Informatica® PowerCenter
10.2

XML Guide

Informatica PowerCenter XML Guide
10.2
September 2017

Publication Date: 2022-04-28

Table of Contents

Preface

The *XML Guide* is written for developers and software engineers responsible for working with XML in a data warehouse environment. Before you use the *XML Guide*, ensure that you have a solid understanding of XML concepts, your operating systems, flat files, or mainframe system in your environment. Also, ensure that you are familiar with the interface requirements for your supporting applications.

Informatica Resources

Informatica Network

Informatica Network hosts Informatica Global Customer Support, the Informatica Knowledge Base, and other product resources. To access Informatica Network, visit https://network.informatica.com.

As a member, you can:

- Access all of your Informatica resources in one place.
- Search the Knowledge Base for product resources, including documentation, FAQs, and best practices.
- View product availability information.
- Review your support cases.
- Find your local Informatica User Group Network and collaborate with your peers.

Informatica Knowledge Base

Use the Informatica Knowledge Base to search Informatica Network for product resources such as documentation, how-to articles, best practices, and PAMs.

To access the Knowledge Base, visit https://kb.informatica.com. If you have questions, comments, or ideas about the Knowledge Base, contact the Informatica Knowledge Base team at KB_Feedback@informatica.com.

Informatica Documentation

To get the latest documentation for your product, browse the Informatica Knowledge Base at https://kb.informatica.com/_layouts/ProductDocumentation/Page/ProductDocumentSearch.aspx.

If you have questions, comments, or ideas about this documentation, contact the Informatica Documentation team through email at infa_documentation@informatica.com.

Informatica Product Availability Matrixes

Product Availability Matrixes (PAMs) indicate the versions of operating systems, databases, and other types of data sources and targets that a product release supports. If you are an Informatica Network member, you can access PAMs at https://network.informatica.com/community/informatica-network/product-availability-matrices.

Informatica Velocity

Informatica Velocity is a collection of tips and best practices developed by Informatica Professional Services. Developed from the real-world experience of hundreds of data management projects, Informatica Velocity represents the collective knowledge of our consultants who have worked with organizations from around the world to plan, develop, deploy, and maintain successful data management solutions.

If you are an Informatica Network member, you can access Informatica Velocity resources at http://velocity.informatica.com.

If you have questions, comments, or ideas about Informatica Velocity, contact Informatica Professional Services at ips@informatica.com.

Informatica Marketplace

The Informatica Marketplace is a forum where you can find solutions that augment, extend, or enhance your Informatica implementations. By leveraging any of the hundreds of solutions from Informatica developers and partners, you can improve your productivity and speed up time to implementation on your projects. You can access Informatica Marketplace at https://marketplace.informatica.com.

Informatica Global Customer Support

You can contact a Global Support Center by telephone or through Online Support on Informatica Network.

To find your local Informatica Global Customer Support telephone number, visit the Informatica website at the following link:
http://www.informatica.com/us/services-and-training/support-services/global-support-centers.

If you are an Informatica Network member, you can use Online Support at http://network.informatica.com.

CHAPTER 1

XML Concepts

This chapter includes the following topics:

XML Concepts Overview

Extensible Markup Language (XML) is a flexible way to create common information formats and to share the formats and data between applications and on the internet.

You can import XML definitions into PowerCenter® from the following file types:

- **XML file**. An XML file contains data and metadata. An XML file can reference a Document Type Definition file (DTD) or an XML schema definition (XSD) for validation.
- **DTD file**. A DTD file defines the element types, attributes, and entities in an XML file. A DTD file provides some constraints on the XML file structure but a DTD file does not contain any data.
- **XML schema.** An XML schema defines elements, attributes, and type definitions. Schemas contain simple and complex types. A simple type is an XML element or attribute that contains text. A complex type is an XML element that contains other elements and attributes.

  Schemas support element, attribute, and substitution groups that you can reference throughout a schema. Use substitution groups to substitute one element with another in an XML instance document. Schemas also support inheritance for elements, complex types, and element and attribute groups.

XML Files

XML files contain tags that identify data in the XML file, but not the format of the data. The basic component of an XML file is an element. An XML element includes an element start tag, element content, and element end tag. All XML files must have a root element defined by a single tag at the top and bottom of the file. The root element encloses all the other elements in the file.

An XML file models a hierarchical database. The position of an element in an XML hierarchy represents its relationships to other elements. An element can contain child elements, and elements can inherit characteristics from other elements.

For example, the following XML file describes a book:

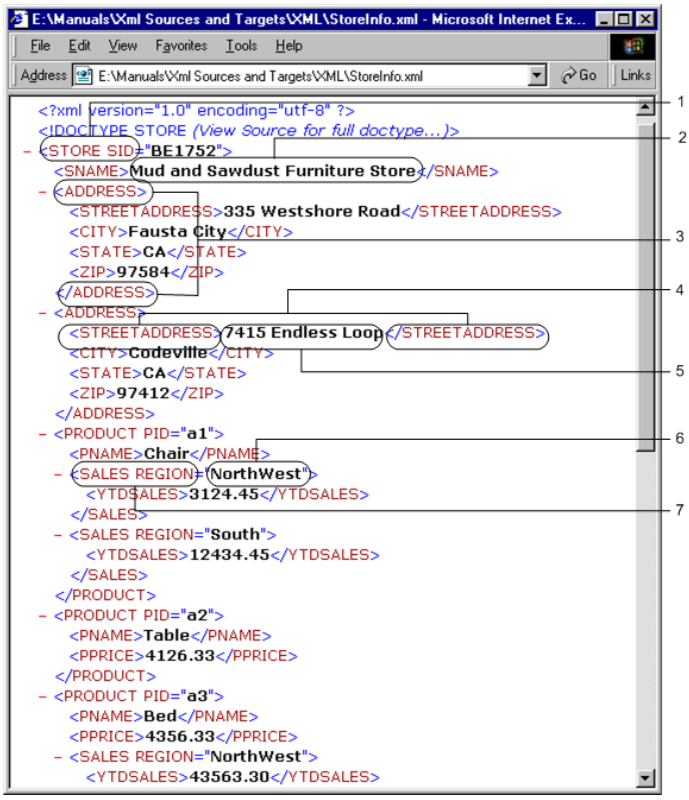
```
<book>
   <title>Fun with XML</title>
   <chapter>
      <heading>Understanding XML</heading>
      <heading>Using XML</heading>
   </chapter>
   <chapter>
      <heading>Using DTD Files</heading>
      <heading>Fun with Schemas</heading>
   </chapter>
</book>
```

Book is the root element and it contains the title and chapter elements. Book is the parent element of title and chapter, and chapter is the parent of heading. Title and chapter are sibling elements because they have the same parent.

An element can have attributes that provide additional information about the element. In the following example, the attribute graphic_type describes the content of picture:

```
<picture graphic_type="gif">computer.gif</picture>
```

The following figure shows the structure, elements, and attributes in an XML file:



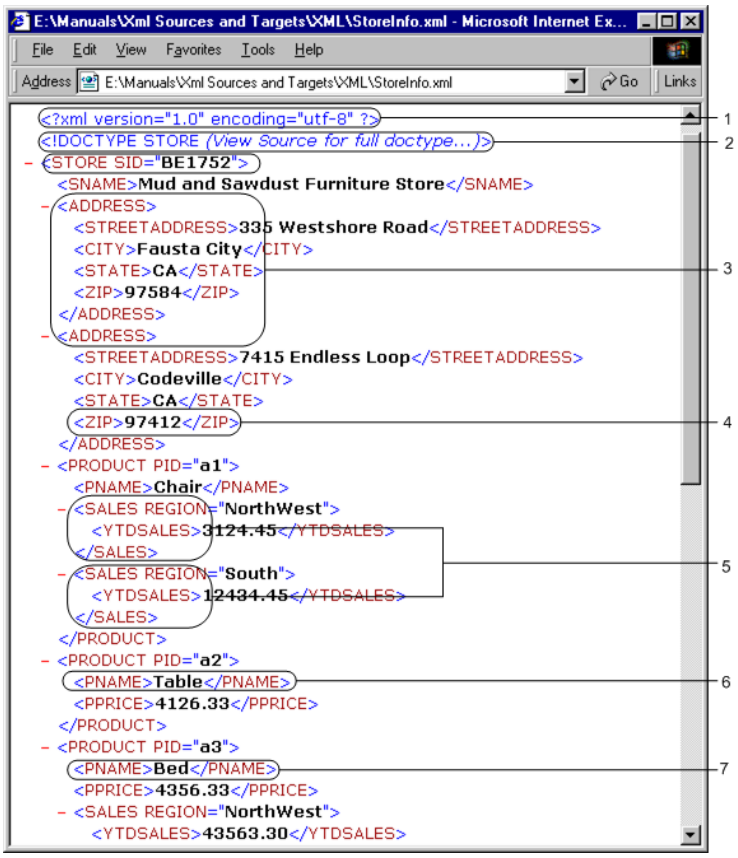
1. Root element.
2. Element data.
3. Enclosure element.
4. Element tags.
5. Element data.
6. Attribute value.
7. Attribute tag.

An XML file has a hierarchical structure. An XML hierarchy includes the following elements:

- **Child element.** An element contained within another element.
- **Enclosure element.** An element that contains other elements but does not contain data. An enclosure element can include other enclosure elements.
- **Global element.** An element that is a direct child of the root element. You can reference global elements throughout an XML schema.
- **Leaf element.** An element that does not contain other elements. A leaf element is the lowest level element in the XML hierarchy.
- **Local element.** An element that is nested in another element. You can reference local elements only within the context of the parent element.
- **Multiple-occurring element.** An element that occurs more than once within its parent element. Enclosure elements can be multiple-occurring elements.
- **Parent chain.** The succession of child-parent elements that traces the path from an element to the root.
- **Parent element.** An element that contains other elements.

- **Single-occurring element.** An element that occurs once within its parent.

The following figure shows some elements in an XML hierarchy:



1. The encoding attribute identifies the code page.
2. The DOCTYPE identifies an associated DTD file.
3. Enclosure Element: Element Address encloses elements StreetAddress, City, State, and Zip. Element Address is also a Parent element.
4. Leaf Element: Element Zip, along with all its sibling elements, is the lowest level element within element Address.
5. Multiple-occurring Element: Element Sales Region occurs more than once within element Product.
6. Single-occurring Element: Element PName occurs once within element Product.
7. Child Element: Element PName is a child of Product, which is a child of Store.

Validating XML Files with a DTD or Schema

A valid XML file conforms to the structure of an associated DTD or schema file.

To reference the location and name of a DTD file, use the DOCTYPE declaration in an XML file. The DOCTYPE declaration also names the root element for the XML file.

For example, the following XML file references the location of the note.dtd file:

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM
"http://www.w3schools.com/dtd/note.dtd">
<note>
    <body>XML Data</body>
</note>
```

To reference a schema, use the schemaLocation declaration. The schemaLocation contains the location and name of a schema.

The following XML file references the note.xsd schema in an external location:

```
<?xml version="1.0"?>
<note xsi:SchemaLocation="http://www.w3schools.com note.xsd">
   <body>XML Data</body>
</note>
```

Unicode Encoding

An XML file contains an encoding attribute that indicates the code page in the file. The most common encodings are UTF-8 and UTF-16. UTF-8 represents a character with one to four bytes, depending on the Unicode symbol. UTF-16 represents a character as a 16-bit word.

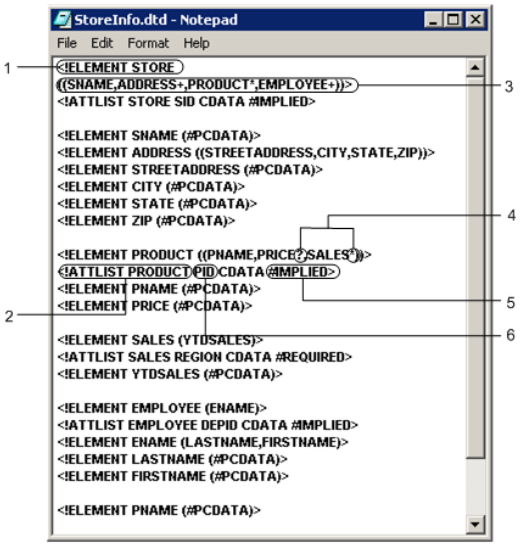
The following example shows a UTF-8 attribute in an XML file:

```
<?xml version="1.0"encoding="UTF-8"?>
<note xsi:SchemaLocation="http://www.w3schools.com note.xsd">
   <body>XML Data</body>
</note>
```

DTD Files

A Document Type Definition (DTD) file defines the element types and attributes in an XML file. A DTD file also provides some constraints on the XML file structure. A DTD file does not contain any data or element datatypes.

The following figure shows elements and attributes in a DTD file:



1. Element
2. Attribute
3. Element list
4. Element occurrence
5. Attribute value option
6. Attribute name

DTD Elements

In the DTD file, an element declaration defines an XML element. An element declaration has the following syntax:

```
<!ELEMENT product (#PCDATA)>
```

The DTD description defines the XML tag <product>. The description (#PCDATA) specifies parsed character data. Parsed data is the text between the start tag and the end tag of an XML element. Parsed character data is text without child elements.

The following example shows a DTD description of an element with two child elements:

```
<!ELEMENT boat (brand, type) >
<!ELEMENT brand (#PCDATA) >
<!ELEMENT type (#PCDATA) >
```

Brand and type are child elements of boat. Each child element can contain characters. In this example, brand and type can occur once inside the element boat. The following DTD description specifies that brand must occur one or more times for a boat:

```
<!ELEMENT boat (brand+) >
```

DTD Attributes

Attributes provide additional information about elements. In a DTD file, an attribute occurs inside the starting tag of an element.

The following syntax describes an attribute in a DTD file:

```
<!ATTLIST element_name attribute_name attribute_type "default_value">
```

The following parameters identify an attribute in a DTD file:

- **Element_name**. The name of the element that has the attribute.
- **Attribute_name.** The name of the attribute.
- **Attribute_type.** The kind of attribute. The most common attribute type is CDATA. A CDATA attribute is character data.
- **Default_value.** The value of the attribute if no attribute value occurs in the XML file.

  Use the following options with a default value:

  - **#REQUIRED.** The XML file must contain the attribute value.
  - **#IMPLIED.** The attribute value is optional.
  - **#FIXED.** The XML file must contain the default value from the DTD file. A valid XML file can contain the same attribute value as the DTD, or the XML file can have no attribute value. You must specify a default value with this option.

    The following example shows an attribute with a fixed value:

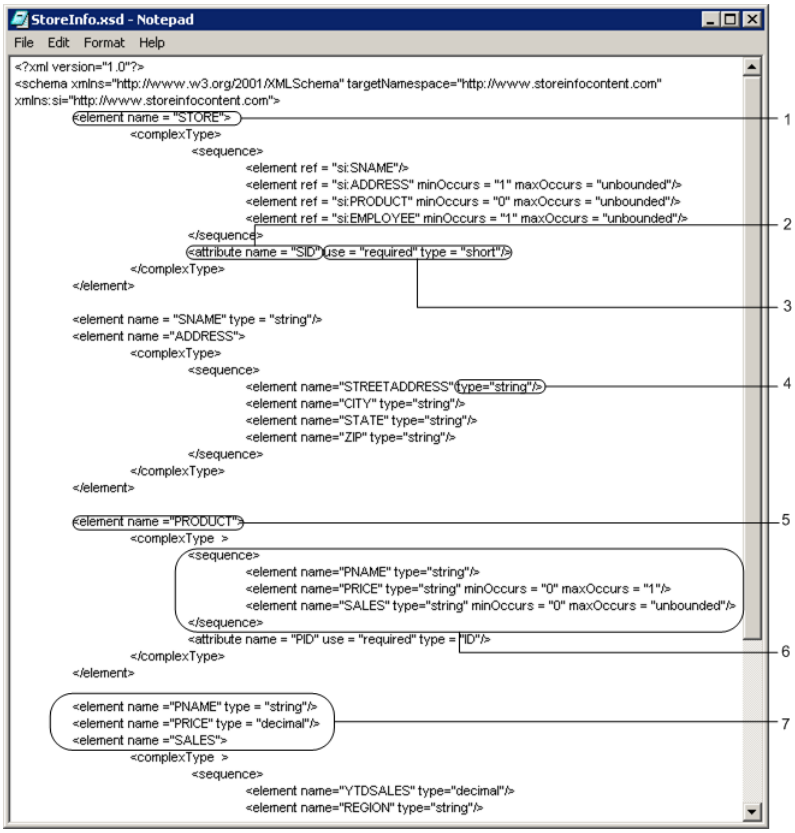
    ```
    <!ATTLIST product product_name CDATA #FIXED "vacuum">
    ```

    The element name is product. The attribute is product_name. The attribute has a default value, vacuum.

XML Schema Files

An XML schema is a document that defines the valid content of XML files. An XML schema file, like a DTD file, contains only metadata. An XML schema defines the structure and type of elements and attributes for an associated XML file. When you use a schema to define an XML file, you can restrict data, define data formats, and convert data between datatypes. An XML schema supports complex types and inheritance between types. A schema provides a way to specify element and attribute groups, ANY content, and circular references.

The following figure shows XML schema components:



1. Element name.
2. Attribute
3. Attribute type and null construction
4. Element datatype
5. Element data
6. Element list and occurrence
7. Element list and datatype

Related Topics:

- "Simple and Complex XML Types" on page 22
- "Component Groups" on page 28

Types of XML Metadata

You can create PowerCenter XML definitions from XML, DTD, or XML schema files. XML files provide data and metadata. DTD files and XML schema files provide metadata.

PowerCenter extracts the following types of metadata from XML, DTD, and XML schema files:

- **Namespace.** A collection of elements and attribute names identified by a Uniform Resource Identifier (URI) reference in an XML file. Namespace differentiates between elements that come from different sources.
- **Name.** A tag that contains the name of an element or attribute.
- **Hierarchy.** The position of an element in relationship to other elements in an XML file.
- **Cardinality.** The number of times an element occurs in an XML file.
- **Datatype.** A classification of a data element, such as numeric, string, Boolean, or time. XML supports custom datatypes and inheritance.

Namespace

A namespace contains a URI to identify schema location. A URI is a string of characters that identifies an internet resource. A URI is an abstraction of a URL. A URL locates a resource, but a URI identifies a resource. A DTD or schema file does not have to exist at the URI location.

An XML namespace identifies groups of elements. A namespace can identify elements and attributes from different XML files or distinguish meanings between elements. For example, you can distinguish meanings for the element "table" by declaring different namespaces, such as *math:table* and *furniture:table*. XML is case sensitive. The namespace *Math:table* is different from the namespace *math:table*.

You can declare a namespace at the root level of an XML file, or you can declare a namespace inside any element in an XML structure. When you declare multiple namespaces in the same XML file, you use a namespace prefix to associate an element with a namespace. A namespace declaration appears in the XML file as an attribute that starts with xmlns. Declare the namespace prefix with the xmlns attribute. You can create a prefix name of any length.

The following example shows two namespaces in an XML instance document:

```
<example>
  xmlns:math = "http://www.mathtables.com"
  xmlns:furniture = "http://www.home.com">
  <math:table>4X6</math:table>
  <furniture:table>Brueners </furniture:table>
</example>
```

One namespace has math elements, and the other namespace has furniture elements. Each namespace has an element called "table," but the elements contain different types of data. The namespace prefix distinguishes between the math table and the furniture table.

The following text shows a common schema declaration:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.w3XML.com"
   xmlns="http://www.w3XML.com"
   elementFormDefault="qualified">...
...</xs:schema>
```

The following table describes each part of the namespace declaration:

Schema Declaration	Description
xmlns:xs="http://www.w3.org/2001/XMLSchema"	Namespace that contains the native XML schema and datatypes. In this example, each schema component has the prefix of "xs."
targetNamespace="http://www.w3XML.com"	Namespace that contains the schema.

Schema Declaration	Description
xmlns="http://www.w3XML.com"	Default namespace declaration. All elements in the schema that have no prefix belong to the default namespace. Declare a default namespace by using an xmlns attribute with no prefix.
elementFormDefault="qualified"	Specifies that any element in the schema must have a namespace in the XML file.

Name

In an XML file, each tag is the name of an element or attribute. In a DTD file, the tag <!ELEMENT> specifies the name of an element, and the tag <!ATTLIST> indicates the set of attributes for an element. In a schema file, <element name> specifies the name of an element and <attribute name> specifies the name of an attribute.

When you import an XML definition, the element tags become column names in the PowerCenter definition, by default.

Hierarchy

An XML file models a hierarchical database. The position of an element in an XML hierarchy represents its relationship to other elements. For example, an element can contain child elements, and elements can inherit characteristics from other elements.

Cardinality

Element cardinality in a DTD or schema file is the number of times an element occurs in an XML file. Element cardinality affects how you structure groups in an XML definition. Absolute cardinality and relative cardinality of elements affect the structure of an XML definition.

Absolute Cardinality

The absolute cardinality of an element is the number of times an element occurs within its parent element in an XML hierarchy. DTD and XML schema files describe the absolute cardinality of elements within the hierarchy. A DTD file uses symbols, and an XML schema file uses the <minOccurs> and <maxOccurs> attributes to describe the absolute cardinality of an element.

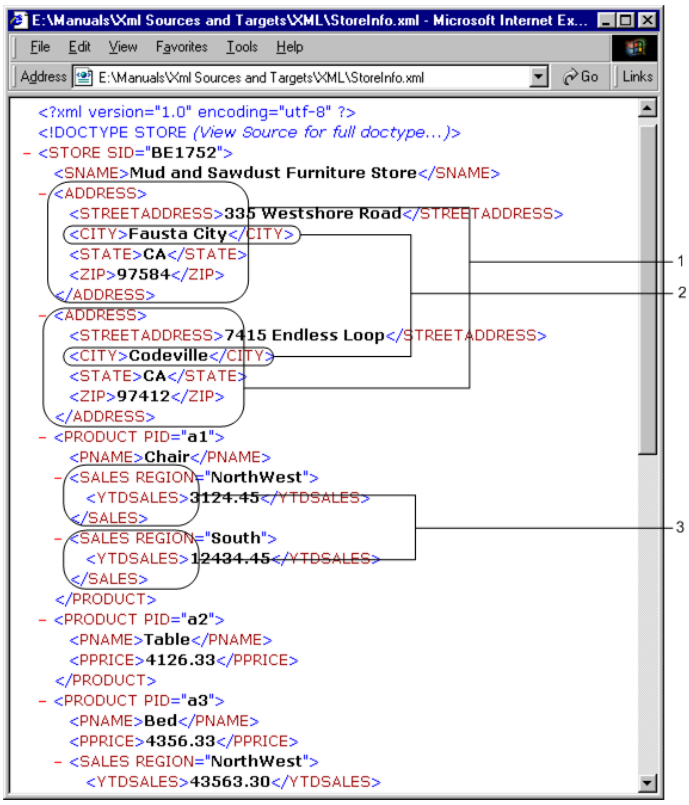
For example, an element has an absolute cardinality of once (1) if the element occurs once within its parent element. However, the element might occur many times within an XML hierarchy if the parent element has a cardinality of one or more (+).

The absolute cardinality of an element determines its null constraint. An element that has an absolute cardinality of one or more (+) cannot have null values, but an element with a cardinality of zero or more (*) can have null values. An attribute marked as fixed or required in an XML schema or DTD file cannot have null values, but an implied attribute can have null values.

The following table describes how DTD and XML schema files represent cardinality:

Absolute Cardinality	DTD	Schema
Zero or once	?	minOccurs=0 maxOccurs=1
Zero or one or more times	*	minOccurs=0 maxOccurs=unbounded minOccurs=0 maxOccurs=n
Once	-	minOccurs=1 maxOccurs=1
One or more times	+	minOccurs=1 maxOccurs=unbounded minOccurs=1 maxOccurs=n
Note: You can declare a maximum number of occurrences or an unlimited occurrences in a schema.		

The following figure shows the absolute cardinality of elements in a sample XML file:



1. Element *Address* occurs more than once within *Store*. Its absolute cardinality is one or more(+).
2. Element City occurs once within its parent element Address. Its absolute cardinality is once(1).
3. Element Sales occurs zero or more times within its parent element Product. Its absolute cardinality is zero or more(*).

Relative Cardinality

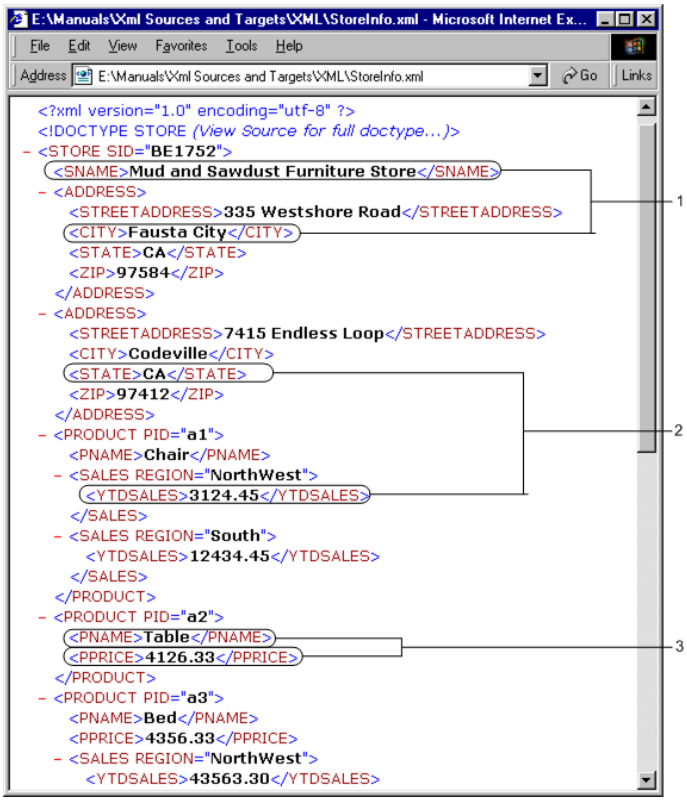
Relative cardinality is the relationship of an element to another element in the XML hierarchy. An element can have a one-to-one, one-to-many, or many-to-many relationship to another element in the hierarchy.

