referenced by other elements or complex types.<br><br>If you have a single root element and want to generate its immediate children as tables, select the **Skip single root element** option on the PDM Generation Options window **Detail** tab. |

| XSM | PDM |
|---|---|
| Simple types | Domains. The datatype of the domain depends on the derivation of the simple type:<br><br>• simple types with a list derivation - `varchar`.<br>• simple types with a restriction derivation - the datatype of the base type<br>• simple types with a union derivation - the most permissive of the unioned types |
| Complex types | Merged with their parent element.<br><br>If the complex type is the restriction or extension of a simple type it will be generated as a column called `Value` linked to the domain generated from the simple type. |
| Attributes | Columns with datatypes determined by resolving any derivation. Attributes and attribute groups defined at the model level are not generated except where they are referenced. |
| Business rules | Business rules |
| Key constraints | Keys |
| Unique constraints | Indexes |
| Keyref constraints | References (if the referenced constraint is a key) |
| IDs (DTD) | Keys |

**Note:** References, substitutions, imports, and includes are always resolved, and attributes and attribute groups defined at the model level are generated only where they are used. Notations, redefines, anys, and (for DTDs) entities, are not generated to PDMs. To view the mappings between your XML objects and the objects generated from them, open the Mapping Editor from the generated model (see *Core Features Guide > Linking and Synchronizing Models > Object Mappings*).

CHAPTER 4    **Checking an XSM**

The XML model is a very flexible tool, which allows you quickly to develop your model without constraints. You can check the validity of your XSM at any time.

A valid XSM conforms to the following kinds of rules:

- Each complex type should have at least one attribute
- Each group must contain elements, groups, group particles and/or Any

**Note:** We recommend that you check your XML model before generating an XML document or another model from it . If the check encounters errors, generation will be stopped. The **Check model** option is enabled by default in the Generation dialog box.

You can check your model in any of the following ways:

- Press F4, or
- Select **Tools > Check Model**, or
- Right-click the diagram background and select Check Model from the contextual menu

The Check Model Parameters dialog opens, allowing you to specify the kinds of checks to perform, and the objects to apply them to. The following sections document the XSM -specific checks available by default. For information about checks made on generic objects available in all model types and for detailed information about using the Check Model Parameters dialog, see *Core Features Guide > Modeling with PowerDesigner > Objects > Checking Models*.

## Group Particle Checks

PowerDesigner provides default model checks to verify the validity of group articles.

| Check | Description and Correction |
|---|---|
| Existence of particle | A group particle must contain elements, groups, group particles and/or Any.<br><br>- Manual correction: Add items to the group particle or delete it<br>- Automatic correction: None |

| Check | Description and Correction |
|-------|---------------------------|
| Invalid cardinality | You should define a minimum (0 or 1) and a maximum cardinality (1 or unbounded) for a group particle occurrence.<br><br>This check is only available in a model targeted with XDR.<br><br>• Manual correction: Double-click the group particle symbol and type a value for Minimum (0 or 1) and Maximum (1 or unbounded) properties<br>• Automatic correction: None |

# Model Checks

PowerDesigner provides default model checks to verify the validity of models built on a schema.

| Check | Description and Correction |
|-------|---------------------------|
| Identifier uniqueness | Two or more objects cannot have the same identifier (ID).<br><br>• Manual correction: Give a unique identifier to each object<br>• Automatic correction: None |
| Undefined identifier | You must define an identifier (ID) for each object in the model.<br><br>• Manual correction: Define an identifier for each object<br>• Automatic correction: None |
| Shortcut code uniqueness | Two shortcuts with the same code cannot be in the same namespace.<br><br>• Manual correction: Change the code of one of the shortcuts<br>• Automatic correction: None |
| Undefined target namespace | You should define a target namespace to your model.<br><br>• Manual correction: Type a URI for the Target Namespace property in the Detail tab of the model property sheet<br>• Automatic correction: None |
| Missing namespaces | There should be at least one namespace defined for the model.<br><br>• Manual correction: Type a URI and a prefix in the Namespaces tab of the model property sheet<br>• Automatic correction: Adds the target namespace URI and a prefix "ns" followed by a number (e.g. "ns1") |

# Data Source Checks

PowerDesigner provides default model checks to verify the validity of data sources.

| Check | Description and Correction |
|-------|----------------------------|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |
| Existence of model | A data source must have at least one model in its definition.<br><br>• Manual correction: Add a model from the Models tab of the data source property sheet<br>• Automatic correction: Deletes data source without a model |
| Data source containing models with different Object Language or DBMS types | The models in a data source represent a single set of information. This is why the models in the data source should share the same DBMS or object language.<br><br>• Manual correction: Delete models with different DBMS or object language, or modify the DBMS or object language of models in the data source<br>• Automatic correction: None |

