SYBASE®

# XML Features in PowerBuilder® 9.0

A white paper from Sybase, Inc.

WHITE PAPER

# Table of Contents

**1. Overview . . . . . 1**
**2. DataWindow Export Engine . . . . . 1**
Export Templates—Graphical User Interface . . . . . 2
Export Template Header Section . . . . . 3
Export Template Detail Section . . . . . 4
Mapping DataWindow Elements to XML Nodes . . . . . 5
Exporting XML . . . . . 5
Importing XML . . . . . **12**
**3. Conclusion . . . . . 13**

# 1. Overview

XML is fast becoming the de-facto industry standard for inter-component and inter-application data transfer. Starting with version 9.0, new XML features will be available for PowerBuilder applications. These features include XML export and import capabilities to and from DataWindows (and DataStores), as well a new XML Parser interface called "Native XML Services".

This white paper will serve as an introduction to the new DataWindow XML export and import features.

# 2. DataWindow Export Engine

The DataWindow Export Engine is a new component of the DataWindow engine. It uses Export Templates as the infrastructure for exporting row data into XML format. Export templates are discussed in detail in the following section.

The Export Engine analyzes the export template to be used for the XML output. Since the export template is itself an XML document, it must be parsed first. To that end, the Export Engine uses the services of the XML Parser in order to analyze the Template. The XML Parser used is the C++ implementation or Apache Xerces. The Xerces parser is accessed via an adapter interface (pbxerces90.dll).

An export template defines the mapping between DataWindow elements and their XML counterparts.

The following DataWindow objects can be used in export templates:

- Column
- Computed Column
- Text Control
- Computed Field
- Nested Report

The above DataWindow objects can be mapped to the following XML constructs:

- Element
- Attribute

> *Note: comments and processing instructions can be added anywhere in the template, but cannot be mapped to. CDATA sections can be added anywhere under any element, but cannot be mapped to. Nested reports can only be mapped to elements, not attributes.*

As the Export Engine analyzes the XML Template, it maps elements from the template to actual DataWindow controls and their text values. After the mapping operation has been successfully completed, the engine will perform the final XML syntax generation, again with the help of the XML Parser/Generator Engine.

Export templates are a part of a DataWindow's definition. A DataWindow can contain multiple export templates. Templates are persisted in both PBL and SRD formats of a DataWindow. A new DataWindow property, Export.XML.UseTemplate, is used to specify a named template object for use in a given export operation. This property can be set at both runtime and design-time. The syntax for XML export templates is shown in Listing 1.

**Listing 1: XML export template syntax (indented for clarity)**

```
export.xml(
   usetemplate="t_orders"
   metadatatype=2 savemetadata=1
   template=(
      comment="Order items with external DTD"
      name="t_orders"
      publicid="c:\xmldw\orders.dtd"
      xml="<?xml version=~"1.0~"
                  encoding=~"UTF-8~"
                  standalone=~"no~"?>
            <!DOCTYPE Orders>
            <Orders><!-- omitted for brevity --></Orders>"
   )
)
```

## Export Templates—Graphical User Interface

The DataWindow painter in PowerBuilder 9.0 features a new view for defining and editing XML export templates. In the painter's default layout, this view is located to the left of the existing Column Specification, Data, and Control List views, and is titled "Export / Import Template - XML".

The new XML Export Template view contains a TreeView control that represents the XML structure of an export template. XML entities (markup and character data) are displayed as TreeView nodes, with a distinctive icon and font color to denote the type of entity. Character data is display as a child node of the respective element node, thus, element end-tags are not shown, but assumed. Markup delimiters are also implicit; i.e., angle brackets are not shown but assumed.

Only one export template can be opened in this view for editing at any given time. Figure 1 shows the new XML Export Template view in the DataWindow painter.

Figure 1: XML Export Template view in DataWindow painter

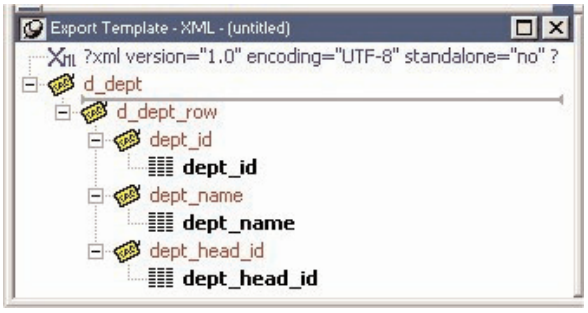


Table 1 lists all possible XML constructs that an export template can contain.