An element has a one-to-one relationship with another element if every occurrence of one element can have one occurrence of the other element. For example, an employee element can have one social security number element. Employee and social security number have a one-to-one relationship.

An element has a one-to-many relationship with another element if every occurrence of one element can have multiple occurrences of another element. For example, an employee element can have multiple email addresses. Employee and email address have a one-to-many relationship.

An element has a many-to-many relationship with another element if an XML file can have multiple occurrences of both elements. For example, an employee might have multiple email addresses and multiple street addresses. Email address and street address have a many-to-many relationship.

The following figure shows the relative cardinality between elements in a sample XML file:



1. One-to-many relationship. For every occurrence of SNAME, there can be many occurrences of ADDRESS and, therefore, many occurrences of CITY.
2. Many-to-many relationship. For every occurrence of STATE, there can be multiple occurrences of YTDSALES. For every occurrence of YTDSALES, there can be many occurrences of STATE.
3. One-to-one relationship. For every occurrence of PNAME, there is one occurrence of PPRICE.

Simple and Complex XML Types

The XML schema language has over 40 built-in datatypes, including numeric, string, time, XML, and binary. These datatypes are called simple types. They contain text but no other elements and attributes. You can derive new simple types from the basic XML simple types.

You can create complex XML datatypes. A complex datatype is a datatype that contains more than one simple type. A complex datatype can also contain other complex types and attributes.

For more information about XML datatypes, see the W3C specifications for XML datatypes at http://www.w3.org/TR/xmlschema-2.

Simple Types

A simple datatype is an XML element or attribute that contains text. A simple type is indivisible. Simple types cannot have attributes, but attributes are simple types.

PowerCenter supports the following simple types:

- **Atomic types.** A basic datatype such as Boolean, string, or integer.
- **Lists.** An array collection of atomic types.
- **Unions.** A combination of one or more atomic or list types that map to a simple type in an XML file.

Atomic Types

An atomic datatype is a basic datatype such as a Boolean, string, integer, decimal, or date. To define custom atomic datatypes, add restrictions to an atomic datatype to limit the content. Use a facet to define which values to restrict or allow.

A facet is an expression that defines minimum or maximum values, specific values, or a data pattern of valid values. For example, a pattern facet restricts an element to an expression of data values. An enumeration facet lists the legal values for an element.

The following example contains a pattern facet that restricts an element to a lowercase letter between a and z:

```
<xs:element name="letter">
   <xs:simpleType>
   <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
   </xs:restriction>
</xs:simpleType></xs:element>
```

The following example contains an enumeration facet that restricts a string to a, b, or c:

```
<xs:element name="letter">
   <xs:simpleType>
   <xs:restriction base="xs:string">
      <xs:enumeration value="a"/>
      <xs:enumeration value=”b”/>
      <xs:enumeration value=”c”/>
   </xs:restriction>
</xs:simpleType></xs:element>
```

Lists

A list is an array collection of atomic types, such as a list of strings that represent names. The list itemType defines the datatype of the list components.

The following example shows a list called names:

```
<xs:simpleType name="names">
   <xs:list itemType="xs:string" />
</xs:simpleType>
```

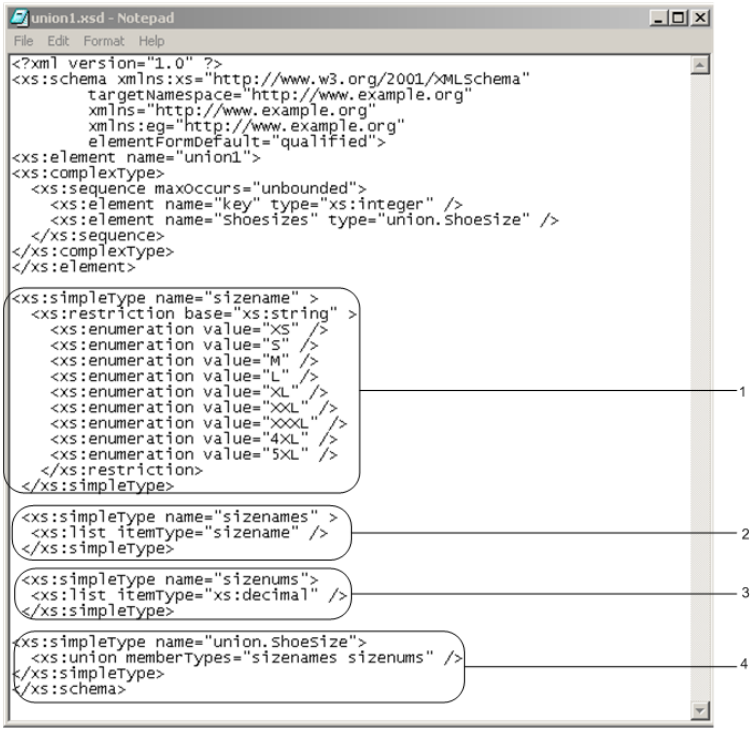
An XML file might contain the following data in the names list:

```
<names>Joe Bob Harry Atlee Will</names>
```

Unions

A union is a combination of one or more atomic or list types that map to one simple type in an XML file. When you define a union type, you specify what types to combine. For example, you might create a type called size. Size can include string data, such as S, M, and L, or size might contain decimal sizes, such as 30, 32, and 34. If you define a union type element, the XML file can include a sizename type for string sizes, and a sizenum type for numeric sizes.

The following figure shows a schema file containing a shoesize union that contains sizenames and sizenums lists:



1. Sizename is a restricted string type.
2. The sizenames type accepts a list of strings.
3. The sizenums type accepts a list decimals.
4. The shoesize union accepts both the decimal and string lists.

The union defines sizenames and sizenums as union member types. Sizenames defines a list of string values. Sizenums defines a list of decimal values.

Complex Types

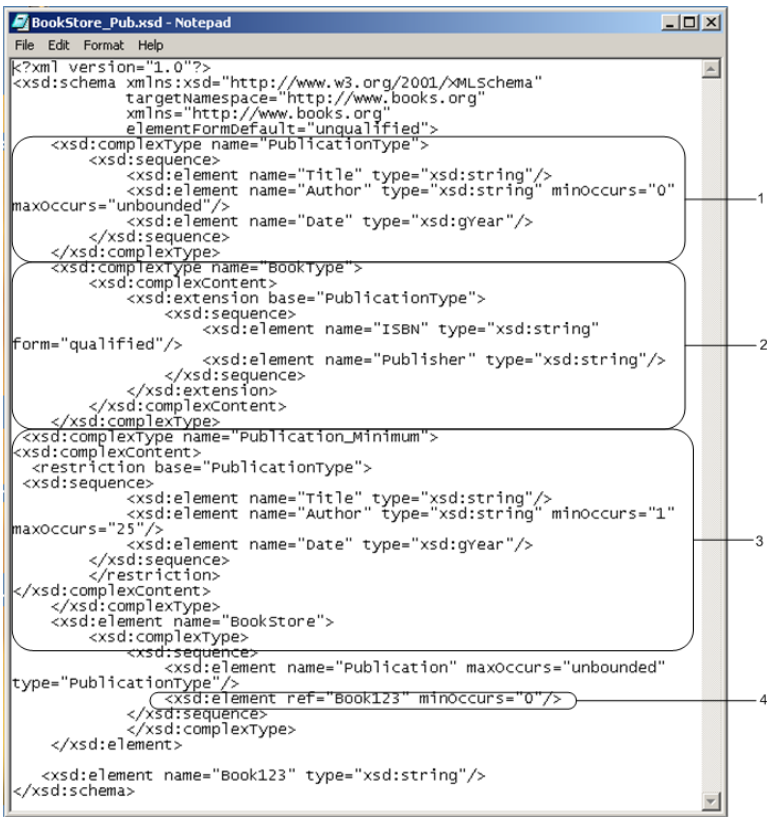
A complex type aggregates a collection of simple types into a logical unit. For example, a customer type might include the customer number, name, street address, town, city, and zip code. A complex type can also reference other complex types or element and attribute groups.

XML supports complex type inheritance. When you define a complex type, you can create other complex types that inherit the components of the base type. In a type relationship, the base type is the complex type from which you derive another type. A derived complex type inherits elements from the base type.

An extended complex type is a derived type that inherits elements from a base type and includes additional elements. For example, a customer_purchases type might inherit its definition from the customer complex type, but the customer_purchases type adds item, cost, and date_sold elements.

A restricted complex type is a derived type that restricts some elements from the base type. For example, mail_list might inherit elements from customer, but restrict the phone_number element by setting the minoccurs and maxoccurs boundaries to zero.

The following figure shows derived complex types that restrict and extend the base complex type:



1. Base complex type
2. Extended complex type
3. Restricted complex type
4. Element reference

In the above figure, the base type is PublicationType. BookType extends PublicationType and includes the ISBN and Publisher elements. Publication_Minimum restricts PublicationType. Publication_Minimum requires between 1 and 25 Authors and restricts the date to the year.

Abstract Elements

Sometimes a schema contains a base type that defines the basic structure of a complex element but does not contain all the components. Derived complex types extend the base type with more components. Since the base type is not a complete definition, you might not want to use the base type in an XML file. You can declare the base type element to be abstract. An abstract element is not valid in an XML file. Only the derived elements are valid.

To define an abstract element, add an abstract attribute with the value “true.” The default is false.

For example, PublicationType is an abstract element. BookType inherits the elements in PublicationType, but also includes ISBN and Publisher elements. Since PublicationType is abstract, a PublicationType element is not valid in the XML file. An XML file can contain the derived type, BookType.

The following schema contains the PublicationType and BookType:

```
<xsd:complexType name="PublicationType" abstract="true">
   <xsd:sequence>
     <xsd:element name="Title" type="xsd:string"/>
     <xsd:element name="Author" type="xsd:string" minOccurs="0" maxOccurs="unbounded"/>
     <xsd:element name="Date" type="xsd:gYear"/>
   </xsd:sequence>
   </xsd:complexType>
<xsd:complexType name="BookType">
   <xsd:complexContent>
     <xsd:extension base="PublicationType" >
       <xsd:sequence>
          <xsd:element name="ISBN" type="xsd:string"/>
           <xsd:element name="Publisher" type="xsd:string"/>
        </xsd:sequence>
     </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
```

Any Type Elements and Attributes

Some schema elements and attributes allow any type of data in an XML file. Use these elements and attributes when you need to validate an XML file that has unidentified element and attribute types.

Use the following element and attributes that allow any type of data:

- **anyType element.** Allows an element to be any datatype in the associated XML file.
- **anySimpleType element.** Allows an element to be any simpleType in the associated XML file.
- **ANY content element.** Allows an element to be any element already defined in the schema.
- **anyAttribute attribute.** Allows an element to be any attribute already defined in the schema.

anyType Elements

An anyType element can be any datatype in an XML instance document. Declare an element to be anyType when the element contains different types of data.

The following schema describes a person with a first name, last name, and an age element that is anyType:

```
<xs:element name="person">
   <xs:complexType>
   <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <<xs:element name="age" type="xs:anyType"/>
   </xs:sequence>
   </xs:complexType>
</xs:element>
```

The following XML instance document includes a date type and a number in the age element:

```
<person>
   <firstname>Danny</firstname>
   <lastname>Russell</lastname>
   <age>1959-03-03</age>
</person>
<person>
```

```
   <firstname>Carla</firstname>
   <lastname>Havers</lastname>
   <age>46</age>
</person>
```

Both types are valid for the schema. If you do not declare a datatype for an element in a schema, the element defaults to anyType when you import the schema in the Designer.

anySimpleType Elements

An anySimpleType element can contain any atomic type. An atomic type is a basic datatype such as a Boolean, string, integer, decimal, or date.

The following schema describes a person with a first name, last name, and other element that is anySimpleType:

```
<xs:element name="person">
   <xs:complexType>
   <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:element name="other" type="xs:anySimpleType"/>
   </xs:sequence>
   </xs:complexType>
</xs:element>
```

The following XML instance document substitutes the anySimpleType element with a string datatype:

```
<person>
   <firstname>Kathy</firstname>
   <lastname>Russell</lastname>
   <other>Cissy</other>
</person>
```

The following XML instance document substitutes the anySimpleType element with a numeric datatype:

```
<person>
   <firstname>Kathy</firstname>
   <lastname>Russell</lastname>
   <other>34</other>
</person>
```

ANY Content Elements

The ANY content element accepts any content in an XML file. When you declare an ANY content element in a schema, you can substitute it for an element of any name and type in an XML instance document. The substitute element must exist in the schema.

When you specify ANY content, you use the keyword ANY instead of an element name and element type.

The following schema describes a person with a first name, last name, and an element that is ANY content:

```
<xs:element name="person">
   <xs:complexType>
   <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
   </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:element name="son" type="xs:string"/>
<xs:element name=”daughter” type="xs:string"/>
```

The schema includes a son element and a daughter element. You can substitute the ANY element for the son or daughter element in the XML instance document:

```
<person>
   <firstname>Danny</firstname>
   <lastname>Russell</lastname>
   <son>Atlee</son>
</person>
<person>
   <firstname>Christine</firstname>
   <lastname>Slade</lastname>
   <daughter>Susie</daughter>
</person>
```

AnyAttribute Attributes

The anyAttribute attribute accepts any attribute in an XML file. When you declare an attribute as anyAttribute you can substitute the anyAttribute element for any attribute in the schema.

The following schema describes a person with a first name, last name, and an attribute that is anyAttribute:

```
<xs:element name="person">
<xs:complexType>
   <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
   </xs:sequence>
   <xs:anyAttribute/>
</xs:complexType>
</xs:element>
```

The following schema includes a gender attribute:

```
<xs:attribute name="gender">
   <xs:simpleType>
   <xs:restriction base="xs:string">
   <xs:pattern value="male|female"/>
   </xs:restriction>
</xs:simpleType>
</xs:attribute>
```

The following XML instance document substitutes anyAttribute with the gender attribute:

```
<person gender="female">
   <firstname>Anita</firstname>
   <lastname>Ficks</lastname>
</person>
<person gender="male">
   <firstname>Jim</firstname>
   <lastname>Geimer</lastname>
</person>
```

Component Groups

You can create the following groups of components in an XML schema:

- **Element and attribute group.** Group of elements or attributes that you can reference throughout a schema.
- **Substitution group**. Group of elements that you can substitute with other elements from the same group.

Element and Attribute Groups

You can put elements and attributes in groups that you can reference in a schema. You must declare the group of elements or attributes before you reference the group.

The following example shows the schema syntax for an element group:

```
<xs:group name="Songs">
      <xs:element name="songTitle" type="xs:string" />
      <xs:element name="artist" type="xs:string" />
      <xs:element name="publisher" type="xs:string" />
  </xs:group>
```

The following example shows the schema syntax for an attribute group:

```
<xs:attributeGroup name="Songs">
   <xs:attribute name="songTitle" type="xs:string" />
   <xs:attribute name="artist" type="xs:string" />
   <xs:attribute name="publisher" type="xs:string" />
</xs:attributeGroup>
```

The following element groups provide constraints on XML data:

- **Sequence group.** All elements in an XML file must occur in the order that the schema lists them. For example, OrderHeader requires the customerName first, then orderNumber, and then orderDate:

  ```
  <xs:group name="OrderHeader">
     <xs:sequence>
        <xs:element name="customerName" type="xs:string" />
        <xs:element name="orderNumber" type="xs:number" />
        <xs:element name="orderDate" type="xs:date" />
     </xs:sequence>
  </xs:group>
  ```

- **Choice group**. One element in the group can occur in an XML file. For example, the CustomerInfo group lists a choice of elements for the XML file:

  ```
  <xs:group name="CustomerInfo">
     <xs:choice>
        <xs:element name="customerName" type="xs:string" />
        <xs:element name="customerID" type="xs:number" />
        <xs:element name="customerNumber" type="xs:integer" />
     </xs:choice>
  </xs:group>
  ```

- **All group.** All elements must occur in the XML file or none at all. The elements can occur in any order. For example, CustomerInfo requires all or none of the three elements:

  ```
  <xs:group name="CustomerInfo">
     <xs:all>
        <xs:element name="customerName" type="xs:string" />
        <xs:element name="customerAddress" type="xs:string" />
        <xs:element name="customerPhone" type="xs:string" />
     </xs:all>
  </xs:group>
  ```

Substitution Groups

Use substitution groups to replace one element with another in an XML file. For example, if you have addresses from Canada and the United States, you can create an address type for Canada and another type for the United States. You can create a substitution group that accepts either type of address.

The following schema fragment shows an Address base type and the derived types CAN_Address and USA_Address:

```
<xs:complexType name="Address">
   <xs:sequence>
      <xs:element name="Name" type="xs:string" />
      <xs:element name="Street" type="xs:string"
```

```
        minOccurs="1" maxOccurs="3" />
      <xs:element name="City" type="xs:string" />
   </xs:sequence>
</xs:complexType>
<xs:element name="MailAddress" type="Address" />
<xs:complexType name="CAN_Address">
   <xs:complexContent>
   <xs:extension base="Address">
      <xs:sequence>
      <xs:element name="Province" type="xs:string" />
      <xs:element name="PostalCode" type="CAN_PostalCode"/>
      </xs:sequence>
   </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:complexType name="USA_Address">
   <xs:complexContent>
   <xs:extension base="Address">
      <xs:sequence>
      <xs:element name="State" type="USPS_StateCode" />
      <xs:element name="ZIP" type="USPS_ZIP"/>
      </xs:sequence>
   </xs:extension>
   </xs:complexContent>
</xs:complexType>
<xs:element name="AddrCAN" type="CAN_Address"
substitutionGroup="MailAddress"/>
<xs:element name="AddrUSA" type="USA_Address"
substitutionGroup="MailAddress"/>
```

CAN_Address includes Province and PostalCode, and USA_Address includes State and Zip. The MailAddress substitution group includes both address types.

Related Topics:

- "Using Substitution Groups in an XML Definition" on page 48

XML Path

XMLPath (XPath) is a language that describes a way to locate items in an XML file. XPath uses an addressing syntax based on the route through the hierarchy from the root to an element or attribute. An XML path can contain long schema component names.

XPath uses a slash (/) to distinguish between elements in the hierarchy. XML attributes are preceded by "@" in the XPath.

You can create a query on an element or attribute XPath to filter XML data.

Related Topics:

- "Using XPath Query Predicates" on page 51

Code Pages

XML files contain an encoding declaration that indicates the code page used in the file. The most common code pages in XML are UTF-8 and UTF-16. All XML parsers support these code pages. For information on the XML character encoding specification, see the W3C website at http://www.w3c.org.

PowerCenter supports the same set of code pages for XML files that it supports for relational databases and other flat files. PowerCenter does not support a user-defined code page.

When you create an XML source or target definition, the Designer assigns the PowerCenter Client code page to the definition. If you import an XML schema that contains a code page assignment, the XML Wizard displays the code page from the schema. However, the XML Wizard does not apply that code page to the XML definition you create in the repository.

You can not configure the code page for an XML source definition. The Integration Service converts XML source files to Unicode when it parses them.

You can configure the code page for a target XML definition in the Designer. You can also change the code page for an XML target instance in session properties.

CHAPTER 2

Using XML with PowerCenter

This chapter includes the following topics:

Using XML with PowerCenter Overview

You can create an XML definition in PowerCenter from an XML file, DTD file, XML schema, flat file definition, or relational table definition. When you create an XML definition, the Designer extracts XML metadata and creates a schema in the repository. The schema provides the structure from which you edit and validate the XML definition.

An XML definition can contain multiple groups. In an XML definition, groups are called views. The relationship between elements in the XML hierarchy defines the relationship between the views.

When you create an XML definition, the Designer creates views for multiple-occurring elements and complex types in a schema by default. The relative cardinality of elements in an XML hierarchy affects how PowerCenter creates views in an XML definition. Relative cardinality determines if elements can be part of the same view.

The Designer defines relationships between the views in an XML definition by keys. Source definitions do not require keys, but target views must have them. Each view has a primary key that is an XML element or a generated key.

When you create an XML definition, you can create a hierarchical model or an entity relationship model of the XML data. When you create a hierarchical model, you create a normalized or denormalized hierarchy. A normalized hierarchy contains separate views for multiple-occurring elements. A denormalized hierarchy has one view with duplicate data for multiple-occurring elements.

If you create an entity model, the Designer creates views for complex types and multiple-occurring elements. The Designer creates an XML definition that models the inheritance and circular relationships the schema provides.

PowerCenter can work with XML schema that have less than 400 elements. The PowerCenter profile can contain up to three hierarchy levels, and can contain the following complex type elements:

- Sequence
- Any
- Choice

The PowerCenter XML import wizard can create up to 400 views.

Limitations

The following limitations apply to XML handling in PowerCenter:

- The XML schema must be smaller than 400 elements.
- The XML schema file must be smaller than 100 KB.
- The XML file size must be 10 MB or smaller.
- The complexity profile is limited to three hierarchy levels.
- The XML Import Wizard creates a maximum of 400 views.

PowerCenter does not support the following functions:

- **Concatenated columns.** A column cannot be a concatenation of two elements. For example, you cannot create a column FULLNAME that refers to a concatenation of two elements FIRSTNAME and LASTNAME.
- **Composite keys.** A key cannot be a concatenation of two elements. For example, you cannot create a key CUSTOMERID that refers to a concatenation of two elements LASTNAME and PHONENUMBER.
- **Parsed lists.** PowerCenter stores a list type as one string that contains all array elements. PowerCenter does not parse the respective simple types from the string.

To create a transformation with other element types, and to transform larger XML input files, use a Data Processor transformation. For more information about how to create Data Processor transformations, see the *Informatica Data Transformation User Guide* and the *Informatica Data Transformation Getting Started Guide*.

Importing XML Metadata

When you import an XML definition, the Designer creates a schema in the repository for the definition. The repository schema provides the structure from which you edit and validate the XML definition.

You can create metadata from the following file types:

- XML files
- DTD files
- XML schema files
- Relational tables
- Flat files

Importing Metadata from an XML File

In an XML file, a pair of tags marks the beginning and end of each data element. These tags are the basis for the metadata that PowerCenter extracts from the XML file. If you import an XML file without an associated

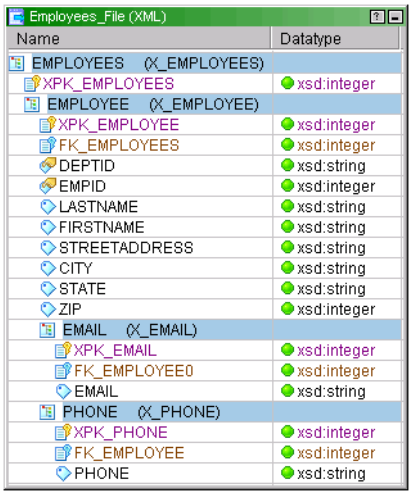
DTD or XML schema, the Designer reads the XML tags to determine the elements, their possible occurrences, and their position in the hierarchy. The Designer checks the data within the element tags and assigns a datatype depending on the data representation. You can change the datatypes for these elements in the XML definition.

The following figure shows a sample XML file:

The root element is Employees. Employee is a multiple occurring element. The Employee element contains the LastName, FirstName, and Address. The Employee element also contains the multiple-occurring elements: Phone and Email.

The Designer determines a schema structure from the XML data.

The following figure shows the default XML source definition with separate views for the root element and the multiple-occurring elements:



When you import an XML file, you do not need all of the XML data to create an XML definition. You need enough data to accurately show the hierarchy of the XML file.

The Designer can create an XML definition from an XML file that references a DTD file or XML schema. If an XML file has a reference to a DTD or an XML schema on another node, the node that hosts the PowerCenter Client must have access to the node where the schema resides so the Designer can read the schema. The XML file contains a universal resource identifier (URI) which is the address of the DTD or an XML schema.

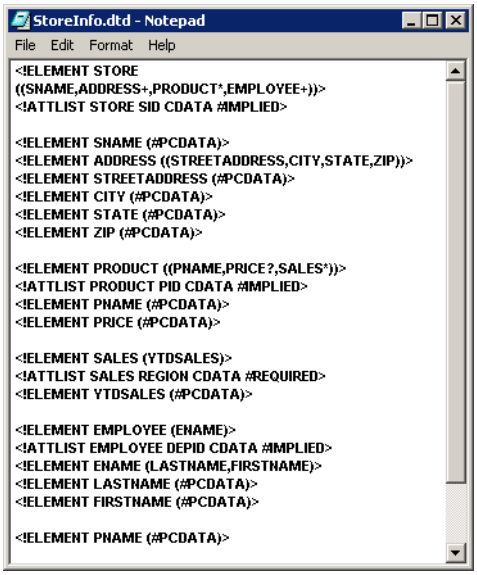
Importing Metadata from a DTD File

A DTD file provides constraints on a XML document structure. A DTD file lists elements, attributes, entities, and notations for an XML document. A DTD file specifies relationships between components. A DTD specifies cardinality and null constraint. However, a DTD file does not contain any data or datatypes.