## Entity Checks

PowerDesigner provides default model checks to verify the validity of entities.

| Check | Description and Correction |
|---|---|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |
| Undefined entity | You must define an entity. In the entity property sheet, you must either type a value (string of characters or URI) in the Value box, or a URI in the Public or System boxes.<br><br>• Manual correction: Type a value in the Value box or a URI in the Public or System boxes<br>• Automatic correction: None |

## Include Checks

PowerDesigner provides default model checks to verify the validity of includes.

| Check | Description and Correction |
|---|---|
| Undefined schema location | You must define a schema location for an include.<br><br>• Manual correction: Define a URI or select a schema file for the schema location. For example: proforma.xsd<br>• Automatic correction: None |

## Simple Type Checks

PowerDesigner provides default model checks to verify the validity of simple types.

| Check | Description and Correction |
|---|---|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |

## Complex Type Checks

PowerDesigner provides default model checks to verify the validity of complex types.

| Check | Description and Correction |
|---|---|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |

| Check | Description and Correction |
|---|---|
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |
| Existence of attribute | A complex type should have at least one attribute.<br><br>• Manual correction: Define an attribute for the complex type<br>• Automatic correction: None |
| Existence of particle | A complex type must contain elements, groups, group particles and/or Any.<br><br>• Manual correction: Add items to the complex type or delete complex type<br>• Automatic correction: None |

# Element Checks

PowerDesigner provides default model checks to verify the validity of elements.

| Check | Description and Correction |
|---|---|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |
| Undefined type | An element without a reference should have a defined data type.<br><br>• Manual correction: In the element property sheet, define a data type with the Type list or the Browse tool<br>• Automatic correction: None |

| Check | Description and Correction |
|---|---|
| Undefined reference | An element without a defined data type must have a reference.<br><br>• Manual correction: In the element property sheet, define a reference with the Reference list or the Browse tool<br>• Automatic correction: None |
| Existence of attribute | An element without a reference, a data type or a substitution group should have at least one attribute.<br><br>• Manual correction: Define an attribute for the element<br>• Automatic correction: None |
| Existence of particle | An element with an embedded complex type must contain child elements, groups, group particles and/or Any.<br><br>• Manual correction: Add items to complex element or delete complex element<br>• Automatic correction: None |
| Invalid cardinality | [only available for model targeted with XDR] You should define a minimum (0 or 1) and a maximum cardinality (1 or unbounded) for a group particle occurrence.<br><br>• Manual correction: Double-click the group particle symbol and type a value for Minimum (0 or 1) and Maximum (1 or unbounded) properties<br>• Automatic correction: None |

## Group Checks

PowerDesigner provides default model checks to verify the validity of groups.

| Check | Description and Correction |
|---|---|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |

| Check | Description and Correction |
|---|---|
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |
| Undefined reference | A group without a name or a code must have a reference.<br><br>• Manual correction: In the group property sheet, define a reference with the Reference list or the Browse tool<br>• Automatic correction: None |
| Existence of group particle | A group must contain elements, groups, group particles and/or Any.<br><br>• Manual correction: Add items to group or delete group<br>• Automatic correction: None |
| Invalid cardinality | [only available for model targeted with XDR] You should define a minimum (0 or 1) and a maximum cardinality (1 or unbounded) for a group particle occurrence.<br><br>• Manual correction: Double-click the group particle symbol and type a value for Minimum (0 or 1) and Maximum (1 or unbounded) properties<br>• Automatic correction: None |

## Attribute Checks

PowerDesigner provides default model checks to verify the validity of attributes.

| Check | Description and Correction |
|---|---|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |

| Check | Description and Correction |
|---|---|
| Undefined reference | An attribute without a name or a code must have a reference.<br><br>• Manual correction: In the attribute property sheet, define a reference with the Reference list or the Browse tool<br>• Automatic correction: None |
| Undefined type | You must define a data type for an attribute.<br><br>• Manual correction: In the attribute property sheet, define a data type with the Type list or the Browse tool<br>• Automatic correction: None |

## Notation Checks

PowerDesigner provides default model checks to verify the validity of notations.

| Check | Description and Correction |
|---|---|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |
| Undefined notation | A notation must have at least one URI defined for Public or System properties.<br><br>• Manual correction: In the notation property sheet, define a URI in the Public or System boxes<br>• Automatic correction: None |

## Attribute Group Checks

PowerDesigner provides default model checks to verify the validity of attribute groups.