**Table 1: XML export template components**

| *Icon* | *XML Construct* |
|---|---|
|  | XML declaration |
|  | Document type declaration |
|  | Root element |
|  | Child element |
| | Attribute (no icon) |
|  | DataWindow column reference |
|  | Static text control reference |
|  | Computed field reference |
|  | DataWindow expression reference |
|  | Literal text |
|  | Comment |
|  | Processing instruction |
|  | CDATA sections |
|  | Nested Report |

## Export Template Header Section

Right-clicking the mouse anywhere on the TreeView control's background area brings up a context-sensitive popup menu that includes the following options:

| | |
|---|---|
| New | Create a new XML template with a root and an empty detail row element. |
| New Default | Create a new XML template with a root and a detailed row element with a child element mapped to each column and computed field control. |
| Open… | Open a previously saved XML template from a dialog for editing. |
| Save | Save the current XML template by name. If the current template is unnamed, a dialog is displayed, prompting for a name and optional comments. |
| Save As… | Save the current XML template as a copy under a new name via a dialog. |
| Delete | Deletes the current XML template. A confirmation dialog is displayed prior to this action, allowing the user to cancel the request. |

*Note: since the Delete menu gives the option to delete the current template only, the Open Template dialog allows for the deletion of any existing template using the Del key.*

Right-clicking any given node in the XML template TreeView control will produce one of a number of additional context menus, depending on the node type and relative location in the tree. For example, the context menu for the XML Declaration node displays the following options:

| | |
|---|---|
| Edit… | Display a dialog for editing the XML version, encoding and the standalone document settings. |
| Delete | Delete the XML declaration. Inserting it back can be done from context menu options on items after it. |

The context menu for the Document Type Declaration node displays the following options:

| | |
|---|---|
| Edit… | Display a dialog for editing the DOCTYPE name and identifier. |
| Insert Before } | Open a cascading menu listing all possible XML constructs allowed in this context. |
| Delete | Delete the Document Type Declaration. Inserting it back can be done from context menu options on items after it. |

*Note: if the current template already has an XML declaration node defined, the corresponding option on the Insert Before cascading menu is disabled.*

The context menu for the root node displays the following options:

| | |
|---|---|
| Edit… | Put the node label in edit mode. If the node has one or more attributes, open a dialog for editing the node name and adding or editing an attribute |
| Edit/Add Attribute... | Open a dialog for editing the node name and adding or editing an attribute. |
| Add Child } | Open a cascading menu listing all possible XML constructs that can be appended to the root node. |
| Insert Before } | Open a cascading menu listing all possible XML constructs that can be inserted before the root node. |
| Schema Options… | Open a dialog for editing the name of the document's root element in the generated inline XML schema, as well as the namespace prefix and URI. |

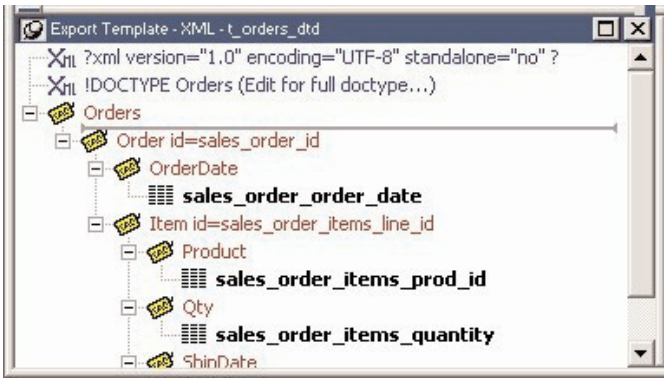
*Note: the XML Declaration and Document Type options on the Insert Before cascading menu will be disabled if the corresponding nodes are already present in the current template. The root node is required and cannot be deleted.*

### Export Template Detail Section

The Detail section is graphically delineated by a gray line across the TreeView control, separating the header section from the detail section, similar to the way a detail band functions in the DataWindow painter's design view. The separator line can be repositioned by the user by checking the 'Starts Detail' option on the context menu of the respective element node. This element is then referred to as the Detail Start element.