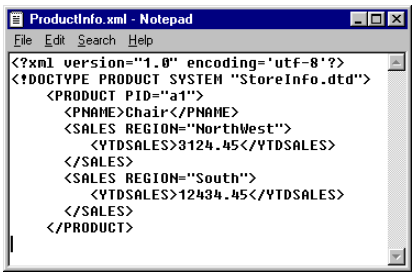
When you import a DTD file, you can change the datatypes for the elements in the XML definition. You can change the null constraint, but you cannot change element cardinality.

If you import an XML file with an associated DTD, the Designer creates a definition based on the DTD structure.

The following figure shows an example of an XML file where StoreInfo.dtd contains the Store element and Product is one of the child elements of Store:

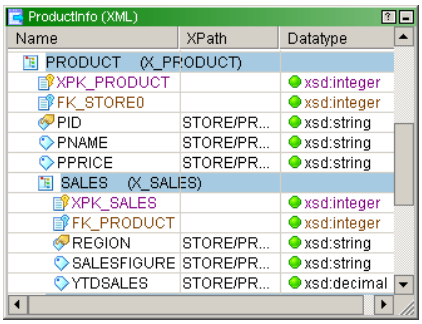


The following figure shows the associated DTD:



In the associated DTD, ProductInfo.xml, uses the Product element from StoreInfo.dtd. Product includes the multiple-occurring Sales element.

The following figure shows the source definition that the Designer creates:



The ProductInfo definition contains the Product and Sales groups. The XML file determines what elements to include in the definition. The DTD file determines the structure of the XML definition.

Importing Metadata from an XML Schema

A schema file defines the structure of elements and attributes in an XML file. A schema file contains descriptions of the type of elements and attributes in the file. When you import an XML schema, the Designer determines the data type, precision, and cardinality of the elements. You cannot change an element definition in PowerCenter if the element definition comes from a schema.

When you import metadata from an XML schema, the .xsd file can contain import or include statements that reference other .xsd files. When you import a schema that includes other schemas, the other schemas must not reference the same namespace.

Example:

```
<IMPORT
schemaLocation="../../../administration/process/bo/LocationTextBO.xsd"
namespace="http://EnterpriseLibrary/com/acs/enterprise/common/program/administration/
process/bo">
<IMPORT
schemaLocation="../../../administration/process/bo/LineOfBusinessBO.xsd"
namespace="http://EnterpriseLibrary/com/acs/enterprise/common/program/administration/
process/bo">
<IMPORT
schemaLocation="../../../administration/process/bo/ClaimExceptionBO.xsd"
namespace="http://EnterpriseLibrary/com/acs/enterprise/common/program/administration/
process/bo">
```

You can replace multiple "import schemalocation" statements with one statement:

```
<xsd:import schemalocation="imported.xsd" namespace="http://EnterpriseLibrary/com/acs/
enterprise/common/program/administration/process/bo"/>
```

The imported.xsd file includes the other XSD files using the following syntax:

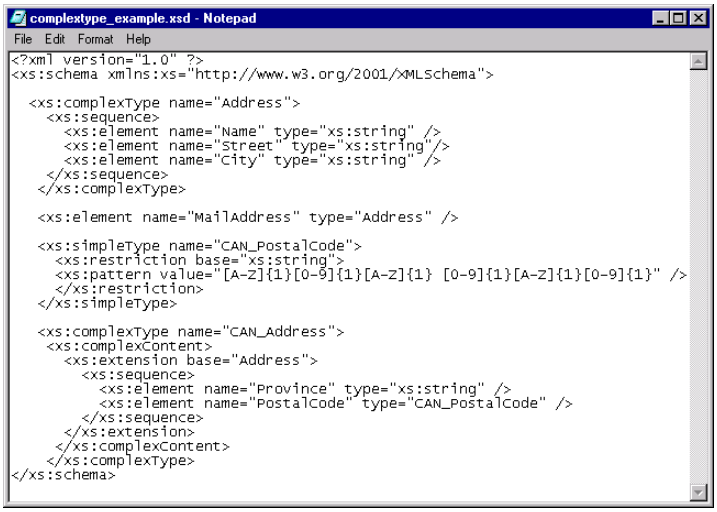
```
   <xsd:schema targetNamespace="http://EnterpriseLibrary/com/acs/enterprise/common/
program/administration/process/bo" elementFormDefault="qualified" >
      <xsd:include schemaLocation=" LocationTextBO.xsd" />
      <xsd:include schemaLocation=" LineOfBusinessBO.xsd" />
      <xsd:include schemaLocation=" ClaimExceptionBO.xsd" />
   </xsd:schema>
```

For more information, see Knowledge Base article 158334.

Each simple type definition in an XML schema is a restriction of another simple type definition in the schema Atomic data types, such as Boolean, string, or integer, restrict the anySimpleType datatype. When you define a simple data type in an XML schema, you derive a new datatype from an existing data type. For example, you can derive a restricted integer type that holds only numbers from 1 to 20. The base type is integer.

When you derive a complex data type from another data type, you create a new datatype that contains the elements of the base type. You can add new elements to the derived type or create restrictions on the inherited elements. The Designer creates views for derived types without duplicating the columns that represent inherited components. This reduces metadata and decreases the size of the XML definition in the repository.

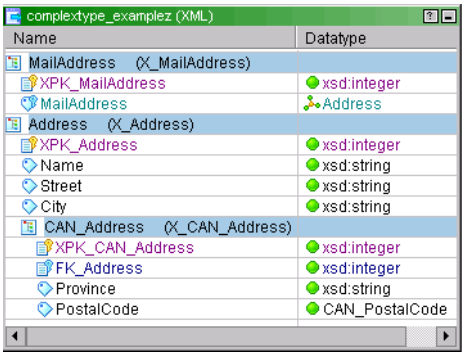
The following image shows a schema with simple and complex derived types:



The MailAddress element is an Address type which is a complex type. A derived type, CAN_Address, inherits the Name, City, and Street from the Address type, and extends Address by adding a Province and PostalCode. PostalCode is a simple type called CAN_PostalCode.

When you import an XML schema, every simple type or attribute in a complex type can become a column in an XML definition. Complex types become views.

The following figure shows an XML definition from the schema if you import the schema with the default options:



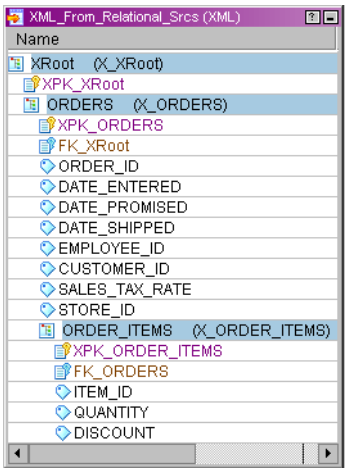
The CAN_Address view contains the elements that are unique for its type. The root element is MailAddress. The Address type contains Name, Street, and City. The CAN_Address has a foreign key to Address. CAN_Address includes Province and PostalCode.

The view does not contain the Name, Street, and City that it inherits from MailAddress.

Creating Metadata from Relational Definitions

You can create an XML definition by selecting multiple relational definitions and creating relationships between them. The Designer creates an XML view for each relational definition you import. The Designer converts every column in the relational definition and generates primary key-foreign key relationships. You can choose to create a root view.

The following figure shows a sample XML target definition from the relational definitions, Orders and Order_Items:

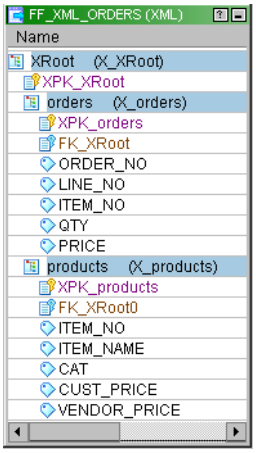


The root is XRoot. XRoot encloses Orders and Order Items. Order_Items has a foreign key that points to Orders.

Creating Metadata from Flat Files

You can create an XML definition by importing a flat file definition from the repository. If you import more than one flat file definition, the Designer creates an XML definition with a view for each flat file. The views have no relationship to each other in the XML definition. If you choose to create a root view, the Designer creates the views with foreign keys to the root.

The following figure shows a sample XML source definition from flat files orders and products:



Products and Orders have a foreign key to the root view, and are highlighted.

Understanding XML Views

The relationship between elements in the XML hierarchy defines the relationship between XML views in a PowerCenter definition. In a source definition, a view does not have to be related to any other view. Therefore, views in a source definition do not require primary or foreign keys. A denormalized view can be independent of any other view. However, the Designer generates keys if you do not designate key columns when views are related to other views.

Each view in a target definition must be related to at least one other group. Therefore, each view needs at least one key to establish its relationship with another view. If you do not designate the keys, the Designer generates primary and foreign keys in the target views. You can define primary and foreign keys for views if you create the views and relationships in the XML Editor instead of allowing the Designer to create them for you.

When the Designer creates a primary or foreign key column, it assigns a column name with a prefix. In an XML definition, the prefixes are XPK_ for a generated primary key column and XFK_ for a generated foreign key column. The Designer uses the prefix FK_ for a foreign key that points to a primary key.

For example, when the Designer creates a primary key column for the Sales group, the Designer names the column XPK_Sales. When the Designer creates a foreign key column connecting a sales group to another group, it names the column XFK_Sales. You can rename any column name that the Designer creates.

If a mapping contains an XML source, the Integration Service creates the values for the generated primary key columns in the source definition when you run the session. You can configure start values for the generated keys.

Creating Custom XML Views

Custom views are groups that you create with the XML Wizard or the XML Editor. If you use the XML Wizard to create custom views, the wizard creates views containing all the components in the schema. If you use the XML Editor, you can define each view and choose the components.

The elements in the views and the relationship between views are dependent on the schema the Designer creates in the repository when you import the definition. The XML Editor validates XML definitions using the rules for valid views.

Rules and Guidelines for XML Views

Consider the following rules and guidelines when you work with view keys and relationships:

- An PowerCenter XML definition can have up to 400 views.
- A view can have one primary key.
- A view can be related to several other views, and a view can have multiple foreign keys.
- A column cannot be both a primary key and a foreign key.
- A view in a source definition does not require a key.
- A view in a target definition requires at least one key.
  - The target root view requires a primary key, but the target root does not require a foreign key.
  - A target leaf view requires a foreign key, but the target leaf view does not require a primary key.
- An enclosure element cannot be a key.
- A foreign key always refers to a primary key in another group. You cannot use self-referencing keys.
- A generated foreign key column always refers to a generated primary key column.

- The relative cardinality of elements in an XML hierarchy affects how PowerCenter creates views in an XML definition. The following rules determine when elements can be part of the same view:
  - Elements that have a one-to-one relationship can be part of the same view.
  - Elements that have a one-to-many relationship can be part of the same normalized or denormalized view.
  - Elements that have a many-to-many relationship cannot be part of the same view.

Understanding Hierarchical Relationships

An XML definition with hierarchical view relationships has each element in the hierarchy appear under its parent element in a view. Multiple-occurring elements can become views. Complex types do not become views, and elements unique to derived complex types do not occur in any view.

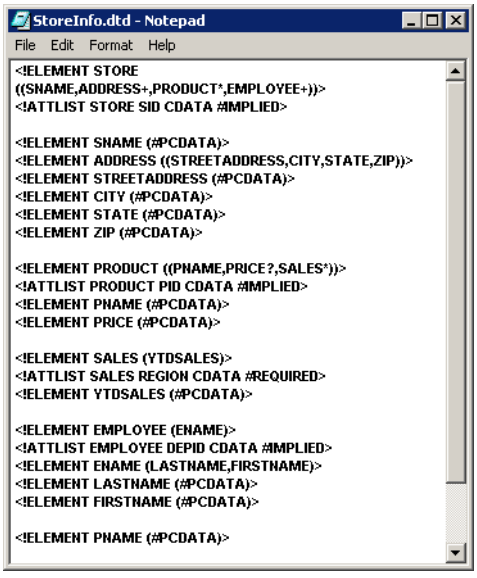
You can generate the following types of hierarchical view:

- **Normalized views.** An XML definition with normalized views reduces redundancy by separating multiple-occurring data into separate views. The views are related by primary and foreign keys.
- **Denormalized views.** An XML definition with a denormalized view has all the elements of the hierarchy that are not unique to derived complex types in the view. A source or target definition can contain one denormalized view.

Normalized Views

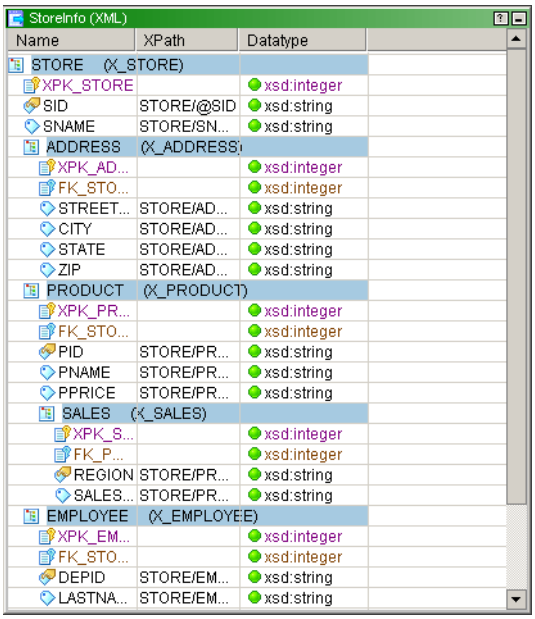
When the Designer generates a normalized view, it establishes the root element and the multiple-occurring elements that become views in an XML definition.

The following figure shows a DTD file and the elements that become views in a normalized XML definition:



Store is the root element. Address, product, employee, and sales are multiple-occurring elements.

The following figure shows a source definition based on the DTD file from the figure above:



The definition has normalized views. The root view is Store. The Address, Product, and Sales views have foreign keys to Store. The Sales view has a foreign key to the Product view.

The following table shows the rows in a data preview for the Store view:

XPK_si_STORE	si_SID	si_NAME
1	BE1752	Mud and Sawdust Furniture Store

The following table shows the rows in a data preview for the Address view:

XPK_si_ADDRESS	FK_si_ADDRESS	si_STREETADDRESS	si_CITY	si_STATE	si_ZIP
1	1	335 Westshore Road	Fausta City	CA	95784
2	1	7415 Endless Loop	Codeville	CA	97412

The following table shows the rows in a data preview for the Product view:

XPK_si_PRODUCT	FK_si_PRODUCT	si_PNAME	si_PRICE	si_PID
1	1	Chair	5690.00	a1
1	1	Table	1240.00	a2
1	1	Bed	1364.99	a3

The following table shows the rows in a data preview for the Sales view:

XPK_si_SALES	FK_si_SALES	si_REGION	si_YTDSALES
1	a1	Northwest	4565.44
2	a2	South	8793.99
3	a3	East	23110.00
4	a4	South	5500.00
5	a5	Northwest	10095.34
6	a6	East	200.00

The following table shows the rows in a data preview for the Employee view:

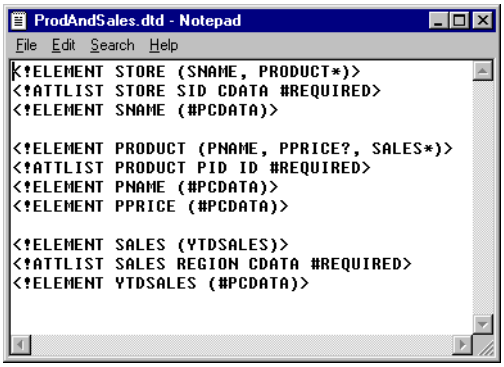
XPK_si_EMPLOYEE	FK_si_EMPLOYEE	si_FIRSTNAME	si_LASTNAME
1	1	James	Bond
2	1	Austin	Powers
3	1	Indiana	Jones
4	1	Foxie	Brown
5	1	Bonnie	Bell
6	1	Laura	Croft

Denormalized Views

When the Designer generates a denormalized view, it creates one view and puts all elements of the hierarchy into the view. All the elements in a denormalized view belong to the same parent chain. Denormalized views, like denormalized tables, generate duplicate data.

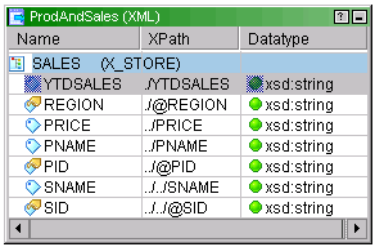
The Designer can generate denormalized views for XML definitions that contain more than one multiple-occurring element if the multiple-occurring elements have a one-to-many relationship and are all part of the same parent chain.

The following figure shows a DTD file that contains multiple-occurring elements, in this case Product and Sales:



Product and Sales are multiple-occurring elements. Because the multiple-occurring elements have a one-to-many relationship, the Designer can create a single denormalized view that includes all elements.

The following figure shows the denormalized view for ProdAndSales.dtd in a source definition:



The Designer creates a single view for all the elements in the ProdAndSales hierarchy. Because a DTD file does not define datatypes, the Designer assigns a datatype of string to all columns. The denormalized view does not need a primary or foreign key.

The following figure shows a data preview for the denormalized view:

SID	SNAME	PID	PNAME	PPRICE	REGION	YTDSALES
BE1752	Mud and Sawdust Furniture Store	a1	Chair	NULL	NorthWest	3124.45
BE1752	Mud and Sawdust Furniture Store	a1	Chair	NULL	South	12434.45
BE1752	Mud and Sawdust Furniture Store	a2	Table	4126.33	South	8252.66
BE1752	Mud and Sawdust Furniture Store	a3	Bed	4356.33	NorthWest	43563.30
BE1752	Mud and Sawdust Furniture Store	a3	Bed	4356.33	South	21781.65
BE1752	Mud and Sawdust Furniture Store	a3	Bed	4356.33	East	26137.98
BE1752	Mud and Sawdust Furniture Store	a4	Etagere	5000.00	South	20000.00
BE1752	Mud and Sawdust Furniture Store	a4	Etagere	5000.00	East	5000.00
BE1752	Mud and Sawdust Furniture Store	a4	Etagere	5000.00	NorthWest	15000.00

Understanding Entity Relationships

You can create entity relationships from an XML schema. When you create an XML definition that contains entity relationships, the Designer generates separate views for multiple-occurring elements, element groups, and complex types. The Designer includes views for all derived complex types. The Designer creates links and keys between the views based on type and hierarchy relationships.

When you work with XML schemas, you can reference parts of the schema rather than repeat the same information in schema components. A component can inherit the elements and attributes of another component and restrict or extend the elements from the component. For example, you might use a complex

type as a base for creating a new complex type. You can add more elements to the new type to create an extended complex type. Or, you might create a restricted complex type, which is a subset of another complex type.

If you create views manually or re-create entity relationships in the XML Editor, you choose how you want to structure the metadata. When you create an XML definition based on an XML schema that uses inheritance, you can generate separate views for the base type and derived type. You might create inheritance relationships if you plan to map the XML data to normalized relational tables.

An XML Type I inheritance relationship is a relationship between two views. Each view root is a global complex type. One view is derived from the other.

You can create an inheritance relationship between a column and a view. This is an XML Type II inheritance relationship.

The Designer generates separate views for substitution groups.

Rules and Guidelines for Entity Relationships

The Designer generates entities based on the following guidelines:

- An entity represents a portion of an XML, DTD, or XML schema hierarchy. This hierarchy does not need to start at the root of the XML file.
- The Designer uses entities defined in a DTD file to create entity relationships.
- The Designer uses type structures defined in an XML schema to generate entity relationships.
- The Designer creates a new entity when it encounters a multiple-occurring element under a parent element.
- The Designer generates a separate view for each member of a substitution group.
- The Designer generates primary keys and foreign keys to relate separate entities.

Type 1 Entity Relationship Example

An XML Type 1 entity relationship is a relationship between two views. Each view must be rooted as a global complex type. One view must be derived from the other.

The following schema contains a PublicationType, BookType, and MagazineType. PublicationType is the base type. A publication includes Title, Author, and Date. BookType and MagazineType are derived types that extend the PublicationType. Book has ISBN and Publisher, and Magazine has Volume and Edition.

```
<xsd:complexType name="PublicationType">
   <xsd:sequence>
     <xsd:element name="Title" type="xsd:string"/>
     <xsd:element name="Author" type="xsd:string" maxOccurs="unbounded"/>
     <xsd:element name="Date" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:element name="Publication" type="PublicationType"/>
<xsd:complexType name="BookType">
   <xsd:complexContent>
   <xsd:extension base="PublicationType">
   <xsd:sequence>
     <xsd:element name="ISBN" type="xsd:string"/>
     <xsd:element name="Publisher" type="xsd:string
   </xsd:sequence>
   </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="MagazineType">
   <xsd:complexContent>
   <xsd:extension base="PublicationType">
```

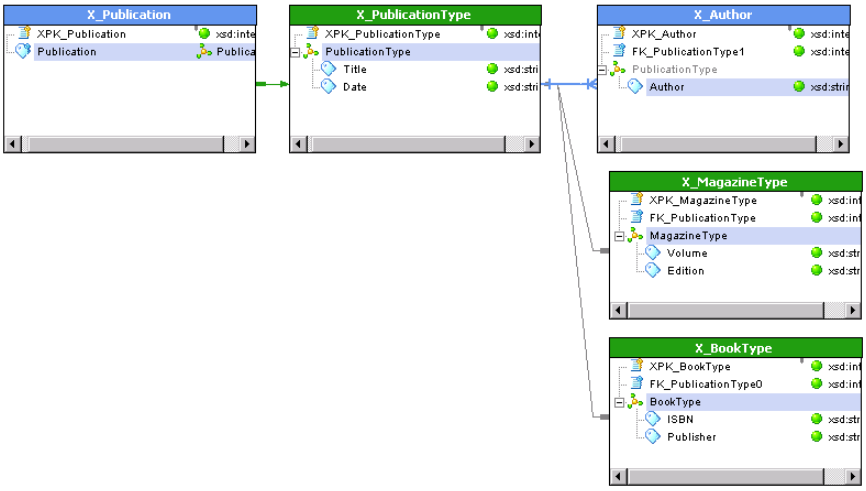
```
    <xsd:sequence>
      <xsd:element name="Volume" type="xsd:string"/>
      <xsd:element name="Edition" type="xsd:string"/>
    </xsd:sequence>
    </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>
  </xsd:schema>
```

When you create XML views as entities in an XML definition, the Title and Date metadata from PublicationType do not repeat in BookType or MagazineType. Instead, these views contain the metadata that distinguishes them from the PublicationType: ISBN and Publisher for BookType, and Volume and Edition for MagazineType. They have foreign keys that link them to PublicationType.

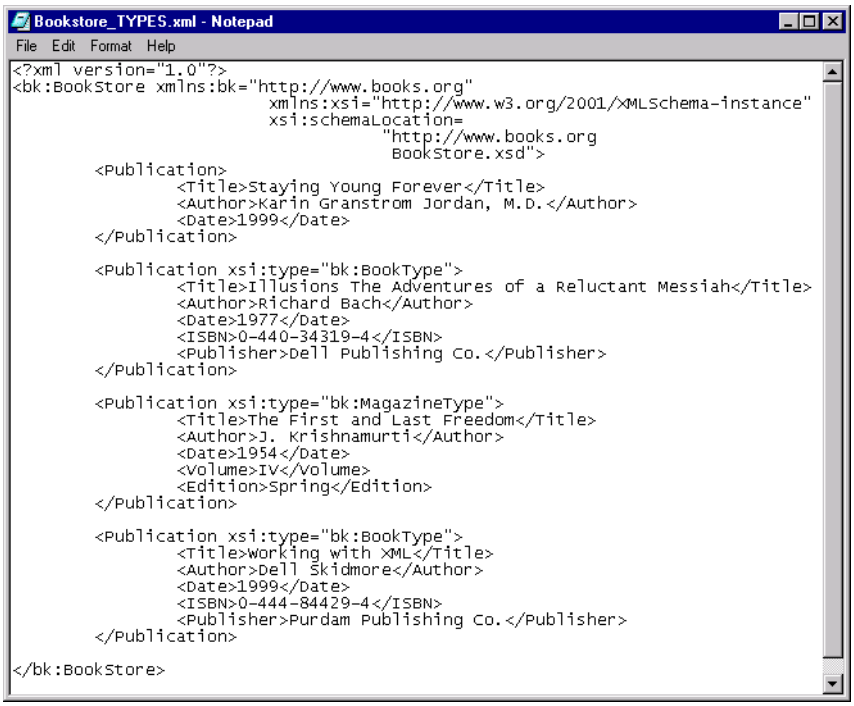
This example uses reduced metadata explosion because none of the elements in the base type repeat in the derived types.

Author is a multiple-occurring element in Publication. Author becomes an XML view.

The following figure shows the default views the Designer generates from the schema:



The following figure shows an XML file that has a publication, a magazine, and books:



If you process the sample XML file using the XML definition in the preceding figure, you create data in the following views:

- **PublicationType view.** Contains the title and date for each publication.

  The following figure shows the PublicationType view:

XPK_boo_PublicationType	Title	Date
1	Staying Yo...	1999
2	Illusions Th...	1977
3	The First a...	1954
4	Working wi...	1999

- **BookType view.** Contains the ISBN and publisher. BookType contains a foreign key to PublicationType.

  The following figure shows the BookType view:

XPK_boo_BookType	FK_boo_PublicationType1	ISBN	Publisher
1	2	0-440-34319-4	Dell Publishing Co.
2	4	0-444-84429-4	Purdam Publishing Co.

- **MagazineType view.** Contains volume and edition. MagazineType also contains a foreign key to the PublicationType.