| Check | Description and Correction |
|---|---|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |
| Undefined reference | An attribute group without a name or a code must have a reference.<br><br>• Manual correction: In the attribute group property sheet, define a reference with the Reference list or the Browse tool<br>• Automatic correction: None |
| Existence of attributes | An attribute group must contain at least one attribute.<br><br>• Manual correction: Add attributes to attribute group or delete attribute group<br>• Automatic correction: Deletes unassigned attribute group |

## Import Checks

PowerDesigner provides default model checks to verify the validity of imports.

| Check | Description and Correction |
|---|---|
| Undefined schema location and namespace | An import must have at least a schema location or a namespace defined.<br><br>• Manual correction: In the import property sheet, define a URI for the schema location and/or the namespace.<br>• Automatic correction: None |

## Redefine Checks

PowerDesigner provides default model checks to verify the validity of redefines.

| Check | Description and Correction |
|---|---|
| Undefined schema location | You must define a schema location for a redefine.<br><br>• Manual correction: In the redefine property sheet, define a URI or select a schema file for the schema location. For example: customers.xsd<br>• Automatic correction: None |
| Existence of component | A redefine must contain at least one of the following items: simple type, complex type, group or attribute group.<br><br>• Manual correction: Add items to the redefine<br>• Automatic correction: None |

## Key Checks

PowerDesigner provides default model checks to verify the validity of keys.

| Check | Description and Correction |
|---|---|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |

| Check | Description and Correction |
|---|---|
| Existence of fields | A key must contain at least one field.<br><br>• Manual correction: Add at least one field to the key or delete the key. For example: @numEmployee<br>• Automatic correction: Deletes unassigned key<br><br>For more information on fields, see *Constraint Properties* on page 48. |
| Undefined selector | You must define an XPath expression for a key selector attribute.<br><br>• Manual correction: In the key property sheet, define an XPath expression for the selector attribute. For example: s:company/s:employee<br>• Automatic correction: None<br><br>For more information on XPath expressions, see *Constraint Properties* on page 48. |

## KeyRef Checks

PowerDesigner provides default model checks to verify the validity of KeyRefs.

| Check | Description and Correction |
|---|---|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |
| Undefined reference | A keyRef must contain a reference to a key or a unique constraint.<br><br>• Manual correction: In the keyRef property sheet, define a reference to a key or a unique constraint with the Reference list.<br>• Automatic correction: None |

| Check | Description and Correction |
|---|---|
| Existence of fields | A keyRef must contain at least one field.<br><br>• Manual correction: Add at least one field to the keyRef or delete the keyRef. For example: @numEmployee.<br>• Automatic correction: Deletes unassigned keyRef.<br><br>For more information on fields, see *Constraint Properties* on page 48. |
| Undefined selector | You must define an XPath expression for a keyRef selector attribute.<br><br>• Manual correction: In the keyRef property sheet, define an XPath expression for the selector attribute. For example: s:company/s:employee.<br>• Automatic correction: None<br><br>For more information on XPath expressions, see *Constraint Properties* on page 48. |

## Unique Checks

PowerDesigner provides default model checks to verify the validity of uniques.

| Check | Description and Correction |
|---|---|
| Name/Code contains terms not in glossary | [if glossary enabled] Names and codes must contain only approved terms drawn from the glossary.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - None. |
| Name/Code contains synonyms of glossary terms | [if glossary enabled] Names and codes must not contain synonyms of glossary terms.<br><br>• Manual correction - Modify the name or code to contain only glossary terms.<br>• Automatic correction - Replaces synonyms with their associated glossary terms. |
| Name/Code uniqueness | Object names must be unique in the namespace.<br><br>• Manual correction - Modify the duplicate name or code.<br>• Automatic correction - Appends a number to the duplicate name or code. |
| Existence of fields | A unique constraint must contain at least one field.<br><br>• Manual correction: Add at least one field to the unique constraint or delete the unique constraint. For example: @numEmployee.<br>• Automatic correction: Deletes unassigned unique constraint.<br><br>For more information on fields, see *Constraint Properties* on page 48. |

| Check | Description and Correction |
|---|---|
| Undefined Selector | You must define an XPath expression for a unique constraint selector attribute.<br><br>• Manual correction: In the unique constraint property sheet, define an XPath expression for the unique constraint selector attribute. For example: s:company/s:employee.<br>• Automatic correction: None<br><br>For more information on XPath expressions, see *Constraint Properties* on page 48. |

## Extension Checks

PowerDesigner provides default model checks to verify the validity of extensions.

| Check | Description and Correction |
|---|---|
| Undefined base type | You must define a base type when you derive a complex type by extension.<br><br>• Manual correction: In the complex type property sheet, click the Properties tool beside the Derivation box to display the Extension property sheet and select a base type with the Base type list or the Browse tool<br>• Automatic correction: None |