Only one element can be marked as the Detail Start element. The root node itself cannot be marked as a Detail Start element. By default, the first child of the root element is designated as the Detail Start element; however, any subsequent child node (and children thereof) may be marked as the Start Detail element, effectively moving all preceding nodes to the header section of the template. At export-time, only the Detail Start element, its sibling and children will be iteratively generated for each row. Figure 2 shows the detail section of an export template with the Order node marked as the Detail Start element.

Figure 2: Export template detail section



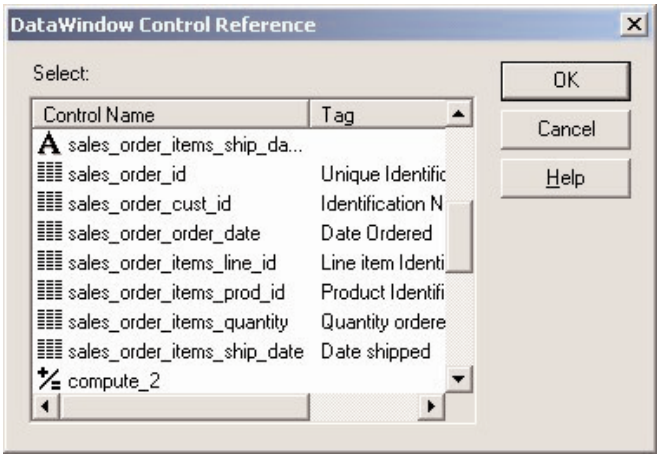

The context menu for element nodes in the detail section displays the following options, in addition to the ones displayed by the root node's context menu (and excluding the Schema Options… menu item):

| | |
|---|---|
| Starts Detail | Position the gray separator bar above the current node. |
| Delete | Delete the element node. |

### Mapping DataWindow Elements to XML Nodes

Once the structure of your target XML document is finalized, you need to bind data values to XML elements and attributes. The mapping process is done by selecting a DataWindow column, computed field, expression and/or text controls from a dialog. Alternatively, the user may drag and drop items from the control list painter view. The selected item appears as a child node under the respective element node in the TreeView control. Figure 10 shows the control reference dialog.

Figure 3: Control reference dialog




*Note: data values mapped to XML elements are treated as character data, regardless of actual data type (number, date, etc.) of the respective DataWindow column.*

### Exporting XML

You can export the data in a DataWindow or DataStore object to XML using similar techniques used for exporting to other formats such as PSR or HTML:

1. Using the Save Rows As menu item in the DataWindow painter when the Preview view is open.
2. Using the SaveAs() method:

```
dw_1.SaveAs( "c:\foo\bar.xml", XML!, TRUE )
```

3. Using PowerScript dot notation:

```
ls_xml = dw_1.object.datawindow.data.xml
```

4. Using PowerScript Describe notation:

```
ls_xml = dw_1.Describe( "datawindow.data.xml" )
```

New DataWindow properties are used to fine tune the generation of XML from DataWindow data. These properties can be set at both runtime and design-time. The following is a list of those properties and their descriptions.

**Export.XML.UseTemplate** is used to specify a named template object for use in a given export operation. This property can be selected at design-time via a drop-down list box containing all saved export templates, or at runtime using the following syntax:

1. Using PowerScript dot notation:

```
dw_1.Object.DataWindow.Export.XML.UseTemplate = "value"
```

2. Using PowerScript Modify notation:

```
dw_1.Modify( "DataWindow.Export.XML.UseTemplate {= 'value'}" )
```

**Export.XML.MetaDataType** is used to specify what metadata will be attached to the XML generated by the DataWindow. This property can be selected at design-time via a drop-down list box containing all saved export templates, or at runtime using the following syntax:

1. Using PowerScript dot notation:

```
dw_1.Object.DataWindow.Export.XML.MetaDataType = "value"
```

2. Using PowerScript Modify notation:

```
dw_1.Modify( "DataWindow.Export.XML.MetaDataType {= 'value'}" )
```

Table 2 lists the possible values for this property.