  The following figure shows the MagazineType view:

XPK_boo_MagazineType	FK_boo_PublicationType	Volume	Edition
1	3	IV	Spring

- **Author view.** Contains authors for all the publications. The Designer generates a separate view for Author because Author is a multiple-occurring element. Each publication can contain multiple authors.

  The following figure shows the Author view:

XPK_Author	FK_boo_PublicationType0	Author
1	1	Karin Granstrom Jordan, M.D.
2	2	Richard Bach
3	3	J. Krishnamurti
4	4	Dell Skidmore

Type II Entity Relationship Example

You can create an inheritance relationship between a column and a complex type view. The column must be an element of a local complex type. The view must be rooted at a global complex type. The local complex type must be derived from the global complex type.

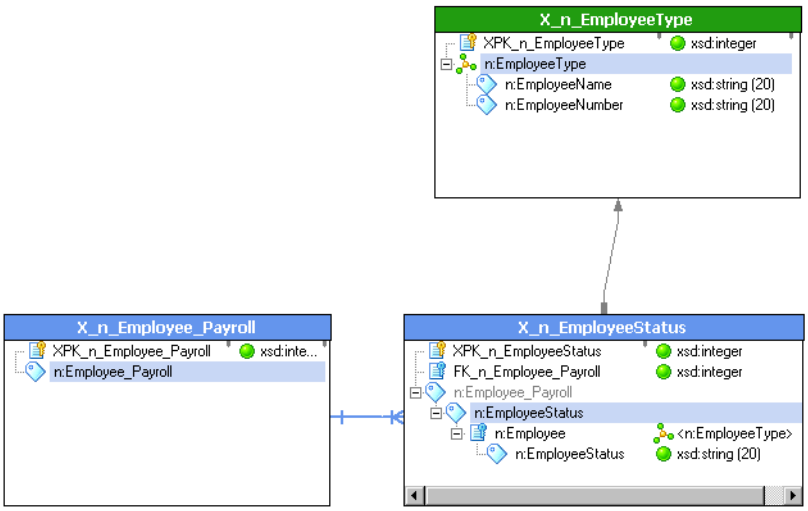
For example, the following schema defines a complex type called EmployeeType. EmployeeType contains EmployeeNumber and EmployeeName elements.

EmployeeStatusType includes an element called Employee that extends EmployeeType. Employee includes an EmployeeStatus element.

```
<xs:element name="Employee_Payroll">
   <xs:complexType>
   <xs:sequence>
      <xs:element name="EmployeeStatus" type="EmpStatusType"
         maxOccurs="unbounded"></xs:element>
   </xs:sequence>
   </xs:complexType>
</xs:element>
<xs:complexType name="EmpStatusType">
   <xs:sequence>
      <xs:element name="Employee" minOccurs="0" maxOccurs="1">
         <xs:complexType>
         <xs:complexContent>
         <xs:extension base="EmployeeType">
            xs:sequence>
               <xs:element name="EmployeeStatus" type="xs:string">
              </xs:element>
            </xs:sequence>
         </xs:extension>
         </xs:complexContent>
         </xs:complexType>
      </xs:element>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="EmployeeType">
   <xs:sequence>
      <xs:element name="EmployeeName" type="xs:string"></xs:element>
      <xs:element name="EmployeeNumber" type="xs:string"></xs:element>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

When you import the schema, the Designer creates a view for Employee_Payroll, EmployeeType, and EmployeeStatus. The EmployeeStatus view contains the column called Employee. Employee derives from EmployeeType.

The following figure shows the Employee_Payroll view, the EmployeeType view, and the EmployeeStatus XML view:



The Employee_Payroll view contains the Employee_Payroll element and a primary key, PK_Employee_Payroll. The Employee_Payroll view is connected to the EmployeeStatus view by a blue line that indicates a one- to-many relationship between the views. Employee_Payroll contains multiple occurrences of EmployeeStatus.

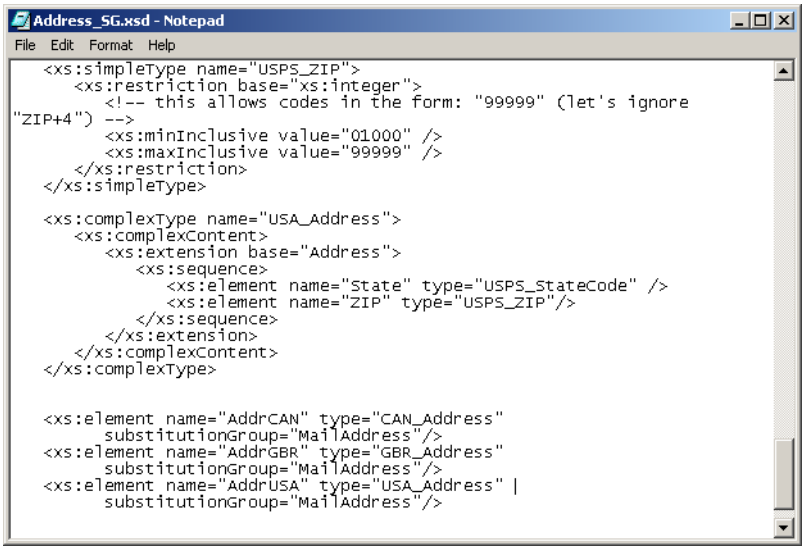
The EmployeeStatus view contains an Employee element of type EmployeeType. The Employee element extends EmployeeType by including an EmployeeStatus element. The EmployeeStatus view also contains a foreign key to Employee_Payroll. The EmployeeStatus view is connected to an EmployeeType view with a gray arrow. The arrow indicates a type relationship between the views.

The EmployeeType view contains an EmployeeType that consists of EmployeeName and EmployeeNumber.

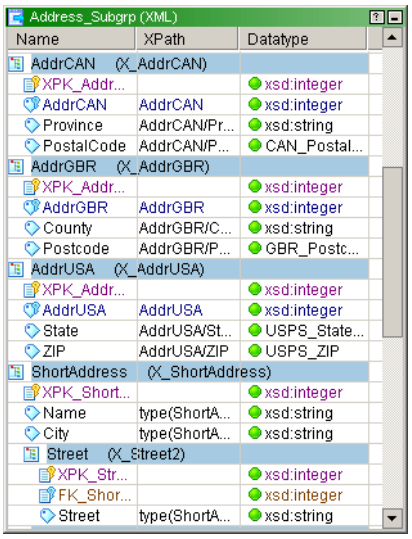
Using Substitution Groups in an XML Definition

When you create an XML definition containing entity relationships, the Designer generates separate views for element groups and complex types. When you import an XML schema that uses substitution groups, the Designer imports each member of the substitution group as a separate entity. The Designer makes a separate view for each group.

The following figure shows a sample portion of an XML schema containing the substitution group MailAddress:



The following figure shows an XML definition with a view for each member of the substitution group, including AddrCAN, AddrGBR, AddrUSA, ShortAddress, and Street:



Working with Circular References

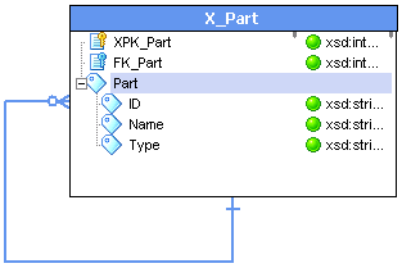
A circular relationship is a circular hierarchy relationship between two views in an XML definition or within a single view in an XML definition. For example, a complex element called Part might contain an ID, part name, and a reference to another part.

The following example shows the Part element components:

```
<xs:element name="Part">
<xs:complexType>
   <xs:sequence>
   <xs:element name="ID" type="xs:string"/>
```

```
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Type" type="xs:string"/>
    <xs:element ref="Part" minOccurs="0" maxOccurs="unbounded"/>
    /xs:sequence>
</xs:complexType>
</xs:element>
```

The following figure shows a circular reference in the XML Editor workspace with a complex element called Part:



You might use the Part XML definition to read the following XML file in a session:

```
<Part>
<ID>1</ID>
<Name>Big Part</Name>
<Type>L</Type>
   <Part>
   <ID>1.A</ID>
   <Name>Middle Part</Name>
       <Type>M</Type>
      <Part>
      <ID>1.A.B</ID>
      <Name>Small Part</Name>
      <Type>S</Type>
      </Part>
   </Part>
</Part>
```

In the XML file, Part 1 contains Part 1.A, and Part 1.A contains Part 1.A.B.

The following figure shows the data and the keys that a session might generate from the XML source:

XPK_Part	FK_Part	ID	Name	Type
1	NULL	1	Big Part	L
2	1	1.A	Middle Part	M
3	2	1.A.B	Small Part	S

Note: You cannot run a session that contains a circular XML reference if the session is enabled for constraint-based loading. The session rejects all rows.

Understanding View Rows

To extract data from an XML document, you specify the rows to generate, the columns of data to include, and when to generate the rows. When you define a view in the XML Editor, you create the view row, an element or a global complex type that the Integration Service requires to generate a row of data.

The Integration Service uses a view row to determine when to read and write data for an XML view. You can set a view row at any single or multiple-occurring element. Once you set the view row, every element you add to the view has a one-to-one correspondence with the view row.

For example, the Employees view contains elements Employee, Name, Firstname, and Lastname. When you set the view row to Employee, the Integration Service extracts data using the following algorithm:

```
For every (Employees/Employee)
extract ./Name/Firstname/Lastname
```

An Employees XML schema might contain the following elements:

```
EMPLOYEES
    EMPLOYEE+
        ADDRESS+
        NAME
           FIRSTNAME
            LASTNAME
        EMAIL+
```

Employee, Address, and Email are multiple-occurring elements. You can create a view that contains the following elements:

```
EMPLOYEE
    ADDRESS
    NAME
```

If you set the view row as Address, the Integration Service extracts a Name for every Employee/Address in the XML data. You cannot add Email to this view because you would create a many-to-many relationship between Address and Email.

You can add a pivoted multiple-occurring column to the view. A view row can contain the pivoted column.

For example, you can add one instance of Email as a pivoted column to the Employee view. The view would contain the following elements:

```
EMPLOYEE
    ADDRESS
    NAME
    EMAIL[1]
```

The view might have the view row, `EMPLOYEE/ADDRESS/EMAIL[1]`. The Integration Service extracts data for the first instance of Employee/Address/Email.

Using XPath Query Predicates

Use a query in an XML view to filter XML source data. The Integration Service extracts data from a source XML file based on the query. If the query is true, the Integration Service extracts data for the view.

To create a query in an XML view, you create an XPath query predicate in the XML Editor. XPath is a language that describes a way to locate items in an XML document. XPath uses an addressing syntax based on the path through the XML hierarchy from a root component. You can create an XPath query predicate for elements in the view row or elements and attributes that have an XPath that includes the view row.

An XPath query predicate includes an element or attribute to extract, and the query predicate that determines the criteria. You can verify the value of an element or attribute, or you can verify that an element or attribute exists in the source XML data.

Rules and Guidelines for Using View Rows

Use the following rules and guidelines to use view rows in an XML definition:

- A view row must be a type or an element. A view row cannot be an attribute.
- Every view must have a view row, which must be an element or complex type.
- The view root is the top-level element in a view. The view root is the parent to all the other elements in the view.
- The view row can be the same as the view root unless the view is denormalized.
- Two views can have the same view row in an XML source or XML Parser transformation.
- The view row element must be the lowest multiple-occurring element in the view. A view cannot contain many-to-many relationships.
- If you add a multiple-occurring element to a view with no other multiple-occurring element, you change the view row to the new element by default. If the view already has a multiple-occurring element, you cannot add another multiple-occurring element.
- You do not need to specify a view row when you create an empty view. However, as soon as you add a column to the view, the Designer creates the view row. This is true even if you add just the primary key.
- You can change a view row at a later time, but you cannot change a view root unless there are no schema components in the view.
- You can specify a view row that consists of a pivoted element, such as:

  ```
  Product/Order[2]/Customer
  ```

- An effective view row for a view is the path of view rows from the top of a hierarchy relationship down to the view row in the view. A view can have multiple effective view rows because the view can have multiple hierarchy relationships in the XML definition.

You can specify options in the XML Editor that affect how view rows and effective view rows affect data output.

Pivoting Columns

Sometimes an element that occurs multiple times is a set of the same elements containing different values. For example, an element called Sales that occurs 12 times might contain the sales figures for each month of the year. Or, an element called Address that occurs twice might be a home address and an office address.

If you have this type of element in an XML source, use pivoting to treat occurrences of elements as separate columns in a group. To pivot occurrences of an element in an XML view, create a column for each occurrence you want to represent in the definition. In the monthly sales example, if you want to represent all 12 occurrences as columns, create 12 sales columns in the view. If you want to represent the sales of one quarter, create three columns. When you run a session, the Integration Service ignores any XML data for the occurrences that you do not include in the definition.

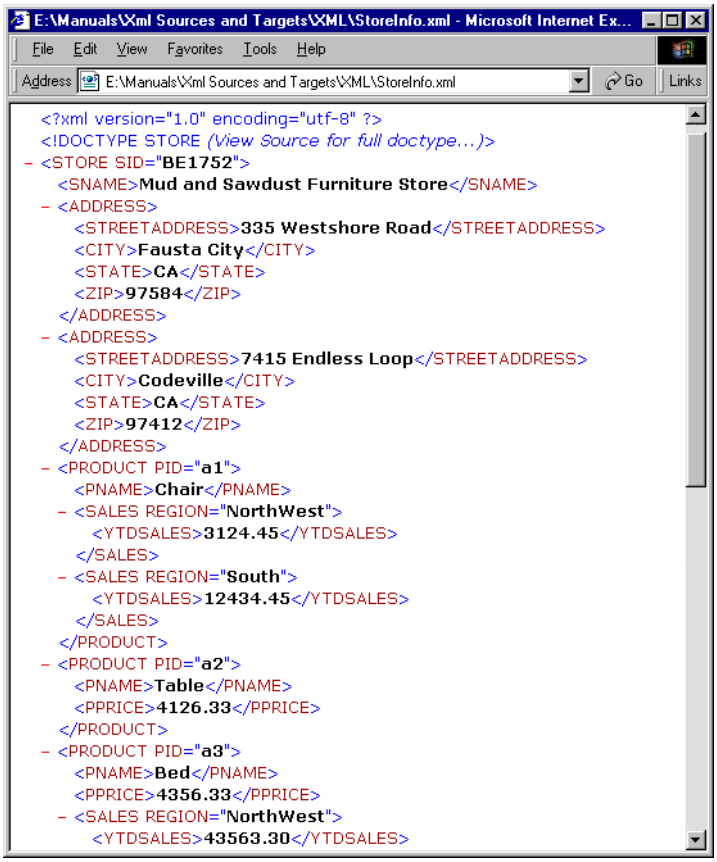
You can pivot columns when you add or edit a view in the XML source definition.

You can pivot simple types and complex types. You cannot pivot a primary key column. When you pivot columns in a view, the resulting group structure must follow the rules for a valid normalized or denormalized view. The Designer displays warnings and errors if the pivoted column invalidates a view.

Pivoting affects an element in the view where you pivot the element. When you pivot an element in a view, you do not change same element in another view.

Note: You cannot pivot columns in an XML target.

The following figure shows two occurrences of the Address element in the StoreInfo XML file:



First occurrence of Address pivots to home address columns with prefix HOM_. The second occurrence of Address pivots to office address columns with prefix OFC_. XPath shows the two sets of columns that come from the same elements.

The following figure shows the ADDRESS element of the StoreInfo XML file pivoted into two sets of address columns:

PowerCenter Column ...	Type	Not ...	XPath
SID	xsd:short	☐	./@SID
si_SNAME	xsd:string (30)	☐	./si:SNAME
HOM_STREETADDRESS	xsd:string (30)	☐	./si:ADDRESS[1]/STREETADDRESS
HOM_CITY	xsd:string (30)	☐	./si:ADDRESS[1]/CITY
HOM_STATE	xsd:string (30)	☐	./si:ADDRESS[1]/STATE
HOM_ZIP	xsd:string (30)	☐	./si:ADDRESS[1]/ZIP
OFC_STREETADDRESS	xsd:string (30)	☐	./si:ADDRESS[2]/STREETADDRESS
OFC_CITY	xsd:string (30)	☐	./si:ADDRESS[2]/CITY
OFC_STATE	xsd:string (30)	☐	./si:ADDRESS[2]/STATE
OFC_ZIP	xsd:string (30)	☐	./si:ADDRESS[2]/ZIP

In the following figure, the first and second address occurrences (with HOM_ and OFC_ prefixes) appear as columns in the group:

GPK_ADDRESS	FK_SID	HOM_STREETADDRESS	HOM_CITY	HOM_STATE	HOM_ZIP	OFC_STREETADDRESS	OFC_CITY	OFC_STATE	OFC_ZIP
1	1	335 West Bayshore Road	Fausta city	CA	97584	7415 Endless Loop	Codeville	CA	97412

Using Multiple-Level Pivots

You can pivot more than one level of elements in a view by specifying fixed offsets for multiple-occurring elements in the XPath for a column. For example, you might have the following elements in a view:

```
STORE
    PRODUCT+
         PNAME
         ORDER+
         ORDERNAME
             CUSTOMER+
                  CUSTNAME
```

The XPath `STORE/PRODUCT[2]/ORDER[1]/ORDERNAME` refers to the ordername for the first order for the second product in the store. The XPath `STORE/PRODUCT[2]/ORDER/CUSTOMER[1]` refers to the first customer for all orders of the second product.

If you pivot a view row, any column in the XML view that occurs below the view row must have an XPath that matches XPath of the view row.

For example, a view might have the following view row:

```
Transaction/Trade[1]
```

The following columns have the same occurrence of Trade in the XPath:

```
Transaction/Trade[1]/Date
Transaction/Trade[1]/Price
Transaction/Trade[1]/Person[1]/Firstname
```

You cannot create a column with the following XPath in the view:

```
Transaction/Trade[2]/Date
```

CHAPTER 3

Working with XML Sources

This chapter includes the following topics:

Working with XML Sources Overview

The Designer provides an XML Wizard that you can use to create XML definitions in the repository. You can import files from a URL or local node to create an XML definition. You can also import relational or flat file definitions from a PowerCenter repository. You can create XML definitions from the following file types:

- XML files
- XML schema files
- DTD files
- Relational definitions
- Flat file definitions

When you create XML definitions, you import files with the XML Wizard and organize metadata into XML views. XML views are groups of columns containing the elements and attributes in the XML file. The wizard can generate views for you, or you can create custom views.

You can create relationships between views in the XML Wizard. You can create hierarchy relationships or entity relationships.

You can synchronize an XML definition against an XML schema file if the structure of the schema changes.

Importing an XML Source Definition

When you import a source definition from an XML schema or DTD file, the Designer can provide an accurate definition of the data based on the description provided in the DTD or XML schema file. When you import a source definition based on an XML file without an associated DTD or XML schema, the XML Wizard determines the types and occurrences of the data based on data represented in the XML file. When you create the XML definition, you can get unexpected results. For example, the Designer might define an inaccurate scale attribute for string columns. If you export the XML source definition and import the definition with the inaccurate scale attributes, errors occur.

After you create an XML source definition, you cannot change the source definition to any other source type. Conversely, you cannot change other types of source definition to XML definitions.

The XML Wizard uses keys to relate the XML views and reconstruct the XML hierarchy. You can choose to generate views and primary keys, or you can create views and specify keys. When you create custom views, you can select roots and choose how to handle metadata expansion.

The XML Wizard saves the XML hierarchy and the view information as an XML schema in the repository. When you import an XML definition, the ability to change the cardinality and datatype of the elements in the hierarchy depends on the type of file you are importing. For example, DTD and XML files do not store datatype information. When you import these files to create an XML definition, you can configure datatype, precision, and scale in the Designer. If you import an XML schema, you can change the precision and scale.

You cannot create an XML source definition from an XML file of exported repository objects.When you import a source definition, the Designer applies a default code page to the XML definition in the repository. The code page is based on the PowerCenter Client code page. You cannot change the code page for an XML source definition, but you can change the code page for an XML target definition after you create it.

Use the XML Wizard to import XML source definitions.

To import an XML file:

1. Click Sources > Import XML Definition.

   The Import XML Definition dialog box appears.

2. Click Advanced Options.

   The Change XML Views Creation and Naming Options dialog box appears. Select options to specify how the Designer creates and names XML views.

   The following table describes the XML view options:

Option	Description
Override all infinite lengths	You can specify a default length for components with undefined lengths, such as strings. If you do not set a default length, the precision for these components sets to infinite. Infinite precision can cause DTM buffer size errors when you run a session with large files.
Analyze elements/attributes in standalone XML as global declarations	Choose this option to create global declarations of standalone XML elements or attributes. You can reuse global elements by referencing them in other parts of the schema. When you clear this option, the standalone XML is a local declaration.

Option	Description
Create an XML view for an enclosure element	You can create a separate view for an enclosure element if the element can occur more than once and the child elements can occur more than once. An enclosure element is an element that has no text content or attributes but has child elements.
Pivot elements into columns	You can pivot leaf elements if they have an occurrence limit. You can pivot elements in source definitions only.
Ignore fixed element and attribute values	You can ignore fixed values in a schema and allow other element values in the data.
Ignore prohibited attributes	You can declare an attribute as prohibited in an XML schema. Prohibited attributes restrict complex types. When you import the schema or file, you can choose to ignore the prohibited attributes.
Generate names for XML columns	You can choose to name XML columns with a sequence of numbers or with the element or attribute name from the schema. If you use names, choose from the following options: - When the XMLColumn refers to an attribute, prefix it with the element name. PowerCenter uses the following format for the name of the XML column: `NameOfElement_NameOfAttribute` - Prefix the XML view name for every XML column. PowerCenter uses the following format for the name of the XML column: `NameOfView_NameOfElement` - Prefix the XML view name for every foreign-key column. PowerCenter uses the following format for the name of a generated foreign key column: `FK_NameOfView_NameOfParentView_NameOfPKColumn` Maximum length for a column name is 80 characters. PowerCenter truncates column names longer than 80 characters. If a column name is not unique, PowerCenter adds a numeric suffix to keep the name unique.

3. Click OK to apply changes.
4. Choose the type of file to import. You can choose the following options:
   - **Import the definition from a local XML file or a URL.** Create a source definition from an XML, DTD, or XML schema file. If you import an XML file with an associated DTD or schema, the XML Wizard uses the DTD or schema to generate the XML document.
   - **Import the definition from a non-XML source or target.** Use this option to create a source definition from flat file or relational definitions. The new source definition contains one group for each input definition plus a root element group.
5. Click Next to complete the XML Wizard.

Multi-line Attributes Values

The XML Wizard does not allow attribute values that contain new line characters and span more than one line. When you import a source or target definition from an XML file that contains an attribute with new line characters, the XML Wizard displays an error and does not import the file.

Working with XML Views

The Designer displays views as groups in the source definition.

The XML Wizard provides options for creating views in the definition or you can create the views manually in the XML Editor.

You can choose from the following options to create XML views:

- **Generate entity relationships**. If you create entity relationships, the XML Wizard generates views for multiple-occurring or referenced elements and complex types.
- **Generate hierarchy relationships.** When you create hierarchical relationships, each reference to a component expands under its parent element. You can generate normalized or denormalized XML views in a hierarchy relationship.
  - **Normalized XML views.** When you generate a normalized XML view, elements and attributes appear once. Multiple-occurring elements, or elements in one-to-many relationships appear in different views related by keys.
  - **Denormalized XML views.** When you generate a denormalized XML view, all elements and attributes appear in one view. The Designer does not model many-to-many relationships between elements and attributes in an XML definition..
- **Create a custom XML views**. You can specify any global element as a root when creating a custom XML view. You can choose to reduce metadata explosion for elements, complex types, and inherited complex types.
- **Synchronize XML definitions**. You can update one or more XML definitions when their underlying schemas change.
- **Skip Create XML views.** When you choose to skip creating the XML views, you can define them later in the XML Editor. When you define views in the XML Editor you can define the views to match targets and simplify the mapping.
- **Create XML views for the elements and attributes in the XML file.** When you import an XML file with an associated schema, you can create XML views for just the elements and attributes in the XML file, instead of all the components in the schema.

If you choose to generate entity or hierarchy relationships, the Designer chooses a default root and creates the XML views. If the XML definition requires more than 400 views, a message appears that the definition is too large. You can manually create views in the XML Editor. Import the XML definition and choose to create custom views or skip generating XML views.

When you import a definition from a XML schema that has no global elements, the Designer cannot create a root view in the XML definition. The Designer displays a message that there is no global element.

After you create an XML view, you cannot change the configuration options you set for the view. For example, if you create a normalized XML view, you cannot change the view to denormalized. You must import a new XML source definition and select the denormalized option.

For information about XML sizing in PowerCenter, see "Using XML with PowerCenter Overview" on page 32. For more information about the limitations that apply to XML handling in PowerCenter, see "Limitations" on page 33.

To create a transformation with other element types, and to transform larger XML input files, use a Data Processor transformation. For more information about how to create Data Processor transformations, see the *Informatica Data Transformation User Guide* and the *Informatica Data Transformation Getting Started Guide*.

Importing Part of an XML Schema

When you import an XML schema, the Designer creates an XML definition that represents the entire schema. If you import an XML file with an associated schema, you can create XML views for just the elements and attributes in the XML file instead of all the components in the schema. The Designer creates a definition that is limited to the components in the XML file.

To import part of a schema, import an XML file that references the schema. Select the option to create XML views only for the elements and attributes in the XML file.

Rules and Guidelines for Importing Part of an XML Schema

Consider the following rules and guidelines to import a part of the schema:

- The Designer limits metadata to what is in the XML file. If you change the XML file and import the XML schema again, the XML definition changes.
- If an element or attribute occurs once in the XML file hierarchy, but occurs in more than one part of the schema hierarchy, the Designer includes all occurrences of the element or attribute in the XML definition.

  For example, a schema might have two address elements, a Store/Address and an Employee/Address. If you import an XML file with the schema, and the XML file has only Store/Address, the Designer creates all Address elements, including the Store/Address and the Employee/Address elements.
- If the XML file contains a derived complex type, the Designer includes all the base types in the XML definition as separate views. The Designer does not expand the derived type to include the base type in the same view.
- The Designer does not expand circular references in the XML definition when the XML file has multiple levels of circular references.

Generating Entity Relationships

You can generate an XML hierarchy as an entity relationship model. When you generate XML view entity relationships, the Designer completes the following actions:

- Generates views for multiple-occurring and referenced elements and complex types.
- Creates relationships between the views.

When the Designer generates entity relationships, the Designer generates different entities for complex types, global elements, and multiple-occurring elements based on the relationships in the schema.

If you want to create groups other than the default groups, or if you want to combine elements from different complex types, you can create custom XML views.

When you view an XML source definition in the XML Editor, you can see the relationships between each element in the XML hierarchy. For each relationship between views, the XML Editor generates links based on the type of relationship between the views.

To generate entity relationships:

1. In the Source Analyzer, click Sources > Import XML Definition.

   The XML Wizard opens.
2. Navigate to the source you want to import and click Open.
3. Enter a name for the file and click Next.

4. Select Entity Relationships and click Finish.

   The XML Wizard generates an XML definition that uses entity relationships.

Generating Hierarchy Relationships

When you create hierarchy relationships, each reference to a component is expanded under its parent element. The XML Wizard selects the default root and uses default settings to create XML groups.

To generate hierarchy relationships:

1. In the Source Analyzer, click Sources > Import XML Definition.

   The XML Wizard opens.
2. Navigate to the source you want to import and click Open.
3. Enter a name for the file and click Next.
4. Select Hierarchy Relationships.
5. Select Normalized XML Views or Denormalized XML Views and click Finish.

   The XML Wizard generates XML views based on hierarchy relationships.

Creating Custom XML Views

You can create custom XML views using the XML Wizard. When you create custom views, you can choose roots and specify how to generate metadata. You can choose to include or exclude global elements based on whether the root information applies to the data you intend to process. For example, if the schema contains information about stores and customers, you might want to create an XML definition that processes just the customers.

You can specify how you want to generate metadata associated with the view. You can reduce metadata explosion for elements, complex types, and inherited complex types by generating entity relationships. If you do not reduce the metadata references, the Designer generates hierarchy relationships and expands all child elements under their parent elements.

To create a custom view using the XML Wizard:

1. In the Source Analyzer, click Sources > Import XML Definition.

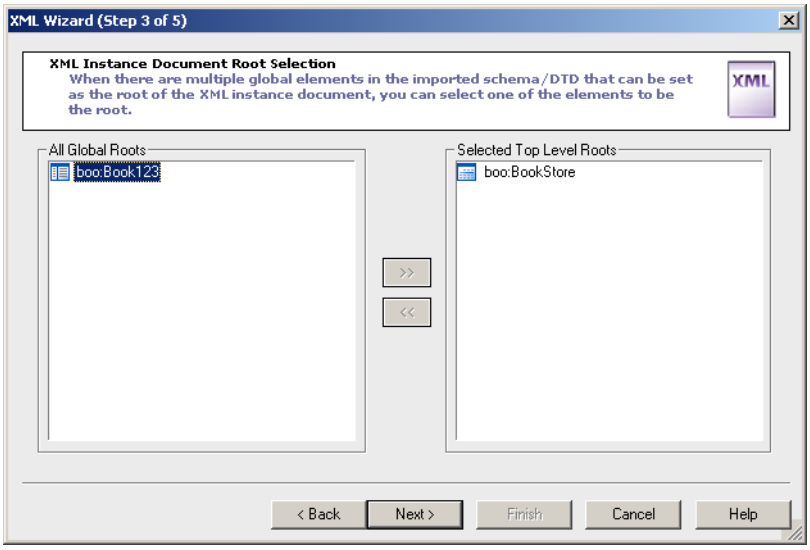
   The XML Wizard opens.
2. Navigate to the source you want to import and click Open.
3. Enter a name for the file and click Next.
4. Select Create Custom XML Views and click Next.

   Note: To manually create all the XML views in the XML Editor, select Skip Create XML Views. The XML Wizard creates the schema in the repository, but does not create the XML views.
5. Select root elements from the list of global root elements and click Next.
6. Choose to reduce metadata for elements, complex types, or inherited complex types and click Finish.

Selecting Root Elements

When you create a custom view, you can choose from the global elements in the imported schema to set the root for the XML instance document. Global elements are elements in an XML schema hierarchy that fall directly beneath the top root element.

The following figure shows the Root Selection page:



In this example, Bookstore element is selected as the root and the Book123 element is cleared as the root element.

Reducing Metadata Explosion

When the Designer creates an XML definition based on an XML schema that uses inheritance, the Designer can expand the metadata for each referenced element or group within the view that references the metadata. Or, the Designer can create a separate view for a referenced object and create relationships between the object and other views.

If you use references within an XML schema, you might want to reduce the number of times the Designer includes the metadata associated with a reference. The XML Wizard provides the following options to reduce metadata references:

- **Reduce element explosion**. The Designer creates a view for any multiple-occurring element or any element that is referenced by more than one other element. Each view can have multiple hierarchical relationships with other views in the definition.
- **Reduce complex type explosion**. The Designer creates an XML view for each referenced complex type or multiple-occurring element. The XML view can have multiple type relationships with other views. If the schema uses inherited complex types, you can also reduce explosion of inherited complex types.
- **Reduce complex type inheritance explosion**. For any inherited type, the XML Wizard creates a type relationship.

When you reduce metadata explosion, the Designer creates entity relationships between the XML views it generates.

Synchronizing XML Definitions

When you work with XML definitions, the files or sources you used to create the XML definition might change. For example, you might add a new element or complex type to an XSD file. You can synchronize an XML definition with any of the following repository definitions or files you used to create the XML definition:

- Relational source definitions
- Relational target definitions
- Flat files
- URLs
- XML files
- DTDs
- Schema files

When you synchronize an XML definition, the Designer updates the XML schema in the Schema Navigator. but it does not update the views in the XML definition. You can manually update the views columns in the XML Editor after you synchronize the XML definition.

Tip: Use schema files to synchronize XML definitions.

To synchronize XML source definitions:

1. In the Source Analyzer, click Sources > Import XML Definition.

   The XML Wizard opens.
2. Navigate to the repository definition or file that you used to create the XML definition, and click Open.
3. In Step 1 of the wizard, click Next. The wizard ignores any change you make to the name.
4. In Step 2 of the XML Wizard, choose to synchronize the XML definition and click Next.

   The XML Wizard skips to Step 5.
5. In Step 5 of the XML Wizard, choose the XML definition you want to synchronize.

   The XML Wizard synchronizes the source with the selected definition.

Use this method to synchronize XML target definitions. If you modify an XML source definition, you might also need to synchronize the target definition.

Note: Verify that you synchronize the XML definition with the source that you used to create the definition. If you synchronize an XML definition with a source that you did not use to create the definition, the Designer cannot synchronize the definitions and loses metadata. Click Edit > Revert to Saved to restore the XML definition.

Editing XML Source Definition Properties

After you import an XML source definition, you can edit source definition properties, such as the definition name. If you configure the session to read a file list, you can configure the mapping to write the source file name to each target row. You can also add metadata extensions.

To edit XML source definition properties:

1. Right-click the top of the definition in the Source Analyzer workspace. Select Edit.

2. On the Table tab, edit the following settings:

Table Settings	Description
Select Table	Displays the source definition you are editing.
Business Name	Descriptive name for the source definition. You can edit Business Name by clicking the Rename button.
Owner Name	Not applicable for XML files.
Description	Description of the source. Character limit is 2,000 bytes/K, where K is the maximum number of bytes for each character in the repository code page. Enter links to business documentation.
Database Type	Source or database type.
Code Page	Read Only. Not applicable for XML source files. The Integration Service converts all XML source files to Unicode.

3. Click the Columns tab.

   On the Columns tab, you can view information about the columns in the definition. To change column names or values, use the XML Editor.

   You can view the following information:

Columns Settings	Description
Select Table	Source definition you are editing.
Column Name	Name of the column.
Datatype	PowerCenter datatypes.
Precision	Length of the column.
Scale	Number of decimal positions in numerical data.
Not Null	Indicates whether the column can accept nulls.
Key Type	Primary key, foreign key, or not a key.
XPath	Path of the element referenced by the current column in the XML hierarchy. XPath does not display for generated primary or foreign keys.
Business Name	User-defined descriptive name for the column. If Business Name is not visible in the window, scroll to the right to view or modify the column.

4. If you configure the session to read a file list, and you want to write the source file name to each target row, click the Properties tab and select Add Currently Processed Flat File Name Port.

   The Designer adds the CurrentlyProcessedFileName port to the Columns tab. It is the last column in the first group. The Integration Service uses this port to return the source file name. The CurrentlyProcessedFileName port is a string port with default precision of 256 characters.

To remove the CurrentlyProcessedFileName port, click the Properties tab and clear Add Currently Processed Flat File Name Port.

5. Click the Metadata Extensions tab to create, edit, and delete user-defined metadata extensions.
6. Click OK.
7. Click Repository > Save to save changes to the repository.

Creating XML Definitions from Repository Definitions

You can import an XML source or target definition from relational or flat file definitions in the repository. When you import an XML definition from a repository definition, the XML Wizard creates an XML hierarchy from the relationships between the selected objects. The XML Wizard creates a root element for the hierarchy. You can choose a root from the groups you create. Or, you can create a separate root and relate the groups to it.

When you create an XML target definition, the XML Wizard generates keys to relate each group to the root.

To create an XML definition from repository sources or targets:

1. In the Source Analyzer, click Sources > Import XML Definition.

   -or-

   In the Warehouse Designer, click Targets > Import XML Definition.
2. In the XML Import dialog box, click Non-XML Sources or Non-XML Targets.
3. Select a definition from the list of sources or targets. Click the Arrow button to add the definition to the selected source list.

   You can select more than one input and more than one input type. If the definitions are related through primary and foreign keys, the XML Wizard uses the keys to relate groups when it generates the hierarchy.
4. To create a separate group for the root element, enter the Optional XML Rootname.

   The root name defaults to XRoot. The root group encloses all of the other groups. Remove the root name if you want to use one of the other groups as the root. The XML Wizard creates a group for each input source or target definition you select and generates a primary key in each group. The XML Wizard creates a foreign key in each group. The foreign key points to the root group link key.
5. Click Open.

   The XML Wizard appears.
6. Use the XML Wizard to generate the source or target groups.

Troubleshooting XML Sources

How can I put two multiple-occurring elements that both have the same parent element into one view? For example, I need to put all the elements of EMPLOYEE in one view:

```
<!ELEMENT EMPLOYEE (EID, EMAIL+, PHONE+)>
```

EMAIL and PHONE belong to the same parent element, but they do not belong to the same parent chain. You cannot put them in the same denormalized view. To put all the elements of employee in one view, you can pivot one of the multiple occurring elements.

Follow these steps to add two multiple-occurring elements to the same view:

1. Create an EMPLOYEE view.
2. Add the EID and EMAIL elements to the EMPLOYEE view.
3. Pivot the number of occurrences of EMAIL that you want to include in the view. Each EMAIL occurrence becomes a single occurring element in the view.
4. Add the PHONE element.

I have the following element definition in my DTD:

```
<!ELEMENT EMPLOYEE (EMPNO, SALARY)+>
```

How can I match the EMPNO and SALARY in the same view?

The DTD example is ambiguous. The definition is equivalent to the following:

```
<!ELEMENT EMPLOYEE (EMPNO+, SALARY+)>
```

In the DTD example, EMPLOYEE has multiple-occurring elements EMPNO and SALARY. You cannot have two multiple-occurring elements in the same view.

Use one of the following solutions:

- **Rewrite the element definition to make the definition unambiguous.**

  You might define the EMPLOYEE element as follows:

  ```
  <!ELEMENT EMPLOYEES (EMPLOYEE+)>
  <!ELEMENT EMPLOYEE (EMPNO, SALARY)>
  ```

  When you use this syntax, you define one EMPNO and one SALARY for each EMPLOYEE. The EMPLOYEE view contains both elements. Include EMPLOYEE as a multiple-occurring element in EMPLOYEES.

- **Leave the elements in separate views and use the source definition twice in a mapping.**

  When EMPNO and SALARY are in different views, you can still combine the data in a mapping. Use two instances of the same source definition and use a Joiner transformation.

I imported an XML file with the following structure:

```
<Bookstore>
                    <Book>Book Name</Book>
               <Book>Book Name</Book>
                                   <ISBN>051022906630</ISBN>
</Bookstore>
```

When I import this XML file, the Designer drops the ISBN element. Why does this happen? How can I get the Designer to include the ISBN element?

- **Use the schema to import the XML definition**. When you use an XML file to import an XML definition, the Designer reads the first element as simple content because the element has no child elements. The Designer discards the ISBN child element from the second Book instance. If you use a schema to import the definition, the Designer uses the schema definition to determine how to read XML data.
- **Verify the XML file accurately represents the associated schema**. If you use an XML file to import a source definition, verify the XML file is an accurate representation of the structure in the corresponding XML schema.

For information about XML sizing in PowerCenter, see "Using XML with PowerCenter Overview" on page 32. For information about the limitations that apply to XML handling in PowerCenter, see "Limitations" on page 33.

To create a transformation with other element types, and to transform larger XML input files, use a Data Processor transformation. For more information about how to create Data Processor transformations, see the *Informatica Data Transformation User Guide* and the *Informatica Data Transformation Getting Started Guide*.

CHAPTER 4

Using the XML Editor

This chapter includes the following topics:

Using the XML Editor Overview

When you import an XML definition in the Designer, you create an XML definition with default views, custom views, or no views. After you create an XML definition, you use the XML Editor to make changes to the definition.

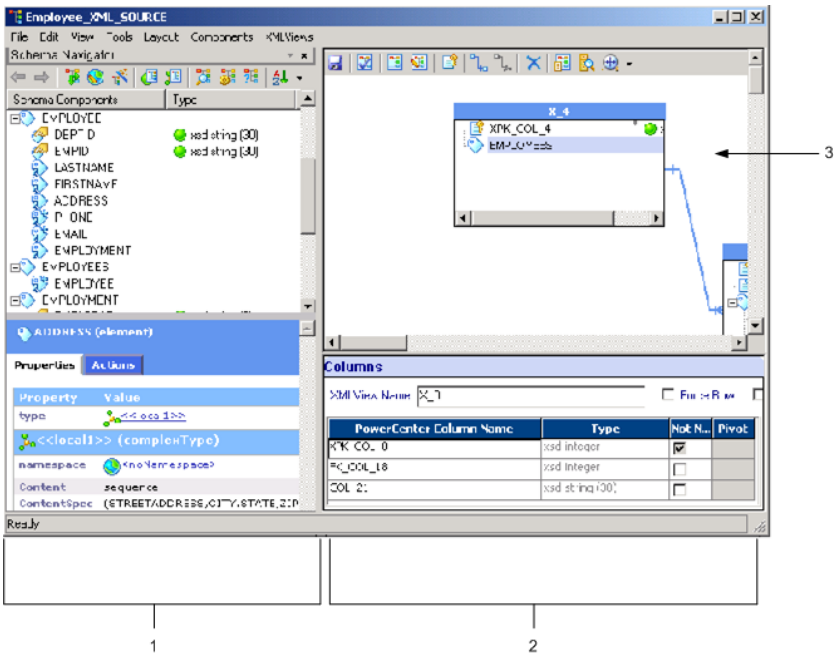
Use the XML Editor to create views, modify components, add columns, and maintain view relationships in the workspace. When you update an XML definition, the Designer propagates the changes to any mapping that includes the source. Some changes to XML definitions can invalidate mappings.

Note: If you make significant changes to the source you used to create an XML definition, you can synchronize the definition to the new source rather than editing the definition manually.

The XML Editor contains the following windows:

- Navigator
- XML Workspace
- Columns Window

The following figure shows the XML Editor:



1. Navigator
2. Columns Window
3. XML Workspace

The XML Editor uses icons to represent XML component types. To view a legend that describes the icons in the XML Editor, click View > Legend.

XML Navigator

The Navigator displays the schema in a hierarchical form and provides information about selected components. You can sort components in the Navigator by type, hierarchy, or namespace. You can also expand a component to see components below the component in the hierarchy.

The Navigator toolbar provides shortcuts to most of the Navigator functions. The toolbar also provides navigation arrows you can click to find previously displayed components in the hierarchy quickly.

The Navigator has the following tabs:

- **Properties tab.** Displays information about a selected component such as the component type, length, and occurrence.
- **Actions tab.** Provides a list of options to view more information about the selected component.

Properties Tab

The Properties tab displays information about a component you select in the Navigator. If the component is a complex element, you can view element properties in the schema, such as namespace, type, and content. When you view a simple element or attribute, the Properties tab shows the type and length of the element. The Properties tab also displays annotations.

If you import the definition from an XML file, you can edit the datatype and cardinality from the Properties tab. If you create the definition from a DTD file, you can edit the component type.

If a schema uses a namespace, you can change the namespace, prefix, and schema location for the namespace. The prefix identifies the elements or attributes that belong to a namespace. Elements and attributes in a default namespace have no prefix. You can select a namespace as a default namespace for an XML target.

Actions Tab

The Actions tab lists options you use to see more information about a selected component. You can also reverse changes you make to components from the Actions tab.

The following options appear on the Actions tab, depending on the properties of the component you select:

- **ComplexType references.** Displays the schema components that are of this type.
- **ComplexType hierarchy.** Displays the complex types derived from the selected component.
- **SimpleType reference.** Displays all the components that are this type.
- **Propagate SimpleType values.** Propagates the length and scale values to all the components of this SimpleType.
- **Element references.** Displays the components that reference the selected element.
- **Child components.** Displays the global schema components that the selected component uses.
- **Revert simpleType.** Changes the type, length, and precision values back to the original value if you have changed them.
- **XML view references.** Displays all the XML views and columns that reference the selected component.

XML Workspace

The XML workspace displays the XML views and the relationships between the views. You can create XML views in the workspace and define relationships between views.

The XML workspace toolbar provides shortcuts to most of the functions that you can do in the workspace.

You can modify the size of the XML workspace in the following ways:

- **Hide the Columns window.** Click View > Column Properties.
- **Hide the Navigator.** Click View > Navigator.
- **Reduce the workspace.** Click the Zoom button on the Workspace toolbar.

Columns Window

The Columns window displays the columns for a view in the workspace. Use the Columns window to name columns that you add. If you use pivoted columns, you use the Columns window to select and rename occurrences of multiple-occurring elements. You can also specify options, such as Not Null, Force Row, Hierarchy or Type Relationship Row, and Non-Recursive Row. These options affect how the Integration Service writes data to XML targets.

Creating and Editing Views

Use the XML Editor to create custom XML views or to edit XML views that you created with the XML Wizard. To create a view, you define the view and specify the columns in the view. If the schema has multiple-occurring elements, you can specify which element occurrences to include in the view. You can also create

special ports for XML target file names, and pass-through ports for XML Parser and XML Generator transformations.

Complete the following tasks to create and edit XML views:

- **Create an XML view.** Add a view to the workspace.
- **Add columns to a view.** Create new columns in a view.
- **Delete columns from a view.** Delete columns from a view.
- **Expand a complex type.** Choose a derived complex type to add to a view.
- **Import an anyType element.** Import anyType elements.
- **Apply content to an anyAttribute element.** Define content for anyAttribute elements.
- **Use an anySimpleType element.** Use an anySimpleType element in an XML definition.
- **Add a pass-through port.** Add a port to pass non-XML data through an XML transformation.
- **Add a FileName column.** Add a column to generate a new file name for each XML target file.

Creating an XML View

You can create views in the XML workspace. When you create an XML definition with no views, the XML Editor displays an empty workspace. You can create a view and add the columns and the view row to the view.

To create a new XML view in the workspace:

1. Open the XML definition in the XML Editor.
2. Click XML Views > Create XML View.

   The XML Editor creates a blank view in the workspace and displays empty columns in the Columns window.
3. Enter a name for the view in the Columns window.

   The name appears on the XML view in the workspace.
4. Highlight the node in the Schema Navigator from which you want to create a view.
5. Click Components > XPath Navigator.

   The XPath Navigator appears in the Navigator window.
6. Set the Column Mode in the XPath Navigator to View Row Mode to add the view row.
7. Select the element in the Navigator and drag the element to the view in the workspace.

   The XML Editor highlights the view row in blue.

   The first time you add a column to a view, the Designer verifies the column can be a view row. This occurs even if you do not specify to add a view row.
8. To change the view row to another column, right-click the appropriate row in the view and choose Set As View Row.

Adding Columns to a View

You can add columns to an XML view inside the XML workspace. To add a column, select the column from the XPath Navigator.

You can add a column to an XML view when the following conditions are true:

- The component path starts from the element in the schema that is the view root for that view.

- The component is not an enclosure element.
- The component does not violate normalization or pivoting rules. For example, you cannot add more than one multiple-occurring element to a view.
- You can add mixed content elements as either simple or complex types.
- Two views can share the same column, and a view might contain multiple identical columns.

To add columns to a view:

1. Select an XML view in the workspace.
2. Highlight a parent element in the Navigator that contains the components you want to add.
3. Click Components > XPath Navigator.

   The XPath Navigator appears in the Navigator window.
4. Click the Mode button and choose to add a column or a view row.
5. Choose Advanced to add a pivoted column to the view.

   A pivoted column is an occurrence of a multiple-occurring element. You can add single-occurring columns in Advanced Mode.

   Note: You cannot create pivoted columns in XML target definitions.
6. Drag a column from the XPath Navigator to the appropriate view in the XML workspace. You can select multiple columns at a time.

   The XML Editor validates the column you add. If the column is invalid for the view, a message appears in the status bar while you are dragging the column. New columns display as you add them to views.

Adding a Pivoted Column

A pivoted column in an XML definition is a multiple-occurring element that forms separate columns for the element occurrences in a view. You can create pivoted columns from any multiple-occurring element in an XML definition. You can also pivot elements in a view row.

If you add a pivoted column to a view, a default occurrence number appears in the Columns window. This number indicates which occurrence of the element to use in the column. You can change the occurrence number or add more occurrences of the element as new columns. If you do not rename the columns, the XML Editor adds a sequence number to each pivoted column name.

Note: You cannot change a pivot value if the pivot value is part of a view row.

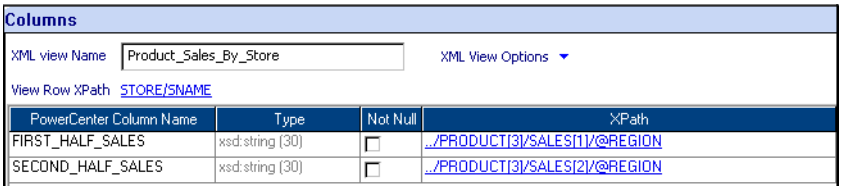
To add a pivoted column to a view:

1. Select the XML view in the workspace.
2. Highlight the element in the Navigator that you want to pivot.

   Or, highlight any element in the parent chain of the element that you want to pivot.
3. Click Components > XPath Navigator.
4. Select Advanced Mode.
5. Drag the column to pivot from the XPath Navigator to the view in the XML workspace.

   The Designer adds a default occurrence number in the Columns window. This number indicates which occurrence of the element to use in the column.

In the following figure, the view contains two occurrences of Sales for the third occurrence of Product:



The first Sales occurrence is in the column called First_Half_Sales. The second Sales occurrence is Second_Half_Sales. Region is an attribute.

6. Click the XPath link to change an element occurrence number.

   The Specify Query Predicate for XPath window appears.

7. Select the multiple-occurring element to edit.
8. Change the occurrence number in the Edit Pivot box and click OK.

Deleting Columns from a View

You can delete columns from a view.

To delete a column from a view:

1. Right-click the column in the view in the workspace.
2. Select Delete This Column.

   The XML Editor prompts you to confirm that you want to delete the column.

3. Click Yes to confirm.

   The XML Editor removes the column from the view. However, the column remains in the XPath Navigator hierarchy.

Deleting a Pivoted Column

To delete an occurrence of a pivoted column, select and delete the column from the Columns window.

To delete a pivoted column:

1. Right-click the column you want to delete in the Columns window.
2. Click Delete > Pivot.
3. Click Yes to confirm the delete.

Expanding a Complex Type

A schema can define a complex type that is a base type for more than one type. For example, a Publication can be a Magazine or a Newspaper. When you create the XML view, you can choose to use Publication as a Magazine or a Newspaper type.

When you view a complex type in the XPath Navigator, you can view the derived types.

To expand a complex type:

1. Highlight the complex type in the XPath Navigator.

   The Expand Complex Types list shows derived types.

2. Select the type you want to use.

   If you add the component to an XML definition, the definition contains the type you select.

Importing anyType Elements

You can import an XML schema that contains an anyType element. An element of type anyType can contain any datatype that occurs in an XML document. For example, the following section of an XML schema includes an element Document that is anyType:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="Publication" type="xsd:string"/>
   <xsd:element name="Date" type="xsd:string"/>
   <xsd:element name="Document" type="xsd:anyType"/>
```

When you import a schema with an anyType element, the anyType element appears in the Schema Navigator as anyType. The Designer does not create a port for an element of the type anyType.