## Restriction Checks

PowerDesigner provides default model checks to verify the validity of restrictions.

| Check | Description and Correction |
|---|---|
| Undefined base type | You must define a base type when you derive a simple or a complex type by restriction.<br><br>• Manual correction: In the simple or complex type property sheet, click the Properties tool beside the Derivation box to display the Extension property sheet and select a base type with the Base type list or the Browse tool<br>• Automatic correction: None |

| Check | Description and Correction |
|---|---|
| Existence of facet | A simple type restriction must have at least one facet defined. Facets are defined in the Detail, Enumerations and Patterns tabs of a simple type restriction property sheet.<br><br>• Manual correction: Define one or more facets in the simple type restriction property sheet<br>• Automatic correction: None |

## Simple Type List Checks

PowerDesigner provides default model checks to verify the validity of simple types.

| Check | Description and Correction |
|---|---|
| Undefined base type | You must define a base type when you derive a simple type by list.<br><br>• Manual correction: In the simple type property sheet, click the Properties tool beside the Derivation box to display the simple type list property sheet and select a data type with the Type list or the Browse tool<br>• Automatic correction: None |

## Simple Type Union Checks

PowerDesigner provides default model checks to verify the validity of simple type unions.

| Check | Description and Correction |
|---|---|
| Undefined base type | You must define at least two data types when you derive a simple type by union.<br><br>• Manual correction: In the simple type property sheet, click the Properties tool beside the Derivation box to display the simple type union property sheet and type a white space separated list of at least two data types (qualified names) in the Member types box<br>• Automatic correction: None |

## Annotation Checks

PowerDesigner provides default model checks to verify the validity of annotations.

| Check | Description and Correction |
|---|---|
| Existence of items | An annotation must contain at least one URI for a Documentation or an Application Information. <br><br> • Manual correction: Define a URI for a Documentation or an Application Information <br> • Automatic correction: None |

CHAPTER 5 **Working with XML and Databases**

Many relational databases now support XML so that you can store or retrieve data through XML files. You can use an XML model to generate an *annotated schema* that will allow you to store or retrieve data in such a database.

The following databases are available :

| Database | Mapped XML model | Targeted XML language | Required XEM file |
|---|---|---|---|
| Microsoft SQL Server 2000 and higher | Yes | XSD or XDR | Microsoft SQL Server |
| Oracle 9i2 and higher | No | XSD | Oracle 9i2 |
| IBM DB2 v8.1 and higher | Yes | DTD | IBM DB2 DAD |

By attaching the SQL/XML extensions to an XML model mapped to a PDM, you can also generate *SQL/XML queries* to retrieve data in an XML format, from relational databases supporting SQL/XML.

**Note:** You can also generate PDM tables from an XML schema. For more information, see *Generating Other Models from an XSM* on page 69.

## Generating an SQL/XML Query File

SQL/XML is an XML extension of the Structured Query Language, which allows you to retrieve relational data using extended SQL syntax, and produce an XML result. You can generate SQL/XML queries for *global elements* in your XSM, whatever the targeted XML language (XSD, DTD or XDR).

SQL/XML has five main elements:

- *XMLELEMENT* - to edit an element with a name, a list of attributes (optional) and a list of values (optional)
- *XMLATTRIBUTES* - to edit a list of attributes with names and values
- *XMLAGG* - to edit in multiple rows a concatenation of elements, from a single XML value corresponding to a single column
- *XMLCONCAT* - to edit in the same row a concatenation of elements, from several XML values corresponding to several columns

- *XMLFOREST* - to edit in the same row a concatenation of elements, from several SQL values corresponding to several columns. The name and value of a column become the name and value of an element

**Warning!** The following procedure assumes you have an XML model open in the workspace and mapped to a PDM. Generated SQL/XML queries cannot be parameterized.

1. To enable the SQL/XML extensions in your model, select **Model > Extensions**, click the **Attach an Extension** tool, select the `SQL/XML` file (on the **General Purpose** tab), and click **OK** to attach it.
2. Select **Tools > Generate SQL/XML Queries** to open the Generation dialog box.
3. Specify the directory in which to generate the file.
4. Click the Selection tab and specify which of the global elements you want to generate queries from. A separate file will be generated for each global element selected.
5. Click OK to begin the generation.

   The Result dialog box is displayed with the path of the query file selected.
6. Click Edit to open the generated query file in your associated editor:

```
select '<?xml version="1.0" encoding="UTF-8" ?>' ||
XMLELEMENT( NAME "Root",
(select XMLAGG ( XMLELEMENT( NAME "DEPARTMENT", XMLATTRIBUTES (DEPARTMENT.DEPNUM
 (select XMLAGG ( XMLELEMENT( NAME "EMPLOYEE", XMLATTRIBUTES (EMPLOYEE.DEPNUM AS
  from EMPLOYEE
  where DEPARTMENT.DEPNUM = EMPLOYEE.DEPNUM)) )
 from DEPARTMENT))
```

# Generating an Annotated Schema for Microsoft SQL Server

Microsoft SQL Server is an XML-enabled database server, which supports annotations that can be used on XSD or XDR files, to map XML data to relational data.