**Export.XML.SaveMetaData** is used to specify how the generated metadata will be stored. This property can be selected at design-time via a drop-down list box containing all saved export templates, or at runtime using the following syntax:

1. Using PowerScript dot notation:

```
dw_1.Object.DataWindow.Export.XML.SaveMetaData = "value"
```

2. Using PowerScript Modify notation:

```
dw_1.Modify( "DataWindow.Export.XML.SaveMetaData {= 'value'}" )
```

**Export.XML.HeadGroups** is used to specify how XML header data will be repeated for detail rows. This property can be selected at design-time via a checkbox, combined with the ability to apply a conditional expression, or at runtime using the following syntax:

1. Using PowerScript dot notation:

```
dw_1.Object.DataWindow.Export.XML.HeadGroups = "value"
```

2. Using PowerScript Modify notation:

```
dw_1.Modify( "DataWindow.Export.XML.HeadGroups {= 'value'}" )
```

**Export.XML.IncludeWhiteSpace** is used to specify whether white space should be preserved in the generated XML. This property can be selected at design-time via a checkbox, combined with the ability to apply a conditional expression, or at runtime using the following syntax:

1. Using PowerScript dot notation:

```
dw_1.Object.DataWindow.Export.XML.IncludeWhiteSpace = "value"
```

2. Using PowerScript Modify notation:

```
dw_1.Modify( "DataWindow.Export.XML.IncludeWhiteSpace {= 'value'}" )
```

Table 3 lists the possible values for this property. Figure 4 shows the DataWindow painter user interface for setting the XML export properties of a DataWindow.

**Table 2: Enumerated values for MetaDataType property**

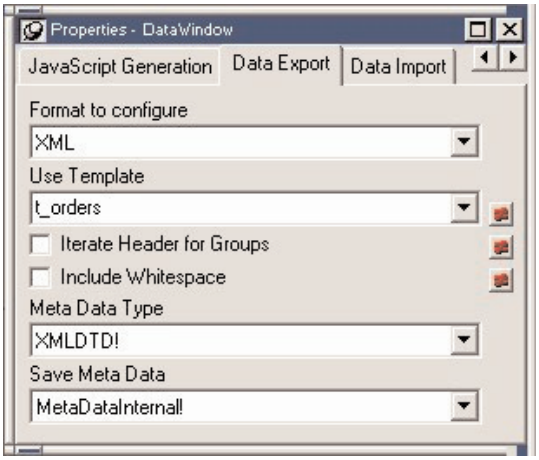
| *Enumerated Value* | *Numeric Value* | *Meaning* |
|---|---|---|
| XMLNone! | 0 | Metadata (XML Schema or DTD) is not generated along with the output XML document. |
| XMLSchema! | 1 | XML Schema will be generated along with the output XML document. |
| XMLDTD! | 2 | DTD will be generated along with the output XML document. |

**Table 3: Enumerated values for MetaDataType property**

| *Enumerated Value* | *Numeric Value* | *Meaning* |
|---|---|---|
| MetaDataInternal! | 0 | The metadata will be written into the DOCTYPE section of the output XML document. |
| MetaDataExternal! | 1 | The metadata will be saved as an external file with the same name as the XML document. The file name extension will be based on the metadata type, i.e., .xsd (for XMLSchema! type) or .dtd (for XMLDTD! type).<br><br>The output XML document will include a DOC TYPE reference to the metadata file. |

*Note: the Modify syntax can use both the numeric and enumerated values in the expression.*

**Figure 4: XML Export DataWindow properties**




### Example I

**Table 4: Settings**

| *Template* | *MetaDataType* | *SaveMetaData* | *IncludeWhiteSpace* | *HeadGroups* |
|---|---|---|---|---|
| t_default | XMLNone! | N/A | Checked | Unchecked |