You must change the anyType element to a global complex type in the Designer to use the anyType element in PowerCenter.

To change an anyType element to another global complex type:

1. Highlight the anyType element in the Schema Navigator.

   The anyType property appears in the Properties tab of the Schema Navigator.

2. Click the anyType property.

   If the schema does not contain a global complex type, the Designer displays an error that there are no global complex types to choose from.

3. Select a complex type and click OK.

Applying Content to anyAttribute or ANY Elements

You can import an XML schema that contains the anyAttribute, or ANY content element. To use the element in a view, you must define content for the element in the XML Editor. When you import a schema with an anyAttribute or ANY content element, the element appears in the Schema Navigator with no properties. The element does not appear in a view.

For example, the following schema element includes ANY content:

```
<xs:element name="person">
   <xs:complexType>
   <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
      <xs:any minOccurs="0"/>
   </xs:sequence>
   </xs:complexType>
</xs:element>
```

To apply content to anyAttribute or ANY content:

1. Click the element content link in the Schema Navigator.

   The Edit Any or anyAttribute Type Content dialog box appears.

2. Click Add Type.

   A new row appears.

3. Choose an element from the list of valid elements in the schema for the XML definition.

4. Select a cardinality and click OK.

   The substituted type appears in the Schema Navigator.

Using anySimpleType in the XML Editor

An anySimpleType element can contain any atomic datatype, such as string, integer, decimal, or date. Use anySimpleType in a schema when you do not know the datatype of the element in an XML instance document.

When you define an element as anySimpleType, the Designer creates an anySimpleType column for the element when you import the schema. When you use the column in a mapping, the XML Source Qualifier maps this type to a string.

Adding a Pass-Through Port

You can add a pass-through port to an XML Parser or XML Generator transformation to pass non-XML data through the transformation. When you define the port in the transformation, you add the port to the DataInput group in the XML Parser transformation or the DataOutput group in the XML Generator transformation.

Once you generate the pass-through port, you add another port to pass the data through the transformation. This port is the reference port. In an XML Parser transformation, the pass-through port passes the data into the transformation and the reference port passes the data out of the transformation. In an XML Generator, the pass-through port passes the data out of the transformation and the reference port passes data into the transformation.

If you have pass-through ports in an XML definition, you can determine the corresponding reference ports.

To determine the reference port for a pass-through port:

1. Right-click the pass-through port.
2. Select Navigate to > Referenced Column.

   The XML Editor highlights the referenced column in the workspace.

Adding a FileName Column

When you run a session, the Integration Service outputs a new target XML file each time a new root value occurs in the data. You can add a FileName column to an XML view to generate a unique file name for each XML file. The file name overrides the default output file name in the session properties.

When you use the FileName column, you set up an Expression transformation or other transformation in the mapping to generate the unique file names to pass to FileName column.

To create a FileName column in an XML view:

1. Right-click the view in the XML Editor.
2. Select Create FileName Column.

   The XML Editor creates a new string column in the view.
3. Change the column name in Columns window.
4. Exit the XML Editor.

   The column appears in the XML definition. The XPath is $Filename.

   Note: When you use XML FileName ports, you need to specify the directory name for the file.

Creating an XPath Query Predicate

You can filter XML data in a session by creating an XPath query predicate on a view row or column beneath the view row in an XML hierarchy. When you create an XPath query predicate, the XML Editor adds the XPath query predicate to the view row XPath.

When you create an XPath query predicate in the XML Editor, the XML Editor provides elements, attributes, operators, and functions to build the query. You can select the components, enter components, or copy components into a query. The XML Editor validates each query that you create.

You can query the value of an element or attribute, or you can verify that an element or attribute exists.

Querying the Value of an Element of Attribute

You can create an XPath query predicate to filter element or attribute values in a view. For example, to extract employees in department 100, create the following XPath query predicate:

```
EMPLOYEE [./DEPT = '100']
```

The query expression is in brackets. The XPath of Dept is abbreviated by "./" to indicate that the path is continuing from Employee.

The following XPath query predicate extracts employees if the last name is Smith:

```
EMPLOYEE[./NAME/LASTNAME='SMITH']
```

Name is a child element of Employee and is the parent of Lastname.

Use Boolean or numeric operators in an XPath query predicate. You can also use string, numeric, and boolean functions in a query.

Querying Mixed Content

An XML element has mixed content when it contains a value and it contains child elements. You can create an XPath query predicate to filter element values that are divided by child elements. However, the Integration Service does not evaluate predicates that occur after the first child element in mixed content.

For example, an XML file might contain a NAME element with mixed content:

```
<NAME>
  Kathy
    <MIDDLE> Mary </MIDDLE>
  Russell
</NAME>
```

Element NAME has the value "Kathy", a child element "MIDDLE", and a second value "Russell." The NAME column value is "KathyRussell." However, the Integration Service evaluates the NAME "Kathy."

The following query is false:

```
EMPLOYEE [./NAME = 'KathyRussell']
```

The following query is true:

```
EMPLOYEE [./NAME = 'Kathy']
```

Boolean Operators

Use the following Boolean operators in an XPath query predicate:

```
and or < <= > >= = !=
```

Use the following XPath query predicate to extract employees in department 100 with the last name Jones:

```
EMPLOYEE [./DEPT = '100' and ./ENAME/LASTNAME = 'JONES']
```

Numeric Operators

Use the following numeric operators in an XPath query predicate:

```
+ - * div mod
```

Use the following XPath query predicate to extract products when the price is greater than cost plus tax:

```
PRODUCT[./PRICE > ./COST + /.TAX]]
```

Functions

Use the following types of function in an XPath query predicate:

- **String.** Use string functions to test substring values, concatenate strings, or translate strings into other strings. The following XPath query predicate determines if an employee's full name is equal to the concatenation of last name and first name:

  ```
  EMPLOYEE[./FULLNAME=concat(./ENAME/LASTNAME,./ENAME/FIRSTNAME)]
  ```

- **Numeric.** Use numeric functions with element and attribute values. Numeric functions operate on numbers and return integers. For example, the following XPath query predicate rounds discount and tests if the result is greater than 15:

  ```
  ORDER_ITEMS[round(./DISCOUNT > 15]
  ```

- **Boolean.** Boolean functions return either true or false. Use them to test elements, check the language attribute, or force a true or false result. For example, a string is true if the string value is greater than zero:

  ```
  boolean(string)
  ```

Testing for Elements or Attributes

You can determine if an element or attribute occurs in the XML data. The following XPath query predicate determines if a department has a department name attribute:

```
COMPANY/DEPT[@DEPTNAME]/EMPLOYEE
```

Deptname is an attribute. Attributes are preceded by "@" in an XML query predicate expression.

When you run a session, the Integration Service extracts employee data from the XML source if the employee's department has a department name. Otherwise, the Integration Service does not extract the employee data.

XPath Query Predicate Rules and Guidelines

Use the following rules and guidelines when you create an XPath query predicate:

- You can configure an XPath query predicate for any element in a view row.

  For example, if a view row is Company/Dept, you can create the following XPath query predicate:

  ```
  COMPANY[./DEPT=100]
  ```

- You can match content.
- You can add an XPath query predicate to a column if the column occurs below the view row in the view XML hierarchy and the column XPath includes the view row.

  For example, if the view row is Product/Toys[1], you can create the following XPath query predicate:

  ```
  Product/Toys[1][./Sales > 100]
  ```

The following example shows an invalid XPath query predicate for the Product/Toys[1] view row:

```
Product/Toys[2][./Sales > 100]
```

Product/Toys[1] is the view row. You cannot use Product/Toys[2].

- Use a single-occurring element or attribute. You cannot create an XPath query predicate on a multiple-occurring element.
- You cannot create an XPath predicate query on an enclosure element because an enclosure element contains no values.

Steps for Creating an XPath Query Predicate

You can create an XPath query predicate for a view in an XML source definition.

To create an XPath query predicate:

1. Open an XML source definition in the XML Editor.
2. Select a view in the XML Editor workspace.

   The view columns appear in the Columns window.
3. Click View Row XPath.

   The Specify Query Predicate for XPath window appears. You can enter an XPath query predicate in the XPath Predicate window, or you can choose elements, operators, and functions from the tabs in the dialog box.
4. Click the Child Elements and Attributes tab to display the elements and attributes that you can add to an XPath query predicate.
5. Double-click an element or attribute to add it to the XPath query predicate.

   The component appears in the panel.
6. Click the Operators tab.

   Use operators to create expressions. You can compare elements to values or other elements, or you can create mathematical expressions.

   The following table describes the operators you can add to an XPath query predicate:

Operator	Description
`+`	Add
`-`	Subtract
`*`	Multiply
`div`	Divide
`mod`	Modulus
`and`	Boolean and
`or`	Boolean or

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
=	Equal
!=	Not equal

7. Double-click an operator to add the operator to the XPath query predicate.
8. To add the PowerCenter XPath query predicate functions, click the Functions tab. Functions accept arguments and return values.
9. Select another element or attribute from the Child Element and Attributes tab, or enter a value in the XPath Predicate window to complete an expression.
10. Click Validate to validate the XPath query predicate.

Maintaining View Relationships

You can complete the following tasks to maintain view relationships in the XML Editor:

- **Create a relationship between views**. Define relationships between views in the workspace.
- **Create a type relationship.** Define a type relationship between a column in a view and a type view in the workspace.
- **Re-create entity relationships.** Generate views and relationships using the same options as in the XML Wizard.

Creating a Relationship Between Views

Use the XML Editor to create hierarchical or inheritance relationships between views. You cannot create a relationship between an XML source and a non-XML source.

To create a relationship between XML views:

1. Right-click the top of the child XML view in the workspace.
2. Select Create Relationship.
3. Move the pointer to the parent view to establish a relationship.

   As you move the pointer, a link appears between the views, and the XML Editor verifies that the relationship is valid. If a relationship is not valid, an error message appears in the status bar.
4. If no error message appears, click the parent view to establish the relationship.

   The XML Editor creates the relationship and adds the appropriate foreign key.

5. To view details about the relationship, place the cursor over the link between the views.

   The Editor displays the relationship type and the primary and foreign keys.

Creating a Type Relationship

You can create a type relationship between a column in a view and a type view. If the column is pivoted, you can choose occurrences to include in the relationship.

To create a type relationship:

1. Right-click the column in the view you want to use.
2. Select Create Relationship.
3. If the column is pivoted, select an occurrence to use.
4. Move the pointer to the type view to establish a relationship.

   As you move the pointer, a link appears between the views, and the XML Editor verifies that the relationship is valid. If a relationship is not valid, an error message appears in the status bar.
5. If no error message appears, click the parent view to establish the relationship.

   The XML Editor creates the relationship.

Re-Creating Entity Relationships

You can re-create entity relationships for an XML definition. Use the Recreate Entity Relationships dialog box to generate new XML views using the same options as in the XML Wizard. When you regenerate views, you can choose to keep the existing views. The XML definition contains all the views.

To re-create entity relationships for an XML definition:

1. Open the XML definition in the XML Editor.
2. Highlight the XML root in the Navigator.

   If you highlight another component, the XML Editor uses that component as the root.
3. Click XML Views > Create Entity Relationship.
4. Choose from the following options to reduce metadata explosion:
   - **Reduce element explosion**. For any multiple-occurring or reference element, the XML Wizard creates one XML view with multiple hierarchical relationships.
   - **Reduce complex type explosion**. For any multiple-occurring or referenced complex type, the XML Wizard creates one XML view with multiple type relationships. If the schema uses inherited complex types, you can also reduce explosion of inherited complex types.
   - **Reduce complex type inheritance explosion**. For any inherited type, the XML Wizard creates one XML view using multiple type relationships.
   - **Share existing XML views.** Do not remove existing XML views.
   - **Refresh shared XML views.** Save existing views but update them.
5. Click Next.

   The Recreate Entity Relationships dialog box appears.
6. To display a child component, select a shared element or complex type and click the name.
7. To exclude a child component, clear the element in the Exclude Child Components pane.

   To generate a new view, select the element or complex type. When you create the new entity relationships, you generate a view with that element as a view root.

Viewing Schema Components

Complete the following tasks to view components in the Navigator and workspace:

- **Update the namespace**. Change the location of a schema or the default namespace in an XML target.
- **Navigate to components.** Find components by navigating from a component to another component or area of the XML Editor window.
- **Arrange views in the workspace.** Arrange the views in the workspace hierarchically. You can organize the views into a hierarchical arrangement in the workspace. To arrange views in the workspace, click Layout > Arrange, or right-click the workspace and select Arrange.
- **Search for components.** Find components in the Navigator or in the workspace.
- **Display the hierarchy of simple or complex types.** View a hierarchy of the simple or complex types in the XML schema.
- **View XML metadata**. View an XML file, schema, or DTD that the XML Editor creates from the XML definition.
- **Preview XML data**. Display an XML view using sample data from an external XML file.
- **Validate the XML definition.** Validate the XML definition and view errors.

Updating a Namespace

When you create an XML definition, you can change the namespace prefix and schema location in the XML Editor. You can also add a schema to the namespace.

If you create a target XML definition that has one or more namespaces, you can choose a default namespace. When you run a session, the Integration Service writes the elements and attributes from the default namespace without a namespace prefix.

Do not use “xml” or “xmlns” as a namespace prefix. An “xml” prefix points to the `http://www.w3.org/XML` namespace by default. “Xmlns” links elements to namespaces in an XML schema.

Note: You cannot add a namespace using the XML Editor.

To update a namespace:

1. Select an element in the Navigator.
2. Click the Properties tab.
3. Click the Namespace link.

   The Edit Namespace Prefix and Schema Location dialog box appears.
4. To change the prefix or schema location, select the text you want to change and enter the new value.
5. To add more than one schema to a namespace, select a schema location and click Add.

   A blank schema appears in the namespace Schema Location.
6. To delete a schema, highlight the schema location and click Delete.
7. To create a default namespace in the XML target, select a namespace.

   All the components from the namespace you select belong to the default namespace in the XML target. When you run a session, the Integration Service does not write the default namespace prefix in the XML target file.

Navigating to Components

To quickly find components, in large XML definitions select a workspace component to navigate from and select a navigation option. For example, if you click a foreign key in a view, you can navigate to the associated primary key or to the column in the Columns window. You can navigate between components in the workspace, the Columns window, and the Navigator.

To navigate to components:

1. Right-click a component in the workspace or in the Columns window.
2. Select Navigate to.
3. Select an available option.

   You can select from the following options, depending on the component you select to navigate from:

   - **Schema component.** Highlights the component in the Navigator.
   - **PowerCenter column.** Highlights the column in the Columns window.
   - **Primary key**. Highlights the primary key associated with a selected foreign key.
   - **Referenced column.** Highlights the referenced column associated with a pass-through port in an XML Parser or Generator transformation.
   - **XPath Navigator.** Displays the path to the selected the component.
   - **XML view.** Highlights a view in the workspace that contains the selected column from the Columns window.

Searching for Components

You can search for components in an XML definition. The XML Editor displays all occurrences of the components. You can click a search result and the XML Editor highlights the component in the schema or view. You can search the XML schema or the XML views in the workspace.

Searching the XML Schema

You can search for components by name, type, and namespace. You can specify properties to search for, such as an annotation, a fixed or default value, or a length. You can search for legal values using the enumeration property.

To search for components in the schema:

1. Click Edit > Search In Schema.

   The Search Components dialog box appears.
2. To search by component properties, click the Advanced Options link to view the Properties you can search for.
3. Enter a name, type, or property to search.
4. If you want to search in a specific namespace, click All, and select a namespace from the list.
5. Click Search.

   The search results appear in the dialog box.
6. Click a search result to view the component in the Properties window.

Searching in XML Views

You can search for components and columns in the XML views. If you search by component name, you can find all the associated columns in the definition. For example, if you search for a component “Number,” the results contain the views and columns that contain the component “Number.”

To search using a partial key, enter the first few characters of the column or component name.

To search for components in XML views:

1. Click Edit > Search XML Views.

   The Search XML Views and Columns dialog box appears.
2. Enter search criteria.

   You can search for all column types, regular column types, generated keys, or other types. Other types include FileName columns, reference ports, and pass-through fields.
3. Click Search.

   The search results appear in the dialog box.
4. To clear search fields, click New Search.

Viewing a Simple or Complex Type Hierarchy

You can view a hierarchy of the XML simple types in the schema definition. To view a hierarchy of simple types, click View > SimpleType Hierarchy. A window displays a hierarchy of the simple types.

You can display a hierarchy of the complex types in the schema definition. To view a hierarchy of complex types, click View > ComplexType Hierarchy. A window displays a hierarchy of the complex types in the schema. Select a component from the ComplexType Hierarchy window to navigate to the component in the schema.

Viewing XML Metadata

You can view an XML definition as an XML, DTD, or XML schema file.

To view XML metadata:

1. To view the metadata as a sample XML document, choose a global component in the Navigator.
2. Click View > XML Metadata.

   The View XML Metadata dialog box appears.
3. Choose to display the XML definition as an XML file, a DTD file, or an XML schema.

   If you use multiple namespaces, choose the namespace to use.

   A default application or text editor displays the metadata.
4. To save a copy of the XML, DTD, or XML schema file, click Save As.
5. Enter a new file name.

   If the default display application is a text editor, you need to include the appropriate file suffix with the file name. The suffix is .xml, .dtd, or .xsd, depending on what type of file you are working with.

Validating XML Definitions

You can validate an XML definition in the XML Editor.

To validate an XML definition:

1. Open the definition in the XML Editor.
2. Click inside the workspace.
3. Click XML Views > Validate XML Definition.

   The Validate window displays the results.

Setting XML View Options

By default, the Integration Service generates rows for each view that has data in the view row. To change how the Integration Service generates rows for some XML source definition views, select XML View Options in the XML Columns window.

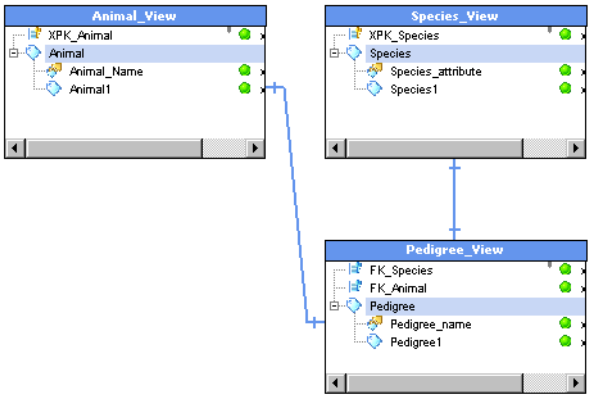
Use the following methods to determine when the Integration Service generates rows from an XML source:

- **Generate all the foreign keys in a view.** By default, the Integration Service generates values for one foreign key in a view. If a view has more than one foreign key, the other foreign keys have null values. You can generate values for all the foreign keys.
- **Stop recursive reads in a circular relationship**. By default, the Integration Service generates rows for all occurrences of data in a circular relationship. You can generate a row for just the first occurrence of recursive data.
- **Generate a row for a child view if a parent exists.** By default, the Integration Service creates rows for all views with data in the view row. You can generate a row for a child view only when a parent view has data.
- **Generate a row for a view without view row data.** By default, the Integration Service generates data for a view when the view row has data. You can generate a row for a view that has no data in the view row.
- **Generate rows for specific complex types in a type relationship.** By default, the Integration Service generates rows for all views in the XML definition. You can generate rows for specific views in type relationships.

Generating All Hierarchy Foreign Keys

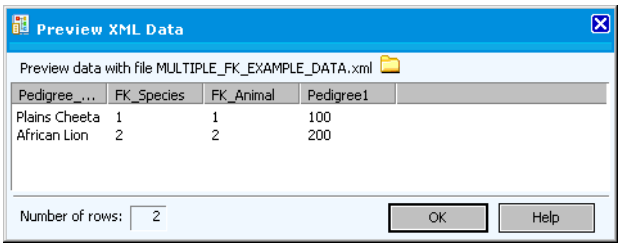
By default, the Integration Service generates values for one foreign key in a view. If a view has more than one foreign key, the other foreign keys have null values. You can choose to generate key values for all foreign keys in a view. Select the All Hierarchy Foreign Keys option.

The following figure shows the Pedigree_View with two foreign keys, FK_Species and FK_Animal:



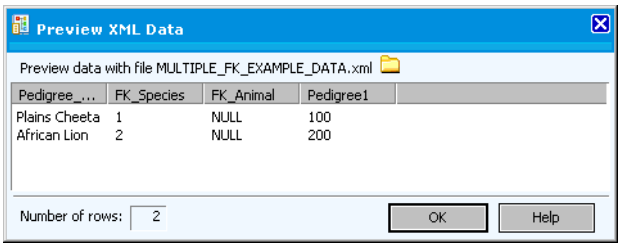
If you select the All Hierarchy Foreign Keys option for the Pedigree_View, the Integration Service generates key values for FK_Species and FK_Animal.

The following figure shows sample data for the Pedigree_View with the All Hierarchy Foreign Keys option:



If you clear the All Hierarchy Foreign Keys option, the Integration Service generates key values for one foreign key column. In this example, the Integration Service generates values for FK_Species because Species_View is the closest parent of Pedigree_View in the XML hierarchy. The FK_Animal foreign key has null values.

The following figure shows sample data for the Pedigree_View if you clear the All Hierarchy Foreign Key option:



The Integration Service generates foreign keys to the closest parent.

Generating Rows in Circular Relationships

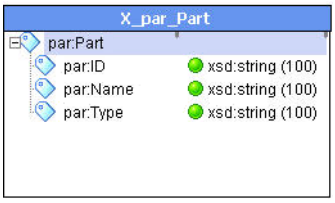
By default, the Integration Service generates rows for all occurrences of data in a circular relationship. You can choose to create a row for only the first occurrence. Select the Non-Recursive row option.

The following XML file contains a Part element with a circular reference:

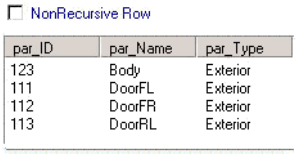
```
<?xml version="1.0" encoding="utf-8"?>
<Vehicle xmlns="http://www.PartInvoice.org"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.PartInvoice.org part.xsd">
<VehInfo>
      <make>Honda</make>
      <model>Civic</model>
      <type>Compact</type>
</VehInfo>
<Part>
  <ID>123</ID>
  <Name>Body</Name>
  <Type>Exterior</Type>
  <Part>
    <ID>111</ID>
    <Name>DoorFL</Name>
        <Type>Exterior</Type>
        <Part>
    <ID>1112</ID>
    <Name>KnobL</Name>
    <Type>Exterior</Type>
  </Part>
  <Part>
    <ID>1113</ID>
    <Name>Window</Name>
```

```
      <Type>Exterior</Type>
    </Part>
  </Part>
</Part>
</Vehicle>
```

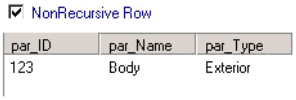
The following figure shows that the Part element is the view row for the X_par_Part view in the XML definition:



The following figure shows that the Integration Services generates rows for Part 123 and all of the component parts when you run a session:



The following figure shows that the Integration Service reads the first occurrence of the Part element and generates one row of data for Part 123 when you select NonRecursive Row:



Generating Hierarchy Relationship Rows

By default, the Integration Service creates rows for all views with data in the view row. Select Hierarchy Relationship Row to generate a row for a child view if the parent view has corresponding data in a hierarchy relationship. The parent view must have data to generate a row for the child view.

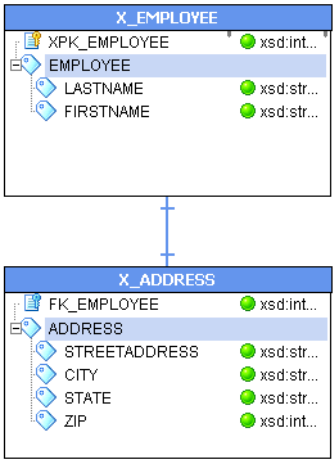
For example, an XML definition might have a hierarchy consisting of an Employee view and an Address view. Employee is the parent view. The address data can include Employee\Addresses or Store\Addresses. You can choose to output Employee\Address.