An *annotated schema* is an XML file that allows you to store or retrieve data in an XML format, from relational databases supporting XML. An XML model allows you to generate an annotated schema (XSD or XDR) for SQL Server 2000.

1. Map an XSM to a PDM. You can do this manually or by generating an XSM from a PDM (or a PDM from an XSM).
2. To enable the Microsoft SQL Server extensions in your model, select **Model > Extensions**, click the **Attach an Extension** tool, select the `Microsoft SQL Server` file (on the **XML in Database** tab), and click **OK** to attach it.
3. [optional] Reinforce the mappings of elements and attributes to tables and columns with extended attributes:

**Note:** If the element and attribute names match the table and column names, you do not need to define extended attributes for XML objects.

| Annotation | Description |
|---|---|
| encode | When an XML element or attribute is mapped to a SQL Server BLOB column, allows requesting a reference (URI) to be returned and used later to return BLOB data. <br><br> Available for: Element, Attribute |
| field | Maps an XML item to a database column. <br><br> Available for: Element, Attribute |
| hide | Hides the element or attribute specified in the schema in the resulting XML document. <br><br> Available for: Element, Attribute |
| is-constant | Creates an XML element that does not map to any table. The element is displayed in the query output. <br><br> Available for: Element |
| key-fields | Allows specification of columns that uniquely identify the rows in a table. <br><br> Available for: Element |
| limit-field | Allows limiting the values that are returned on the basis of a limiting value. <br><br> Available for: Element, Attribute |
| limit-value | Allows limiting the values that are returned on the basis of a limiting value. <br><br> Available for: Element, Attribute |
| mapped | Allows schema items to be excluded from the result. <br><br> Available for: Element, Attribute |
| max-depth | Allows you to specify depth in recursive relationships that are specified in the schema. <br><br> Available for: Element |
| overflow-field | Identifies the database column that contains the overflow data. <br><br> Available for: Element |
| relation | Maps an XML item to a database table. <br><br> Available for: Element |
| relationship-child | Specifies an element as the *child table* in a reference (To define only in the child element property sheet). <br><br> Available for: Element |

| Annotation | Description |
|---|---|
| relationship-child-key | Specifies an attribute as the *foreign key* of a child table in a reference (To define only in the child element property sheet).<br><br>Available for: Element |
| relationship-parent | Specifies an element as the *parent table* in a reference (To define only in the child element property sheet).<br><br>Available for: Element |
| relationship-parent-key | Specifies an attribute as the *primary key* of a parent table in a reference (To define only in the child element property sheet).<br><br>Available for: Element |
| use-cdata | Allows specifying CDATA sections to be used for certain elements in the XML document.<br><br>Available for: Element |
| prefix | Creates valid XML ID, IDREF, and IDREFS. Prepends the values of ID, IDREF, and IDREFS with a string.<br><br>Available for: Attribute |

**4.** [optional] Click the **Preview** tab of the model property sheet, to preview the annotated schema.

**5.** Select **Language > Generate** *schema***File** to open the Generation dialog box.

**6.** Specify the directory in which to generate the file and select the XML in Database target on the Targets tab.
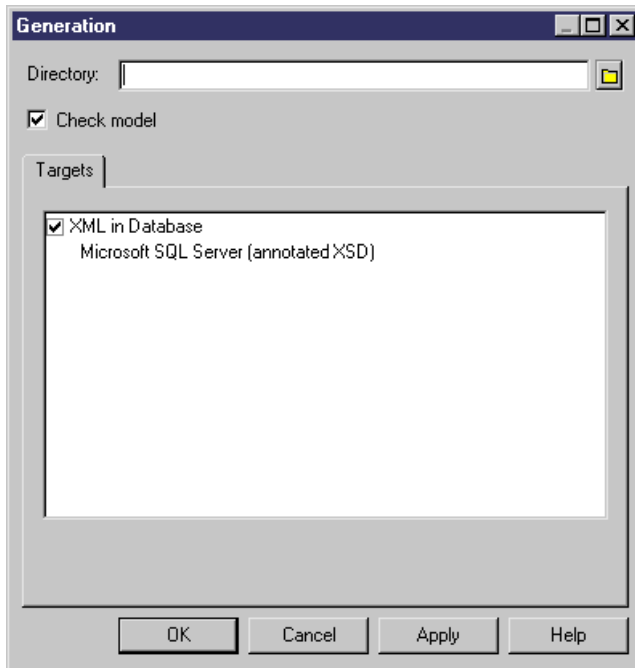
7. Click OK to begin the generation.