Results

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<d_orders>
   <d_orders_row>
      <order_id>2001</order_id>
      <order_cust_id>101</order_cust_id>
      <order_date>3/14/1996</order_order_date>
      <order_items_line_id>1</order_items_line_id>
      <order_items_prod_id>300</order_items_prod_id>
      <order_items_quantity>12</order_items_quantity>
      <order_items_ship_date>1996-09-15</order_items_ship_date>
      <customer_fname>Michaels</customer_fname>
      <customer_lname>Devlin</customer_lname>
   </d_orders_row>
   <d_orders_row>
   ...
   </d_orders_row>
<d_orders>
```

### Example II

Table 5: Settings

| *Template* | *MetaDataType* | *SaveMetaData* | *IncludeWhiteSpace* | *HeadGroups* |
|---|---|---|---|---|
| t_default | XMLDTD! | MetaDataInternal | Checked | Unchecked |

Results

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE d_orders [
<!ELEMENT d_orders (d_orders_row*)>
<!ELEMENT d_orders_row (order_id, order_cust_id, order_date,
           order_items_line_id, order_items_prod_id,
           order_items_quantity, order_items_ship_date,
           customer_fname, customer_lname)>
<!ELEMENT order_id (#PCDATA)>
<!ELEMENT order_cust_id (#PCDATA)>
<!ELEMENT order_date (#PCDATA)>
<!ELEMENT order_items_line_id (#PCDATA)>
<!ELEMENT order_items_prod_id (#PCDATA)>
<!ELEMENT order_items_quantity (#PCDATA)>
<!ELEMENT order_items_ship_date (#PCDATA)>
<!ELEMENT customer_fname (#PCDATA)>
<!ELEMENT customer_lname (#PCDATA)>
]>
<d_orders>
   <d_orders_row>
   ...
   </d_orders_row>
</d_orders>
```

### Example III

Table 6: Settings

| *Template* | *MetaDataType* | *SaveMetaData* | *IncludeWhiteSpace* | *HeadGroups* |
|---|---|---|---|---|
| t_orders | XMLSchema! | MetaDataExternal! | Checked | Unchecked |

**Results**

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Orders>
<Orders xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="xml11E.xsd">
   <Customer id="101" fname="Michaels" lname="Devlin">
      <Order id="2001">
         <Item id="1">
            <Product>300</Product>
            <Qty>12</Qty>
            <ShipDate>1996-09-15</ShipDate>
         </Item>
         <Item id="2">
            <Product>301</Product>
            <Qty>12</Qty>
            <ShipDate>1996-09-14</ShipDate>
         </Item>
         <Item id="3">
            <Product>302</Product>
            <Qty>12</Qty>
            <ShipDate>1996-09-14</ShipDate>
         </Item>
      </Order>
   </Customer>
   <Customer id="101" fname="Michaels" lname="Devlin">
      <Order id="2005">
         <Item id="1">
            <Product>700</Product>
            <Qty>12</Qty>
            <ShipDate>1996-09-24</ShipDate>
         </Item>
      </Order>
   </Customer>
   <Customer id="102" fname="Beth" lname="Reiser">
      ...
   </Customer>
</Orders>
```

**Contents of xml11E.xsd file**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Orders">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="Customer" maxOccurs="unbounded"
minOccurs="0"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Customer">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="Order"/>
         </xs:sequence>
         <xs:attribute name="id" type="xs:string"/>
         <xs:attribute name="fname" type="xs:string"/>
         <xs:attribute name="lname" type="xs:string"/>
```

```
      </xs:complexType>
   </xs:element>
   <xs:element name="Order">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="Item" maxOccurs="unbounded"
minOccurs="0"/>
         </xs:sequence>
         <xs:attribute name="id" type="xs:string"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Item">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="Product"/>
            <xs:element ref="Qty"/>
            <xs:element ref="ShipDate"/>
         </xs:sequence>
         <xs:attribute name="id" type="xs:string"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Product" type="xs:int"/>
   <xs:element name="Qty" type="xs:int"/>
   <xs:element name="ShipDate" type="xs:date"/>
</xs:schema>
```

*Note: notice the repetition of the <Customer> element in the generated XML – we expect customer #101's (Michael Devlin) orders to be listed sequentially within the same <Customer> element, but they're not. We will fix this using the HeadGroups setting in Example 4 below.*