The following XML file has an Address within the Store element and an Address within the Employee element:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE STORE >
<STORE SID="BE1752">
  <SNAME>Mud and Sawdust Furniture Store</SNAME>
  <ADDRESS>
    <STREETADDRESS>335 Westshore Road</STREETADDRESS>
    <CITY>Fausta City</CITY>
    <STATE>CA</STATE>
    <ZIP>97584</ZIP>
  </ADDRESS>
  <EMPLOYEE DEPID="34">
    <ENAME>
      <LASTNAME>Bacon</LASTNAME>
      <FIRSTNAME>Allyn</FIRSTNAME>
    </ENAME>
  <ADDRESS>
    <STREETADDRESS>1000 Seaport Blvd</STREETADDRESS>
```

```
      <CITY>Redwood City</CITY>
      <STATE>CA</STATE>
      <ZIP>94063</ZIP>
    </ADDRESS>
   <EPHONE>(408)226-7415</EPHONE>
   <EPHONE>(650)687-6831</EPHONE>
  </EMPLOYEE>
  </STORE>
```

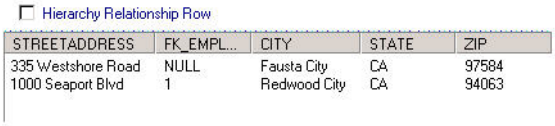
The following figure shows a hierarchical relationship between the Employee view and the Address view:



The Employee view is connected to the Address view with a blue line that defines a one-to-one relationship between the parent and the child view. The Employee view has a primary key, XPK_Employee, and an Employee element consisting of LastName and FirstName. The Address view has a foreign key, FK_Employee, and an Address element that consists of StreetAddress, City, State, and Zip.

By default, the Integration Service generates a row for each occurrence of the Address element. The Integration Service generates one row for the Store\Address and another for Employee\Address.

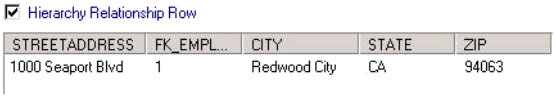
The following figure shows the Address XML data if you clear the Hierarchy Relationship Row option:



When you select the Hierarchy Relationship Row option, the Integration Service generates rows in a session as follows:

- The Integration Service generates a row for the Address view when the Employee view has corresponding data in a session.
- The Integration Service generates a row representing the Employee\Address hierarchy relationship.
- The Integration Service does not generate a row for Store\Address.

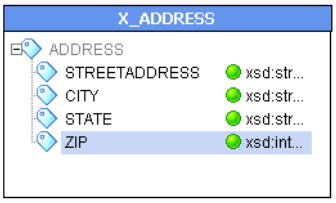
The following shows the Address data if you select the Hierarchy Relationship Row option:



Setting the Force Row Option

By default, the Integration Service generates data for a view when the view row has data. Select Force Row to generate a row for the XML view whether or not the view row element contains data. For example, if a view row is address\zip, you can choose to output address, even if the view row has no zip data.

The following figure shows the zip element as the view row:



The zip element is highlighted at the bottom of the list, with the street, city and state element rows listed above it.

By default, the Integration Service generates a row for every occurrence of the zip element within the address element.

For example, you might process the following XML file in a session:

```
<?xml version="1.0" ?>
<company xmlns="http://www.example.org"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org forcerow.xsd" >
<name>company1</name>
  <address>
    <street>stree1</street>
    <city>city1</city>
    <zip>1001</zip>
  </address>
<employee>
  <name>emp1</name>
  <address>
    <street>emp1_street</street>
    <city>emp1_city</city>
    </address>
</employee>
<employee>
  <name>emp2</name>
  <address>
    <street>emp2_street</street>
    <city>emp2_city</city>
    <zip>2001</zip>
  </address>
</employee>
</company>
```

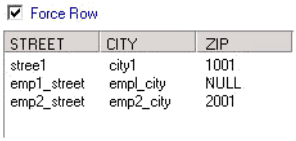
By default, the Integration Service generates the stree1 and emp2_street rows for the Address view.

The following figure shows the stree1 and emp2_street rows for the Address view:

The Integration Service does not generate a row for emp1, because the view row is zip, and emp1 has no data for the zip element.

If you enable Force Row, you can output the street and city elements with or without the zip. The session generates a row for emp1, even though emp1 does not have data for the zip element.

The following figure shows the rows that the Integration Service generates when you enable Force Row:



Generating Rows for Views in Type Relationships

By default, the Integration Service generates rows for all views in a type relationship. Choose the Type Relationship Row option to generate rows for a specific complex type in a type relationship. Set the Type Relationship Row for each view that you want to output.

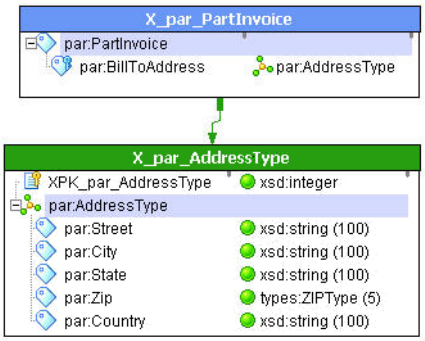
For example, a definition might have a hierarchy that includes BillToAddress and ShipToAddress. If you want to generate rows for the BillToAddress, use the Type Relationship Row option.

Defining a Type Relationship

The following schema defines BillToAddress and ShipToAddress. BillToAddress is an AddressType, and ShipToAddress is a USAddressType. USAddressType extends AddressType.

```
<xsd:sequence>
  <xsd:element name="Street" type="xsd:string" />
  <xsd:element name= "City" type="xsd:string" />
  <xsd:element name="State" type="xsd:string" />
  <xsd:element name="Zip" type="types:ZIPType" />
  <xsd:element name="Country" type="xsd:string" />
</xsd:sequence>
</xsd:complexType>
  <xsd:complexType name="USAddressType">
    <xsd:complexContent>
    <xsd:extension base="AddressType">
      <xsd:sequence>
      <xsd:element name="PostalCode" type="xsd:string" minOccurs="0"/>
      <xsd:sequence>
    </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:element name="BillToAddress" type="AddressType" />
<xsd:element name="ShipToAddress" type="USAddressType" />
```

The following figure shows a type relationship in the XML definition with the PartInvoice view showing the BillToAddress element in the X_par_PartInvoice XML view and the related X-par_AddressType XML view below:



The PartInvoice view contains invoice data. The view includes a BillToAddress. The type relationship in the XML definition is between BillToAddress and AddressType.

To limit the AddressType data to BillToAddress, select the X_par_PartInvoice view in the XML Editor workspace. Choose the Type Relationship Row option. When you run a session, the Integration Service generates Address rows for BillToAddress but not ShipToAddress. ShipToAddress is not in the type relationship.

Example

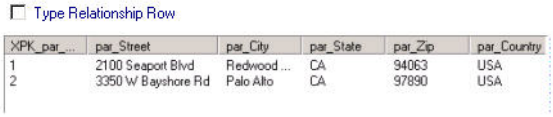
The following example shows how to limit data to specific types in a type relationship. The example uses the PartInvoice view and the AddressType view.

The following XML file contains invoice data that includes the BillToAddress and ShipToAddress:

```
<xsd:complexType name="AddressType">
<?xml version="1.0" encoding="utf-8"?>
<Invoices xmlns="http://www.PartInvoice.org"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.PartInvoice.org Part.xsd">
<PartInvoice InvoiceNum="185" DateShipped="2005-01-01">
    <PartOrder>
          <PartID>HLG100</PartID>
          <PartName>Halogen Bulb</PartName>
          <Quantity>2</Quantity>
          <UnitPrice>35</UnitPrice>
    </PartOrder>
        <BillToAddress>
              <Street>2100 Seaport Blvd</Street>
              <City>Redwood City</City>
              <State>CA</State>
              <Zip>94063</Zip>
              <Country>USA</Country>
        </BillToAddress>
        <ShipToAddress xsi:type="USAddressType">
              <Street>3350 W Bayshore Rd</Street>
              <City>Palo Alto</City>
              <State>CA</State>
              <Zip>97890</Zip>
              <Country>USA</Country>
              <PostalCode>97890</PostalCode>
        </ShipToAddress>
    </PartInvoice>
</Invoices>
```

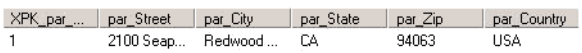
When you run a session, the Integration Service generates an X_par_AddressType row for each AddressType.

The following figure shows the BillToAddress and ShipToAddress rows that the Integration Service generates by default:



To generate AddressType rows related to the PartInvoice view, set the Type Relationship Row Option for the PartInvoice view.

The following figure shows the BillToAddress row that the Type Relationship Row Option generates:



It does not generate a row for ShipToAddress because ShipToAddress is not in the type relationship.

Troubleshooting Working with the XML Editor

When I validate an XML definition, I get an error that says the XML definition is too large. Why does this occur?

If you import an XML file that has components defined with infinite lengths, you can easily exceed the 500 MB limit for total column length. You can change the column lengths in the XML Editor, or you can set an option to override all infinite lengths and reimport the file.

I cannot find the DTD or XML schema file that I created when I viewed XML metadata.

The DTD or XML schema file that you can view is a temporary file that the Designer creates for viewing. If you want to use the file for other purposes, save the file with another name and directory when you view it.

When I add columns to XML source views, the hierarchy in the source XML file remains the same.

When you add columns to XML source views, you do not add elements to the underlying hierarchy. The XML hierarchy that you import remains the same no matter how you create the views or how you map the columns in a view to the elements in the hierarchy. You can modify the datatypes and the cardinality of the elements, but you cannot modify the structure of the hierarchy.

For information about XML sizing in PowerCenter, see "Using XML with PowerCenter Overview" on page 32. For more information about the limitations that apply to XML handling in PowerCenter, see "Limitations" on page 33.

To create a transformation with other element types, and to transform larger XML input files, use a Data Processor transformation. For more information about how to create Data Processor transformations, see the *Informatica Data Transformation User Guide* and the *Informatica Data Transformation Getting Started Guide*.

CHAPTER 5

Working with XML Targets

This chapter includes the following topics:

Working with XML Targets Overview

You can create XML target definitions in the following ways:

- **Import the definition from an XML schema or file.** You can create a target definition from an XML, DTD, or XML schema file. You can import XML file definitions from a URL or a local node. If you import an XML file with an associated DTD, the XML Wizard uses the DTD to generate the XML file.
- **Create an XML target definition based on an XML source definition.** You can drag an existing XML source definition into the Target Designer. If you create an XML target definition, the Designer creates a target definition based on the hierarchy of the XML definition.
- **Create an XML target based on a relational file definition**. You can import an XML target definition from a relational or flat file repository definition.

In addition to creating XML target definitions, you can complete the following tasks with XML targets in the Target Designer:

- **Edit target properties.** Edit an XML target definition to add comments documenting changes to target XML, DTD, or XML schema files.
- **Synchronize target definitions**. You can synchronize the target XML definition to a schema if you need to make changes. When you synchronize the definition, you update the XML definition instead of importing the schema if the schema changes.

Importing an XML Target Definition from an XML File

You can import XML definitions from an XML schema, DTD file, or XML file. You can import local files or files that you reference with a URL. To ensure that the Designer can provide an accurate definition of the data, import a target definition from an XML schema.

You can choose from the following options to create XML views:

- **Create entity relationships**. Use this option to create views for multiple-occurring or referenced elements and complex types. You create relationships between views instead of creating one large hierarchy.
- **Create hierarchical relationships.** Use this option to create a root and expand the XML components under the root. You can choose to create normalized or denormalized views. If you choose normalized, every element or attribute appears once. One-to-many relationships become separate XML views with keys to relate the views. If you create denormalized XML views, all elements and attributes display in one hierarchical group.
- **Create custom XML views.** Use this option to select multiple global elements as roots for XML views and select options for reducing metadata explosion.
- **Skip creating XML views**. Use this option to import the definition without creating any views. If you choose this option, use the XML Editor to create XML views in the workspace at a later time.
- **Synchronize XML definitions.** Use this option to update one or more XML definitions when their underlying schemas change.

  Tip: Import DTD or XML schema files instead of XML files. If you import an XML file with an associated DTD, the XML Wizard uses the DTD.

To import XML target definitions:

1. In the Target Designer, click Targets > Import XML Definition.

   The Import XML Definitions window opens. The local folder schema files appear by default.
2. Click Local File or URL to browse for XML files.
3. To browse for DTD or XML files, select the appropriate file extension from the Files of Type list.

Creating a Target from an XML Source Definition

When you want to create a target definition that closely resembles an existing source definition, you use the source definition or a shortcut to the source definition to create the target definition. Drag the XML source definition into the Target Designer to create an XML target definition or a relational target definition.

Use the following guidelines to create XML target definitions:

- When you create an XML target definition from an XML source definition, you create a duplicate of the XML source definition.
- A valid XML source definition does not necessarily create a valid XML target definition. To ensure that you create a valid target definition, validate the target definition.

Note: XML target definitions cannot contain pivoted columns.

Use the following guidelines to create relational target definitions:

- If you create a relational target definition, the Designer creates the relational target definitions based on the groups in the XML source definition. Each group in the XML source definition becomes a target definition.
- The Designer creates the same relationship between the target definitions as between the groups in the source definition.

To create a target definition from an XML source definition:

1. Drag an XML source definition from the Navigator into the Target Designer workspace.

   The XML Export dialog box appears.
2. Select to create a relational or XML target. Click OK.

   The target definition appears in the Target Designer workspace. If you select relational targets, more than one target definition might appear in the workspace, depending on the source.

Editing XML Target Definition Properties

After you create an XML target definition, you can edit the properties to reflect changes in the target data, add business names and comments, or change the code page.

To edit XML target definition properties:

1. Open the XML target in the Target Designer.
2. Right-click and select Edit.
3. On the Table tab, edit the settings.

   The following table describes the Table settings:

Table Settings	Description
Select Table	Name of the target definition. To change the name, click the Rename button.
Business Name	Descriptive name for the target table. Edit the Business Name using the Rename button.
Constraints	Not applicable to XML targets. Any entry is ignored.
Creation Options	Not applicable to XML targets. Any entry is ignored.
Description	Description of target table. Character limit is 2,000 bytes/K, where K is the maximum number of bytes for each character in the repository code page. Enter links to business documentation.
Code Page	Select the code page to use in the target definition.
Database Type	Indicates that the target definition is an XML target.
Keywords	Use keywords to organize and identify targets. Keywords might include developer names, mappings, or XML schema names. Use keywords to perform searches in the Repository Manager.

4. On the Columns tab, you can view XML column definitions.

   The following table describes the Columns settings:

Columns Settings	Description
Select Table	Displays the target definition you are editing. To choose a different definition to edit, select one from the list of definitions you have available in the workspace.
Column Name	Name of the column.
Datatype	PowerCenter datatype for the column.
Precision	Size of column. You can change precision for some datatypes, such as string.
Scale	Maximum number of digits after the decimal point for numeric values.
Not Null	Indicates if the column can have null values.
Key Type	Type of key the XML Wizard generates to link the views.
XPath	Path through the XML file hierarchy that locates an item.

5. On the Properties tab, you can modify the transformation attributes of the target definition.

   If you use a source-based commit session or Transaction Control transformation with the XML target, you can define how you want to flush data to the target.

   The following table describes the attributes that you can edit:

Columns Settings	Description
Select Table	Displays the source definition you are editing. To choose a different source definition to edit, select the source definition from the list.
Duplicate Group Row Handling	Choose one of these options to handle processing duplicate rows in the target: - First Row. The Integration Service passes the first duplicate row to the target. Rows following with the same primary key are rejected. - Last Row. The Integration Service passes the last duplicate row to the target. - Error. The Integration Service passes the first row to the target. Rows with duplicate primary keys increment the error count. The session fails when the error count reaches the error threshold.
DTD Reference	DTD or XML schema file name for the target XML file. The Integration Service adds the document type declaration to the XML file when you create the XML file.
On Commit	The Integration Service can generate multiple XML files or append to one XML file after a commit. Use one of the following options: - Ignore Commit. The Integration Service creates an XML file and writes to the XML file at end of file. - Create New Document. Creates a new XML file at each commit. - Append to Document. Writes to the same XML file after each commit.

Columns Settings	Description
Cache Directory	Directory for the XML target cache files. Default is the $PMCacheDir service process variable.
Cache Size	Total size in bytes for the XML target cache. Default is auto.

6. On the Metadata Extensions tab, you can create, modify, delete, and promote non-reusable metadata extensions, and update their values. You can also update the values of reusable metadata extensions.
7. Click OK.
8. Click Repository > Save.

Validating XML Targets

You can create customized XML views that describe how to extract data to an XML file. However, not all view structures or relationships between views are valid in an XML definition. Some view structures might be valid for an XML source, but not for an XML target. The Designer prevents you from creating ambiguous definitions.

PowerCenter validates target XML views when you perform the following tasks:

- The Designer does limited validation when you save or fetch an XML target from the repository.
- The XML Editor validates each step when you edit XML in the XML workspace.
- You can choose to validate a target definition that you are editing in the XML Editor.
- The Designer validates XML target connections when the Designer validates mappings.

The Designer uses rules to validate hierarchy relationships, type relationships, and inheritance relationships.

Note: The Integration Service does not validate the target XML instance against a schema in a session. You can set the Validate Target session property to validate simple types in the data.

Hierarchy Relationship Validation

The Designer uses the following rules to validate hierarchy relationships:

- A view that has a root at a type cannot be a standalone view. The view must be a child in an inheritance relationship or the view must have a type relationship with another view. An XML target is invalid if the XML target has no views that are rooted at an element.
- You must connect a view with a multiple-occurring view row to another view.
- Two views cannot have the same effective view row.
- An XML target is invalid if the XML target has no view root at an element.
- You can separate parent and child views by other elements, but if you have a choice of two parents for a view, you must use the closest one. Determine the closest parent by the path of the effective view row. One parent comes before the other in the path. Choose the view that comes second in the path.
- You must connect all views with the same view root in the same hierarchy. The definition cannot contain multiple trees for the same view root.
- An XML view can have a hierarchical relationship to itself if the view row and the view root are identical for the view.

Type Relationship Validation

A type relationship is a relationship between a column and a view. A type relationship is not a relationship between two views. The following rules apply to type relationships:

- A column in a view, V1, can have a type relationship to a view, V2, if the view roots are the same type, or the V2 view root type is derived from the V1 view root. Both view roots must be global complex types.
- If a column in a view has a type relationship to another view, you cannot expand the column.

Inheritance Validation

You can create two types of inheritance relationship with XML views:

- **View-to-view inheritance.** A view is a derived type of another view. Both views must have global complex view roots.

  A view can have an inheritance relationship to another view if its view root is a complex type derived from the view root type of the other view.

  A view can be a parent in multiple inheritance relationships, but a view can be a child in just one inheritance relationship.
- **Column-to-view inheritance.** The column is an element of a local complex type, Type1, and the view is rooted at a global complex type, Type2. Type1 is derived from Type2.

  A column in a view can have an inheritance relationship to another view if the column is a local complex type and the type is derived from the view root type of the other view.

  If a column in a view, V1, has an inheritance relationship to a view, V2, you cannot put the content of V2 into view V1.

Using an XML Target in a Mapping

When you add an XML target to a mapping, you need to following mapping guidelines for multigroup transformations.

The following components affect how you map an XML target in a mapping:

- Active sources
- Root elements
- Target port connections
- Abstract elements
- Transaction control points
- FileName columns

Active Sources

An active source is a transformation that can return a different number of rows for each input row. The Integration Service can load data from different active sources to an XML target. However, all ports within a single group of the XML target must receive data from the same active source.

The following transformations are active sources:

- Aggregator

- Application Source Qualifier
- Custom, configured as an active transformation
- Java, configured as an active transformation
- Joiner
- MQ Source Qualifier
- Normalizer (VSAM or pipeline)
- Rank
- Sorter
- Source Qualifier
- SQL
- XML Source Qualifier
- Mapplet, if the mapplet contains one of the above transformations

Selecting a Root Element

If an XML definition has more than one possible root, you can specify a root element for a target instance.

To specify the root element:

1. Right-click the target definition in the Mapping Designer and select Edit.
2. Click the Properties tab.
3. Click the arrow in the Root Element value column.

   The Select Root dialog box appears.
4. Select an element from the list.

Connecting Target Ports

You need to correctly connect XML target ports in a mapping so the Integration Service can create a valid XML file during a session. When you save or validate a mapping with an XML target, the Designer validates the target port connections.

Use the following guidelines when you connect ports in a mapping:

- If you connect one port in a group, you must connect both the foreign key and primary key ports for the group.
- If you connect a foreign key port in a group, you must connect the associated primary key port in the other group. If you do not connect the primary key port of the root group, you do not need to connect the associated foreign key ports in the other groups.
- If you use an XML schema with a default attribute value, you must connect the attribute port to create the default attribute in the target. If you pass a null value through the connected port, the Integration Service writes the default value to the target.

Connecting Abstract Elements

An abstract element cannot occur directly in an XML instance file. Instead, you must use an element derived from the abstract element. By default, the Designer creates a view for any abstract complex element. To reduce metadata, elements from the abstract type do not repeat in any derived type. When you map data to the abstract type, you need to also map data to at least one derived type.

During a session, if the Integration Service loads data to an abstract type, then the Integration Service should also have data for a non-abstract derived type associated with the abstract type. If the derived type has no data, then the Integration Service does not write the abstract element in the target XML file.

Flushing XML Data to Targets

You can flush data to an XML target at each commit point in a session. However, each input group must receive data from the same transaction control point in the mapping. When you create a session based on this mapping, you can append data to the XML file target at each commit or create a new file at each commit. You can specify either option with the On Commit target property.

When you connect the XML target input groups to multiple transaction control points, the Integration Service writes the data to the XML file target after it processes all source rows.

Naming XML Files Dynamically

You can add a FileName column to an XML target definition to dynamically create file names for XML files. When the Integration Service passes data to the FileName column, the Integration Service overrides the output file name in the target properties. For example, if you pass the string “Harry” to the FileName column, the Integration Service names the XML file Harry.

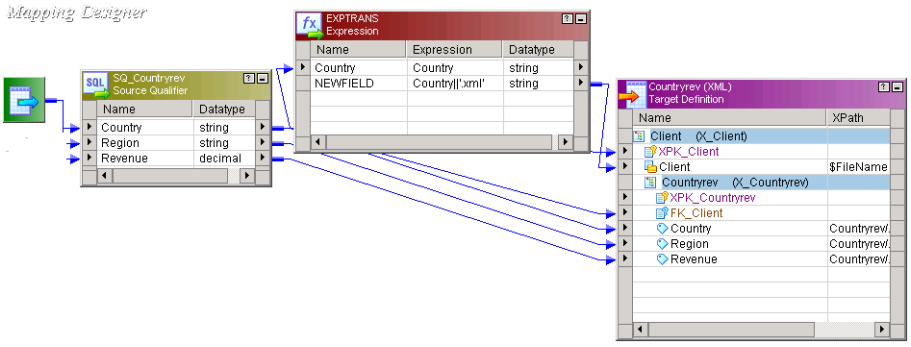
Note: If you are creating a new XML file on each commit, you need to dynamically name each XML file you create. If you do not dynamically name each XML file, the Integration Service overwrites the XML file from the previous commit.

The Integration Service generates a new XML file for each distinct primary key value in the root group of the target. You add a FileName column to set different names for each file. Each name overrides the output file name in the session properties.

Example

The Expression transformation generates a file name from the Country XML element and passes the value to the FileName column. The mapping passes a country to the target root, which is called Client. Whenever the Client value changes, the Integration Service creates a new XML file. The Integration Service creates a list file that contains each XML target file name. The Integration Service lists the absolute path to each file in the list.

The following figure shows a mapping containing an XML target with a FileName column:



The Integration Service passes the following rows to the target:

```
Country,Region,Revenue
USA,region1,1000
France,region1,10
Canada,region1,100
USA,region2,200
USA,region3,300
```

```
USA,region4,400
France,region2,20
France,region3,30
France,region4,40
```

The session produces the following files by country name:

```
Canada.xml
France.xml
USA.xml
```

The list file name is the output file name from the session properties:

```
revenue_file.xml.lst
```

Troubleshooting XML Targets

I imported a source definition from an XML file. Then I imported a target definition from the same XML file. The default groups for the source and target definitions are not the same.

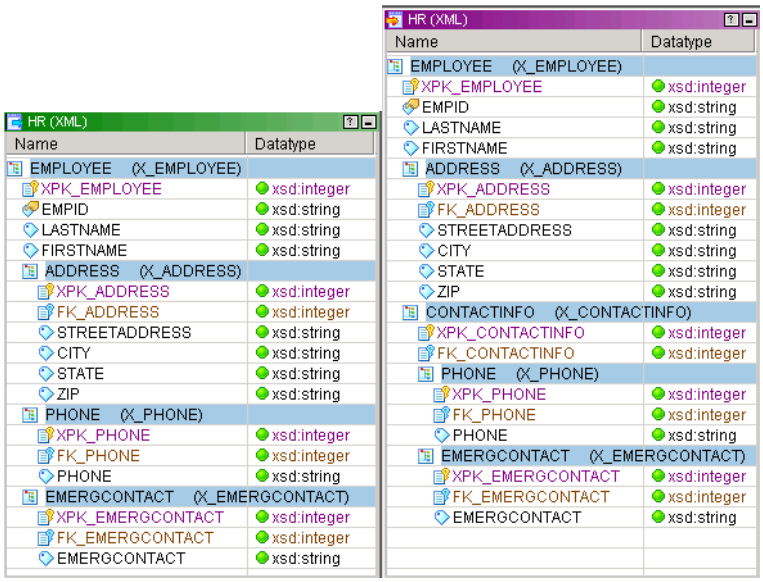
The XML Wizard does not always create the same group structure for a source definition and a target definition if you change some of the options when you import the target.

For example, the ContactInfo element in the following DTD is an enclosure element. The enclosure element has no text content, but has maxOccurs > 1. The child elements also have maxOccurs > 1.