The Result dialog box is displayed with the path of the annotated schema file selected.

8. Click Edit to open the generated annotated schema in your associated editor:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
    <xs:element name="Root" sql:is-constant="1">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="DEPARTMENT" sql:relation="DEPARTMENT">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="EMPLOYEE" sql:relation="EMPLOYEE">
                                <xs:annotation>
                                    <xs:appinfo>
                                        <sql:relationship parent="DEPARTMENT" parent-key="DEPNUM" child-key="DEPNUM" child="EMPLOYEE"/>
                                    </xs:appinfo>
                                </xs:annotation>
                                <xs:complexType>
                                    <xs:attribute name="DEPNUM" type="xs:int" sql:field="DEPNUM">
                                    </xs:attribute>
                                    <xs:attribute name="EMPID" type="xs:int" sql:field="EMPID">
                                    </xs:attribute>
                                    <xs:attribute name="FIRSTNAME" type="xs:string" sql:field="FIRSTNAME">
                                    </xs:attribute>
                                    <xs:attribute name="LASTNAME" type="xs:string" sql:field="LASTNAME">
                                    </xs:attribute>
                                </xs:complexType>
                            </xs:element>
                        </xs:sequence>
                        <xs:attribute name="DEPNUM" type="xs:int" sql:field="DEPNUM">
                        </xs:attribute>
                        <xs:attribute name="DEPNAME" type="xs:string" sql:field="DEPNAME">
                        </xs:attribute>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Note the SQL namespace (with the sql prefix) and the SQL annotations for tables (sql:relation), columns (sql:field) and reference (sql:relationship).

# Generating an Annotated Schema for Oracle

Oracle 9i2 is a database server with a native XML storage and retrieval technology called *Oracle XML DB*. There is no mapping between XML data and relational data. Tables, columns and abstract data types (ADT) are created from *annotated schemas* (XSDs). Annotated schemas are XML-coded files, targeted with an XML language and tagged with specific DBMS annotations, that allow you to store or retrieve data in an XML format, from relational databases supporting XML.

An XML model allows you to generate an annotated schema (XSD) for Oracle 9i2. Oracle 9i2 uses by default the name of the XML elements present in the annotated schema to generate SQL objects. You can override the creation of SQL objects by defining *extended attributes* for elements, complex types and the XML model.

1. To enable the Oracle extensions in your model, select **Model > Extensions**, click the **Attach an Extension** tool, select the `Oracle XML DB` (on the **XML in Database** tab), and click **OK** to attach it.
2. [optional] Specify the following properties on the **Extended Attributes** tab of the property sheets of elements:

| Annotation | Description |
| --- | --- |
| beanClassname | Can be used within element declarations. If the element is based on a global complexType, this name must be identical to the beanClassname value within the complexType declaration. If a name is specified by the user, the bean generation will generate a bean class with this name, instead of generating a name from the element name |
| columnProps | Specifies the column storage clause that is inserted into the default CREATE TABLE statement. It is useful mainly for elements that are mapped to tables, namely top-level element declarations and out-of-line element declarations |
| defaultTable | Specifies the name of the table into which XML instances of this schema should be stored. This is most useful in cases when the XML is being inserted from APIs where table name is not specified (for example, FTP and HTTP) |
| javaClassname | Used to specify the name of a Java class that is derived from the corresponding bean class, to ensure that an object of this class is instantiated during bean access. If a JavaClassname is not specified, Oracle XML DB will instantiate an object of the bean class directly |

| Annotation | Description |
|---|---|
| maintainDOM | If true, instances of this element are stored so that they retain DOM fidelity on output. This implies that all comments, processing instructions, namespace declarations, and so on, are retained in addition to the ordering of elements. If false, the output need not be guaranteed to have the same DOM behavior as the input |
| maintainOrder | If true, the collection is mapped to a VARRAY. If false, the collection is mapped to a NESTED TABLE |
| SQLCollSchema | Name of the database user owning the type specified by SQLCollType |
| SQLCollType | Specifies the name of the SQL collection type corresponding to this XML element that has maxOccurs > 1 |
| SQLInline | If true this element is stored inline as an embedded attribute (or a collection if maxOccurs > 1). If false, a REF (or collection of REFs if maxOccurs > 1) is stored. This attribute will be forced to false in certain situations (like cyclic references) where SQL will not support inlining |
| SQLName | Specifies the name of the attribute within the SQL object that maps to this XML element |
| SQLSchema | Name of the database user owning the type specified by SQLType |
| SQLType | Specifies the name of the SQL type corresponding to this XML element declaration |
| tableProps | Specifies the TABLE storage clause that is appended to the default CREATE TABLE statement. This is meaningful mainly for global and out-of-line elements |