### Example IV

Table 7: Settings

| *Template* | *MetaDataType* | *SaveMetaData* | *IncludeWhiteSpace* | *HeadGroups* |
|---|---|---|---|---|
| t_orders | XMLSchema! | MetaDataExternal! | Checked | Checked |

Results

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE Orders>
<Orders xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:noNamespaceSchemaLocation="xml11E.xsd">
   <Customer id="101" fname="Michaels" lname="Devlin">
      <Order id="2001">
         <Item id="1">
            <Product>300</Product>
            <Qty>12</Qty>
            <ShipDate>1996-09-15</ShipDate>
         </Item>
         <Item id="2">
            <Product>301</Product>
            <Qty>12</Qty>
            <ShipDate>1996-09-14</ShipDate>
```

```
            </Item>
            <Item id="3">
               <Product>302</Product>
               <Qty>12</Qty>
               <ShipDate>1996-09-14</ShipDate>
            </Item>
         </Order>
      </Customer>
      <Customer id="101" fname="Michaels" lname="Devlin">
         <Order id="2005">
            <Item id="1">
               <Product>700</Product>
               <Qty>12</Qty>
               <ShipDate>1996-09-24</ShipDate>
            </Item>
         </Order>
      </Customer>
      <Customer id="102" fname="Beth" lname="Reiser">
         ...
      </Customer>
</Orders>
```

*Note that the <customer> node is repeating for orders 2001 and 2005; this seems extraneous. The expected output shown below displays a single <customer> node for all orders of that customer.*

```
<Customer id="101" fname="Michaels" lname="Devlin">
   <Order id="2001">
      <Item id="1">
         <Product>300</Product>
         <Qty>12</Qty>
         <ShipDate>1996-09-15</ShipDate>
      </Item>
      <Item id="2">
         <Product>301</Product>
         <Qty>12</Qty>
         <ShipDate>1996-09-14</ShipDate>
      </Item>
      <Item id="3">
         <Product>302</Product>
         <Qty>12</Qty>
         <ShipDate>1996-09-14</ShipDate>
      </Item>
   </Order>
   <Order id="2005">
      <Item id="1">
      <Product>700</Product>
         <Qty>12</Qty>
         <ShipDate>1996-09-24</ShipDate>
      </Item>
   </Order>
</Customer>
<Customer id="102" fname="Beth" lname="Reiser">
   ...
</Customer>
```

*Note: The release candidate (RC) version of beta 4 still produces the extraneous <customer> node. This issue is under investigation and is expected to be resolved in time for the GA release. Note that as an interim workaround, one could use 2 export templates; one that renders <customer> nodes and one that renders the respective <order> nodes. The 2 output can then be merged using the new XML Parser Interface (PBDOM).*

### Importing XML

You can import the content of an XML document into a DataWindow or DataStore using similar techniques used for importing other structured data formats such as comma- or tab-delimited:

1. Using the ImportFile() method:
```
dw_1.ImportFile( XML!, "c:\foo\bar.xml" {*} )
```
2. Using the ImportString() method:
```
dw_1.ImportString( XML!, ls_xml {*} )
```
3. Using the ImportClipboard() method:
```
ls_xml = dw_1.ImportClipboard()
```

{*} Optional parameters omitted for brevity.

New DataWindow properties are used to fine tune the populating of a DataWindow from an XML source. These properties can be set at both runtime and design-time. The following is a list of those properties and their descriptions.

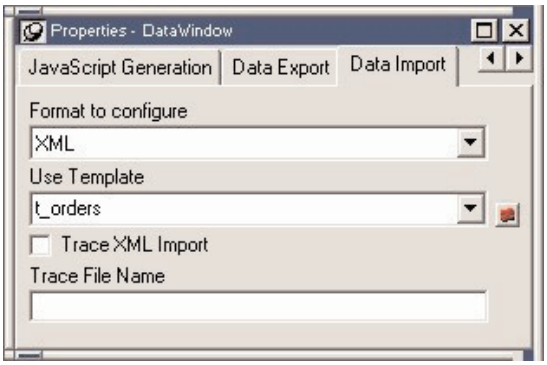
**Import.XML.UseTemplate** is used to specify a named template object for use in a given import operation. Its behavior is identical to that of the Export.XML.UseTemplate property.

**Import.XML.Trace** is used to specify whether the import operation should be traced to a file. **The Import.XML.TraceFile** property specifies the file name to use for recording the trace results.

*Note: if the XML Trace file is not specified but XML Trace option is checked then PowerBuilder generates the default Trace file named "pbxmtrc.log" in the current directory.*

Figure 5 shows the DataWindow painter user interface for setting the XML import properties of a DataWindow.

**Figure 5: XML Import DataWindow properties**



## 3. Conclusion

The PowerBuilder DataWindow has always excelled in the tasks of retrieving, presenting and manipulating relational data. With the addition of XML export and import capabilities in version 9.0, PowerBuilder client applications as well as server-side components gain new and robust features that enable them to better communicate with other applications and components, and the DataWindow is once again poised to handle the challenges of today's and tomorrow's technologies

– Roy Kiesler
*TeamSybase*
***roy.kiesler@teamsybase.com***

**International Contacts**

**Argentina**
+5411 4313 4488

**Australia**
+612 9936 8800

**Austria**
+43 1 504 8510

**Belgium**
+32 2 713 15 03

**Brazil**
+5511 3046 7388

**Bulgaria**
+359 2 986 1287

**Canada**
+905 273 8500

**Central America**
+506 204 7151

**Chile**
+56 2 330 6700

**China**
+8610 6856 8488

**Colombia**
+57 1 218 8266

**Croatia**
+385 42 33 1812

**Czech Republic**
+420 2 24 31 08 08

**Denmark**
+45 3927 7913

**Ecuador**
+59 322 508 593

**El Salvador**
+503 245 1128

**Finland**
+358 9 7250 200

**France**
+33 1 41 91 96 80

**Germany**
+49 69 9508 6182

**Greece**
+30 1 98 89 300

**Guatemala**
+502 366 4348

**Honduras**
+504 239 5483

**Hong Kong**
+852 2506 6000

**Hungary**
+36 1 248 2919

**India**
+91 22 655 0258

**Indonesia**
+62 21 526 7690

**Israel**
+972 3 548 3555

**Italy**
+39 02 696 820 64

**Ivory Coast**
+225 22 43 73 73

**Japan**
+81 3 5210 6000

**Kazakstan**
+7 3272 64 1566

**Korea**
+82 2 3451 5200

**Malaysia**
+603 2142 4218

**Mexico**
+52 5282 8000

**Netherlands**
+31 20 346 9290

**New Zealand**
+64 4473 3661

**Nigeria**
+234 12 62 5120

**Norway**
+47 231 621 45

**Panama**
+507 263 4349

**Peru**
+511 221 4190

**Philippines**
+632 750 2550

**Poland**
+48 22 844 55 55

**Portugal**
+351 21 424 6710

**Puerto Rico**
+787 289 7895

**Romania**
+40 1 231 08 70

**Russian Federation**
+7 095 797 4774

**Slovak Republic**
+421 26 478 2281

**Slovenia**
+385 42 33 1812

**South Africa**
+27 11 804 3740

**South Korea**
+82 2 3451 5200

**Spain**
+34 91 749 7605

**Sweden**
+46 8 568 512 00

**Switzerland**
+41 1 800 9220

**Taiwan**
+886 2 2715 6000

**Thailand**
+662 618 8638

**Turkey**
+90 212 325 4114

**Ukraine**
+380 44 227 3230

**United Arab Emirates**
+971 2 627 5911

**United Kingdom**
+44 870 240 2255

**Venezuela**
+58 212 267 5670

**Asian Solutions Center**
+852 2506 8700

**For other Europe, Middle East, or Africa inquiries:**
+33 1 41 90 41 64 (Sybase Europe)

**For other Asia Pacific inquiries:**
+852 2506 8700 (Hong Kong)

**For other Latin America inquiries:**
+925 236 6820

**Sybase, Inc.**
**Worldwide Headquarters**
One Sybase Drive
Dublin, CA 94568-7902 USA
Tel: +800 8 SYBASE
www.sybase.com

Portions of this whitepaper are excerpted from the upcoming book "Distributed Applications Development with PowerBuilder 9", SAMS Publication book. For more information about the book, please see http://www.pb9books.com.