**3.** [optional] Specify the following properties on the **Extended Attributes** tab of the property sheets of complex types:

| Annotation | Description |
|---|---|
| beanClassname | Can be used within element declarations. If the element is based on a global complexType, this name must be identical to the beanClassname value within the complexType declaration. If a name is specified by the user, the bean generation will generate a bean class with this name, instead of generating a name from the element name |
| SQLSchema | Name of the database user owning the type specified by SQLType |
| SQLType | Specifies the name of the SQL type corresponding to this XML element declaration |

**4.** [optional] Specify the following properties on the **Extended Attributes** tab of the property sheets of the model:

| Annotation | Description |
|---|---|
| mapUnboundedStringToLob | If true, unbounded strings are mapped to CLOB by default. Similarly, unbounded binary data get mapped to BLOB, by default. If false, unbounded strings are mapped to VAR-CHAR2(4000), and unbounded binary components are mapped to RAW(2000) |
| storeVarrayAsTable | If true, the VARRAY is stored as a table (OCT). If false, the VARRAY is stored in a LOB |

**5.** Select **Language > Generate** *schema* File to open the Generation dialog box.

**6.** Specify the directory in which to generate the file and select the XML in Database target on the Targets tab.
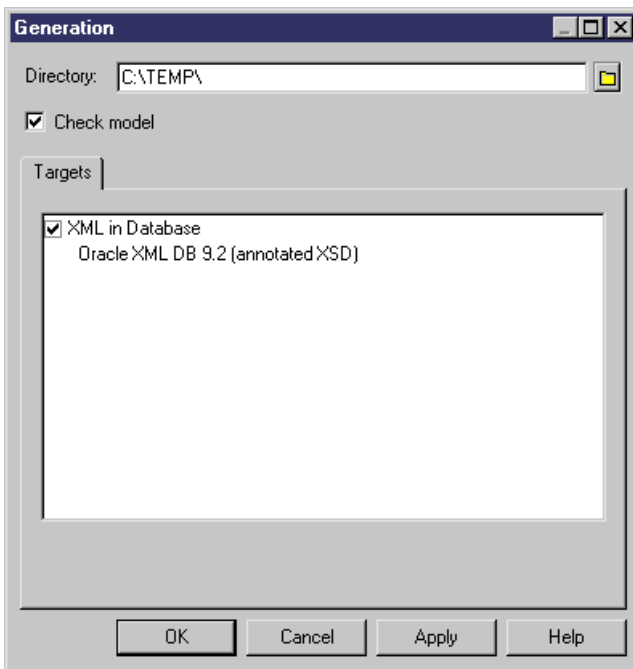


**7.** Click OK to begin the generation.

The Result dialog box is displayed with the path of the annotated schema file selected.

**8.** Click Edit to open the generated annotated schema in your associated editor:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:sql="http://xmlns.oracle.com/xdb">
    <xs:element name="Branch"  sql:SQLName="branch">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Department" type="DepType"  sql:SQLName="office"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:complexType name="DepType"  sql:SQLType="officeType">
    </xs:complexType>
</xs:schema>
```

Note the Oracle namespace (with the sql prefix) and annotations for tables
(sql:SQLName) and ADTs (sql:SQLType)

# Generating a DAD File for IBM DB2

IBM DB2 v8.1 (or higher) is a database server with an add-in for XML storage and retrieval
called IBM DB2 Extender. XML data (elements, attributes) are mapped to relational data
(tables, columns) through Document Access Definition files (.DAD).

There are three types of DAD files:

| Storage Type | Description |
|---|---|
| Xcolumn | Column mapping - the Root element is mapped to a table, and its attributes or child elements are mapped to columns identified by an XPath |
| Xcollection | SQL mapping - the DAD file starts with a SQL statement for the table mapped to the Root element, and each child element or attribute is mapped to a column or a table name |
| Xcollection | RDB mapping - a Relational Database node, with a table and a column name, is associated with each attribute or child element of the Root element |

An XML model targeted with DTD allows you to generate DAD files for IBM DB2.

1. Map an XSM to a PDM. You can do this manually or by generating an XSM from a PDM
   (or a PDM from an XSM).
2. To enable the IBM DB2 DAD extensions in your model, select **Model > Extensions**, click
   the **Attach an Extension** tool, select the IBM DB2 DAD file (on the **XML in Database**
   tab), and click **OK** to attach it..
3. [optional] Set extended attributes on global elements to reinforce their mapping to tables
   and columns The following properties are listed on the **Extended Attributes** tab:

| Extended attribute | Description |
|---|---|
| Database | Name of the database |

---

| Extended attribute | Description |
|---|---|
| DTDID | ID added to the DTD_ref system table in DB2 XML Extender |
| Login | Name of the logged-in user |
| MappingType | Type of mapping for a collection |
| NamespaceNode | Text zone where each line describes a namespace couple (name = value). The separator character is '=' |
| Password | Password of the logged-in user |
| PathGeneration | Generation path |
| ProcessInstruction | A text zone that enables the user to enter some instruction |
| SideTableID | Identifier of the side table (optional) |
| SideTableName | Name of the side table |
| StorageName | If StorageType is Xcolumn, then it is the name of the sidetable column |
| StorageType | Type of storage (Xcollection or Xcolumn) |

4. [optional] Click the **Preview** tab of the Root element property sheet, and select the *DB2XMLExtender.DAD File* tab to preview the DAD file. If the DAD File tab is not available, click the **Select Generation Targets** tool to select IBM DB2 DAD in the **Targets** list and click **OK**.

5. Select **Language > Generate** *schema***File** to open the Generation dialog box.

6. Specify the directory in which to generate the file and select the XML in Database target on the Targets tab.

**7.** [optional] Click the Options tab, and set any appropriate generation options:

| Option | Description |
|---|---|
| Character ending an instruction | Character ending instructions in the SQL file for stored procedures |
| Generates procedures deployment | Generation of a SQL script for stored procedures enabling XML data storage and facilitating XML data retrieval |
| Path of DAD.dtd | Path of the DTD file installed with IBM DB2 Extender and describing the specific syntax of DAD files |
| Schema validation | Validation tag in the DAD files to check the conformity of DAD files with the DAD syntax |

**8.** Click OK to begin the generation.

The Result dialog box is displayed with the path of the generated DAD, DTD and SQL files.

**9.** Click Edit to open the generated DAD file in your associated editor:

- Extract of a DAD file defined with *Xcollection* as StorageType, and *RDB* as MappingType:

```
<!DOCTYPE DAD SYSTEM "E:\dad.dtd">
<DAD>
<validation>YES</validation>
<Xcollection>
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Root SYSTEM "C:\TEMP\DTD_files\CorporateMembership.dtd"</doctype>
<root_node>
<element_node name="Root">
    <element_node name="DEPARTMENT">
        <attribute_node name="DEPNUM">
            <RDB_node>
                <table name="DEPARTMENT"/>
                <column name="DEPNUM" type="INTEGER"/>
            </RDB_node>
        </attribute_node>
        <attribute_node name="DEPNAME">
            <RDB_node>
                <table name="DEPARTMENT"/>
                <column name="DEPNAME" type="VARCHAR(254)"/>
            </RDB_node>
        </attribute_node>
        <element_node name="EMPLOYEE">
            <attribute_node name="DEPNUM">
                <RDB_node>
                    <table name="EMPLOYEE"/>
                    <column name="DEPNUM" type="INTEGER"/>
                </RDB_node>
            </attribute_node>
```

- DAD file defined with *Xcolumn* as StorageType:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "E:\dad.dtd">
<DAD>
<validation>YES</validation>
<Xcolumn>
<table name="DEPARTMENT" key="DEPNUM" orderBy="DEPNUM, , DEPNAME">
    <column name="DEPNUM"
            type="INTEGER"
            path="/DEPARTMENT/@DEPNUM"
            multi_occurrence="NO"/>
    <column name="DEPNAME"
            type="VARCHAR(254)"
            path="/DEPARTMENT/@DEPNAME"
            multi_occurrence="NO"/>
</table>
<table name="EMPLOYEE" key="EMPID" orderBy="EMPID, DEPNUM, FIRSTNAME, LASTNAME">
    <column name="DEPNUM"
            type="INTEGER"
            path="/DEPARTMENT/EMPLOYEE/@DEPNUM"
            multi_occurrence="NO"/>
    <column name="EMPID"
            type="INTEGER"
            path="/DEPARTMENT/EMPLOYEE/@EMPID"
            multi_occurrence="NO"/>
    <column name="FIRSTNAME"
            type="CHAR(1)"
            path="/DEPARTMENT/EMPLOYEE/@FIRSTNAME"
            multi_occurrence="NO"/>
    <column name="LASTNAME"
            type="CHAR(1)"
            path="/DEPARTMENT/EMPLOYEE/@LASTNAME"
            multi_occurrence="NO"/>
</table>
</Xcolumn>
</DAD>
```

# Index

## A

## B

## C

## D

SAP Sybase PowerDesigner