```
<!ELEMENT HR (EMPLOYEE+)>
<!ELEMENT EMPLOYEE (LASTNAME,FIRSTNAME,ADDRESS+,CONTACTINFO+)>
<!ATTLIST EMPLOYEE EMPID CDATA #REQUIRED>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT ADDRESS (STREETADDRESS,CITY,STATE,ZIP)>
<!ELEMENT STREETADDRESS (#PCDATA)>
<!ELEMENT CITY (#PCDATA)>
<!ELEMENT STATE (#PCDATA)>
<!ELEMENT ZIP (#PCDATA)>
<!ELEMENT CONTACTINFO (PHONE+,EMERGCONTACT+)>
<!ELEMENT PHONE (#PCDATA)>
<!ELEMENT EMERGCONTACT (#PCDATA)>
```

If you do not create XML views for the enclosure elements in the source definition, you do not create the ContactInfo element in the source.

The following figure shows the source and target definitions that the XML Wizard creates:



HR (XML)	
Name	Datatype
EMPLOYEE (X_EMPLOYEE)	
XPK_EMPLOYEE	xsd:integer
EMPID	xsd:string
LASTNAME	xsd:string
FIRSTNAME	xsd:string
ADDRESS (X_ADDRESS)	
XPK_ADDRESS	xsd:integer
FK_ADDRESS	xsd:integer
STREETADDRESS	xsd:string
CITY	xsd:string
STATE	xsd:string
ZIP	xsd:string
PHONE (X_PHONE)	
XPK_PHONE	xsd:integer
FK_PHONE	xsd:integer
PHONE	xsd:string
EMERGCONTACT (X_EMERGCONTACT)	
XPK_EMERGCONTACT	xsd:integer
FK_EMERGCONTACT	xsd:integer
EMERGCONTACT	xsd:string

HR (XML)	
Name	Datatype
EMPLOYEE (X_EMPLOYEE)	
XPK_EMPLOYEE	xsd:integer
EMPID	xsd:string
LASTNAME	xsd:string
FIRSTNAME	xsd:string
ADDRESS (X_ADDRESS)	
XPK_ADDRESS	xsd:integer
FK_ADDRESS	xsd:integer
STREETADDRESS	xsd:string
CITY	xsd:string
STATE	xsd:string
ZIP	xsd:string
CONTACTINFO (X_CONTACTINFO)	
XPK_CONTACTINFO	xsd:integer
FK_CONTACTINFO	xsd:integer
PHONE (X_PHONE)	
XPK_PHONE	xsd:integer
FK_PHONE	xsd:integer
PHONE	xsd:string
EMERGCONTACT (X_EMERGCONTACT)	
XPK_EMERGCONTACT	xsd:integer
FK_EMERGCONTACT	xsd:integer
EMERGCONTACT	xsd:string

The source definition does not include the ContactInfo element. The target definition includes the ContactInfo element. The wizard does not include the ContactInfo element in the source definition because you chose not to create views for enclosure elements when you created the source. However, the wizard includes the ContactInfo element in the target definition.

The XML target definition I created from my relational sources contains all elements, but no attributes. How can I modify the target hierarchy so that I can mark certain data as attributes?

You cannot modify the component types that the wizard creates from relational tables. However, you can view a DTD or an XML schema file of the target XML hierarchy. Save the DTD or XML schema file with a new file name. Open this new file and modify the hierarchy, setting the attributes and elements. Then, use the file to import a target definition with a new hierarchy.

CHAPTER 6

XML Source Qualifier Transformation

This chapter includes the following topics:

XML Source Qualifier Transformation Overview

When you add an XML source definition to a mapping, you need to connect the source definition to an XML Source Qualifier transformation. The XML Source Qualifier transformation defines the data elements that the Integration Service reads during a session. The source qualifier determines how the PowerCenter reads the source data. The XML Source Qualifier transformation is an active transformation.

You can manually add a source qualifier transformation, or you can create a source qualifier transformation by default when you add a source definition to a mapping.

You can edit some of the properties and add metadata extensions to an XML Source Qualifier transformation.

When you connect an XML Source Qualifier transformation in a mapping, you must follow rules to create a valid mapping.

Adding an XML Source Qualifier to a Mapping

An XML Source Qualifier transformation has one input/output port for every column in the XML source. When you create an XML Source Qualifier transformation for a source definition, the Designer links each port in the XML source definition to a port in the XML Source Qualifier transformation. You cannot remove or edit any of the links. If you remove an XML source definition from a mapping, the Designer also removes the corresponding XML Source Qualifier transformation. You can link one XML source definition to one XML Source Qualifier transformation.

You can link ports of one XML Source Qualifier group to ports of different transformations to form separate data flows. However, you cannot link ports from more than one group in an XML Source Qualifier transformation to ports in the same target transformation.

If you drag columns from more than one group to a transformation, the Designer copies the columns of all the groups to the transformation. However, the Designer links only the ports of the first group to the corresponding ports of the new columns in the transformation.

You can add an XML Source Qualifier transformation to a mapping by dragging an XML source definition into the Mapping Designer workspace or by manually creating the source qualifier.

Creating an XML Source Qualifier Transformation by Default

When you drag an XML source definition into the Mapping Designer workspace, the Designer creates an XML Source Qualifier transformation by default.

To create an XML Source Qualifier transformation by default:

1. In the Mapping Designer, create a new mapping or open an existing one.
2. Drag an XML source definition into the mapping.

   The Designer creates an XML Source Qualifier transformation and links each port in the XML source definition to a port in the XML Source Qualifier transformation.

Creating an XML Source Qualifier Transformation Manually

You can create an XML Source Qualifier transformation in a mapping if you have a mapping that contains XML source definitions without Source Qualifiers or if you delete the XML Source Qualifier transformation from a mapping.

To create an XML Source Qualifier transformation manually:

1. In the Mapping Designer, create a new mapping or open an existing one.

   Note: There must be at least one XML source definition without a source qualifier in the mapping.
2. Click Transformation > Create.

   The Create Transformation dialog box appears.
3. Select XML Source Qualifier transformation, and type a name for the transformation.

   The naming convention for XML Source Qualifier transformations is XSQ_*TransformationName*.
4. Click Create.

   The Designer lists all the XML source definitions in the mapping with no corresponding XML Source Qualifier transformations.
5. Select a source definition and click OK.

   The Designer creates an XML Source Qualifier transformation in the mapping and links each port of the XML source definition to a port in the XML Source Qualifier transformation.

Editing an XML Source Qualifier Transformation

You can edit XML Source Qualifier transformation properties, such as transformation name and description.

To edit an XML Source Qualifier transformation:

1. In the Mapping Designer, open the XML Source Qualifier transformation.
2. On the Transformation tab, edit the properties.

   The following table describes the transformation properties:

Transformation Setting	Description
Select Transformation	Displays the transformation you are editing. To choose a different transformation to edit, select the transformation from the list.
Rename	Edits the name of the transformation.
Description	Describes the transformation.

3. Click the Ports tab to view the XML Source Qualifier transformation ports.

   Use the Sequence column to set start values for generated keys in XML groups. You can enter a different value for each generated key. Sequence keys are of bigint datatype. Whenever you change these values, the sequence numbers restart the next time you run a session.
4. Click the Properties tab to configure properties that affect how the Integration Service runs the mapping during a session.

   The following table describes the XML Source Qualifier properties:

Properties Setting	Description
Select Transformation	Displays the transformation you are editing. To choose a different transformation to edit, select the transformation from the list.
Tracing Level	Determines the amount of information about this transformation that the Integration Service writes to the session log when it runs the workflow. You can override this tracing level when you configure a session.
Reset	At the end of a session, the Integration Service resets the start values to the start values for the current session.
Restart	At the beginning of a session, the Integration Service starts the generated key sequence for all groups at one.

5. Click the Metadata Extensions tab to create, edit, and delete user-defined metadata extensions.

   You can create, modify, delete, and promote non-reusable metadata extensions, and update their values. You can also update the values of reusable metadata extensions.
6. Click OK.

Setting Sequence Numbers for Generated Keys

Each view in the XML Source Qualifier definition has a primary key and sequence value for the key. During a session, the Integration Service generates keys from sequence values and increments the values.

At the end of the session, the Integration Service updates each sequence value in the repository to the current value plus 1. These values become the start values the next time the Integration Service processes the Sequence Generator transformation.

The repository maintains the following sequence values:

- **Default value**. The sequence value for a key that appears in the XML Source Qualifier when you first create the source qualifier. The default is 1 for each key.
- **Start value.** A sequence number value for a key at the start of a session. You can view the start values in the XML Source Qualifier transformation before you run a workflow.
- **Current value.** A sequence value for a key during a session.

The start values for the generated keys display in the Sequence column in the XML Source Qualifier.

Note: If you edit the sequence start values on the Ports tab, you must save the changes and exit the Designer before you run a workflow.

Changing Sequence Start Values

You can change sequence start values before or after a session by using the following options on the XML Source Qualifier transformation Properties tab:

- **Reset.** At the end of a session, the Integration Service resets the start values back to the start values for the current session. For example, at the beginning of a session, the start value of a key is 2000. At the end of a session, the current value is 2500. When the session completes, the start value in the repository remains at 2000. You might use this option when you are testing and you want to generate the same key numbers the next time you run a session.
- **Restart.** At the beginning of a session, the Integration Service restarts the start values using the default value. For example, if the start value for a key is 1005, and you select Restart, the Integration Service changes the start value to 1. You might use this option if the keys are getting large and you will have no duplicate key conflicts if you restart numbering.

Using the XML Source Qualifier in a Mapping

Each group in an XML source definition is analogous to a relational table, and the Designer treats each group within the XML Source Qualifier transformation as a separate source of data.

The Designer enforces concatenation rules when you connect objects in a mapping. Therefore, you need to organize the groups in the XML source definition so that each group contains all the information you require in one pipeline branch.

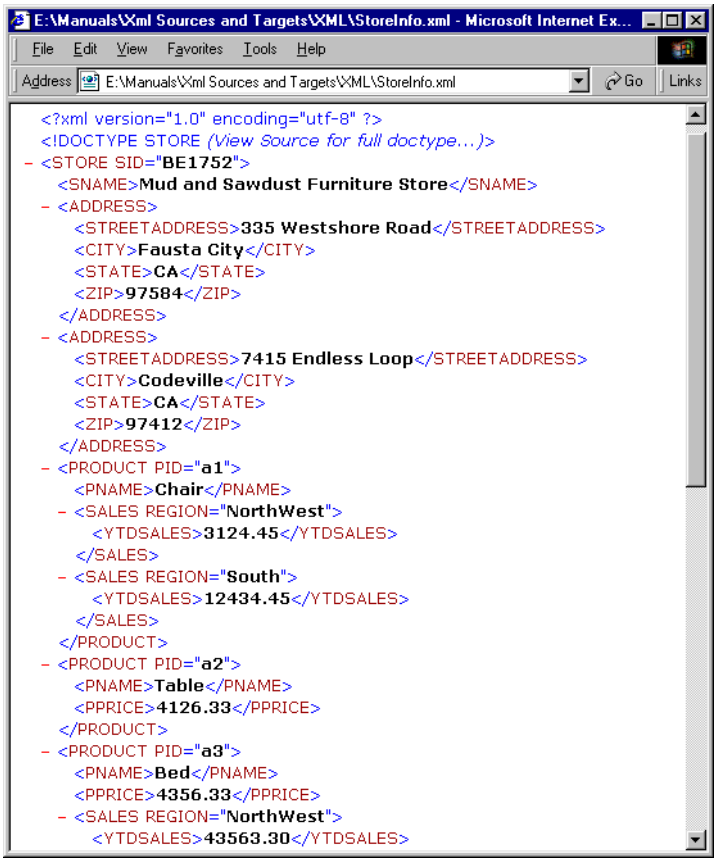
Consider the following rules when you connect an XML Source Qualifier transformation in a mapping:

- **You can link ports from one group in an XML Source Qualifier transformation to ports in one input group of another transformation.** You can *copy* the columns of several groups to one transformation, but you can *link* the ports of only one group to the corresponding ports in the transformation.
- **You can link ports from one group in an XML Source Qualifier transformation to ports in more than one transformation.** Each group in an XML Source Qualifier transformation can be a source of data for more than one pipeline branch. Data can pass from one group to several different transformations.
- **You can link multiple groups from one XML Source Qualifier transformation to different input groups in a transformation.** You can link multiple groups from an XML Source Qualifier transformation to different input groups in most multiple input group transformations, such as a Joiner or Custom transformations. However, you can link multiple groups from one XML Source Qualifier transformation to one Joiner transformation if the Joiner has sorted input. To connect two XML Source Qualifier transformation groups to a Joiner transformation with unsorted input, you must create two instances of the same XML source.

XML Source Qualifier Transformation Example

This section shows an example of an XML Source Qualifier transformation in a mapping. The example uses an XML file containing store, product, and sales information.

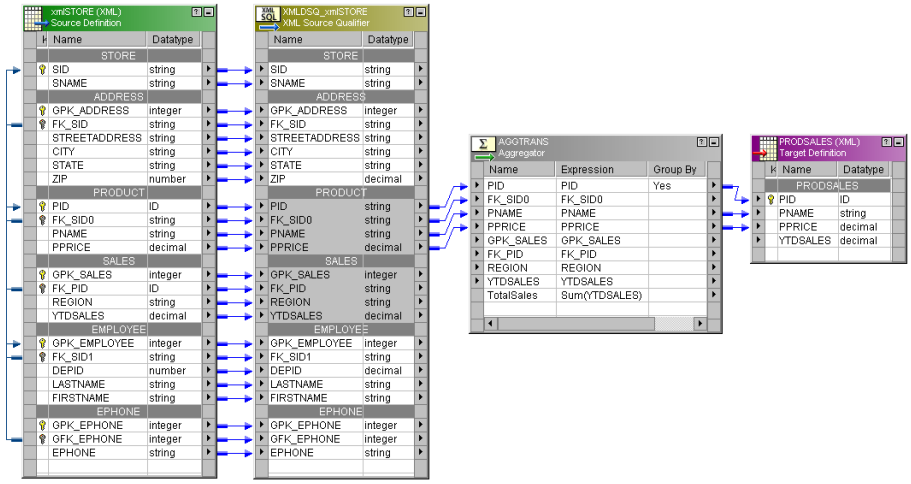
The following figure shows the StoreInfo.xml file:



You might want to calculate the total YTD sales for each product in the XML file regardless of region. Besides sales, you also want the names and prices of each product.

To do this, you need both product and sales information in the same transformation. However, when you import the StoreInfo.xml file, the Designer creates separate groups for products and sales by default.

The following figure shows the default groups for the StoreInfo file with the product and sales information in separate groups:



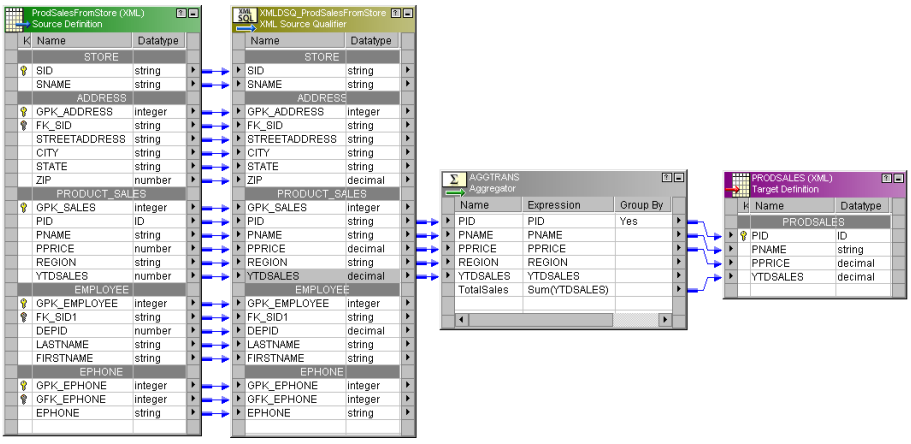
Since you cannot link both the Product and the Sales groups to the same single input group transformation, you can create the mapping in one of the following ways:

- Use a denormalized group containing all required information.
- Join the data from the two groups using a Joiner transformation.

Using One Denormalized Group

You can organize the groups in the source definition so all the information comes from the same group. For example, you can combine the Product and Sales groups into one denormalized group in the source definition. You can process all the information for the sales aggregation from the denormalized group through one data flow.

The following figure shows a denormalized group Product_Sales containing a combination of columns from both the Product and Sales groups:



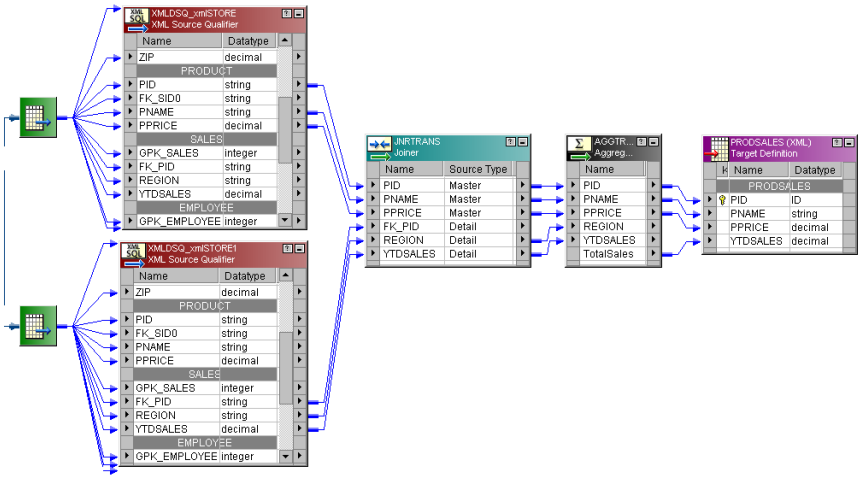
To create the denormalized group, edit the source definition in the Source Analyzer. You can either create a new group or modify an existing group. Add product and sales columns to the group in order to do the sales calculation in the Aggregator transformation. Use the XML Editor to create the group and validate the group.

Joining Two XML Source Qualifier Transformation Groups

You can join data from two source groups into one data flow. Join data from the groups using a Joiner transformation. When you configure the Joiner transformation for sorted input, you can link two groups from one XML Source Qualifier transformation instance to the Joiner transformation. When you use a Joiner transformation configured for unsorted input, you must use two instances of the same XML source and link a group from each XML Source Qualifier transformation instance to the Joiner transformation.

You can then send the data from the Joiner transformation to an Aggregator transformation to calculate the YTDSales for each product.

The following figure shows how you can create two instances of the same XML source and join data from two XML Source Qualifier transformations:



Troubleshooting XML Source Qualifier Transformations

When I drag two groups from an XML Source Qualifier transformation to a transformation, the Designer copies the columns but does not link all the ports.

You can link one group of an XML Source Qualifier transformation to one transformation. When you drag more than one group to a transformation, the Designer copies all the column names to the transformation. However, the Designer links the columns of only the first group.

I cannot break the link between the XML source definition and its source qualifier.

The XML Source Qualifier transformation columns match the corresponding XML source definition columns. You cannot remove or modify the links between an XML source definition and its XML Source Qualifier transformation. When you remove an XML source definition, the Designer removes its XML Source Qualifier transformation.

CHAPTER 7

Midstream XML Transformations

This chapter includes the following topics:

Midstream XML Transformations Overview

XML definitions read or create XML data. However, sometimes you need to extract or generate XML inside a pipeline. For example, you might want to send a message to a TIBCO target containing an XML document as the data field. In this case, you need to generate an XML document before sending the message to TIBCO. Use a XML transformation to generate the XML.

You can create the following types of midstream XML transformation:

- **XML Parser transformation.** The XML Parser transformation reads XML from one input port and outputs data to one or more groups.
- **XML Generator transformation.** The XML Generator transformation reads data from one or more sources and generates XML. The XML Generator transformation has a single output port.

Use a midstream XML transformation to extract XML data from messaging systems, such as TIBCO, WebSphere MQ, or from other sources, such as files or databases. The XML transformation functionality is similar to the XML source and target functionality, except the midstream XML transformation parses the XML or generates the document in the pipeline.

Midstream XML transformations support the same XML schema components that the XML Wizard and XML Editor support. In addition, XML transformations support the following functionality:

- **Pass-through ports**. Use pass-through ports to pass non-XML data through the midstream transformation. These fields are not part of the XML schema definition, but you use them to generate denormalized XML groups. You use these fields in the same manner as top-level XML elements. You can also use a pass-through field as a primary key for the top-level group in the XML definition.

- **Real-time processing.** Use a midstream XML transformation to process data as BLOBs from messaging systems.
- **Support for multiple partitions.** You can generate different XML documents for each partition.

XML Parser Transformation

When the Integration Service processes an XML Parser transformation, it reads a row of XML data, parses the XML, and returns data through output groups. The XML Parser transformation returns non-XML data in pass-through ports. You can parse XML messages from sources such as JMS or IBM WebSphere MQ . The XML Parser transformation is an active transformation.

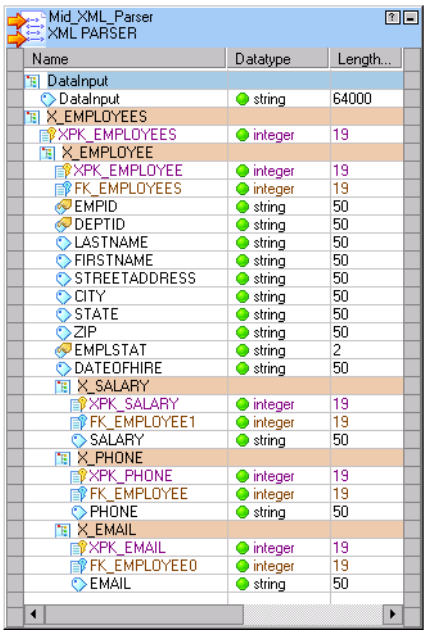
The XML Parser transformation has one input group and one or more output groups. The input group has one input port, DataInput, which accepts an XML document in a string.

When you create an XML Parser transformation, use the XML Wizard to import an XML, DTD, or XML schema file. For example, you can import the following Employee DTD file:

```
<!ELEMENT EMPLOYEES (EMPLOYEE+)>
<!ELEMENT EMPLOYEE (LASTNAME, FIRSTNAME, ADDRESS, PHONE+, EMAIL*,   EMPLOYMENT)>
  <!ATTLIST EMPLOYEE EMPID CDATA #REQUIRED
                     DEPTID CDATA #REQUIRED>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT ADDRESS (STREETADDRESS, CITY, STATE, ZIP)>
<!ELEMENT STREETADDRESS (#PCDATA)>
<!ELEMENT CITY (#PCDATA)>
<!ELEMENT STATE (#PCDATA)>
<!ELEMENT ZIP (#PCDATA)>
<!ELEMENT PHONE (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>
<!ELEMENT EMPLOYMENT (DATEOFHIRE, SALARY+)>
<!ATTLIST EMPLOYMENT EMPLSTAT (PF|PP|TF|TP|O) "PF">
<!ELEMENT DATEOFHIRE (#PCDATA)>
<!ELEMENT SALARY (#PCDATA)>
```

The XML Parser transformation shows the root view, X_Employees, with X_Employees shown as a parent of X_Employee. X_Employee is a parent of X_Salary, X_Phone, and X_Email.

The following figure shows the XML Parser transformation that the Designer creates if you choose to create entity relationships:



The Designer creates a root view, X_Employees. X_Employees is the parent of X_Employee. X_Employee is a parent of X_Salary, X_Phone, and X_Email.

Each view in the XML Parser transformation has at least one key to establish its relationship with another view. If you do not designate the keys in the XML Editor, the Designer creates the primary and foreign keys for each view. The keys are of datatype bigint. The keys are called generated keys because the Integration Service creates the key values each time it returns a row from the XML Parser transformation.

When the Designer creates a primary or foreign key column, it assigns a column name with a prefix. In an XML definition, the prefix is XPK_ for a generated primary key column and XFK_ for a generated foreign key column. A foreign key always refers to a primary key in another group. A generated foreign key column always refers to a generated primary key column.

For example, the group X_Employee has the XPK_Employee primary key. The Designer creates foreign key columns that connect the X_Phone, X_Email, and X_Salary to the X_Employee group. Each group has the foreign key column XFK_Employee.

The repository stores the key values. You cannot change the values in the repository, but you can choose to reset or restart the sequence numbers after a session.

XML Parser Input Validation

You can configure the XML Parser transformation to validate XML before parsing it. The XML Parser transformation validates the XML against a schema. If the XML is not valid for the schema, a row error occurs. The XML Parser transformation returns the XML and associated error messages to a separate output group. You can pass the invalid XML and error message to a target.

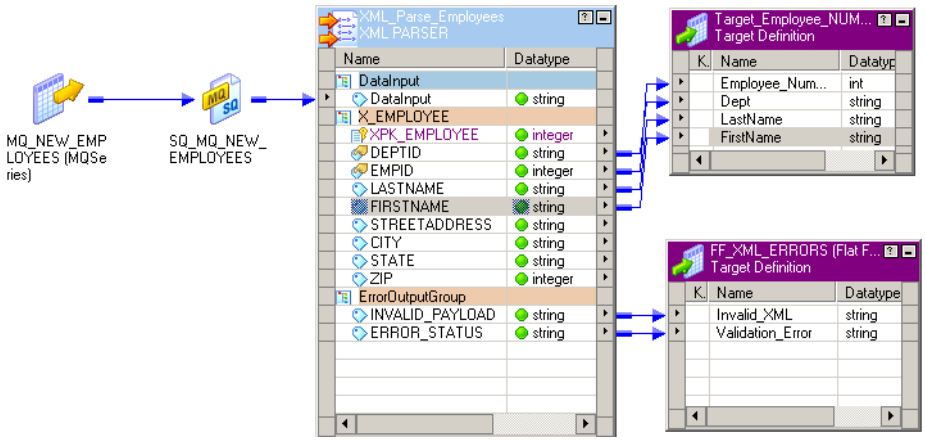
For example, a real-time PowerCenter session reads XML messages from a WebSphere MQSeries source. The session runs with a source-based commit. A message in the commit transaction has an invalid XML payload. To prevent the commit from failing, you can configure the XML Parser transformation to return the invalid XML to a separate output group from the valid data. The XML Parser transformation processes the valid XML messages and completes the transaction.

The session log contains a message that indicates when Route Invalid Payload Through Data Flow is enabled. When you set the session tracing level to Normal, the Integration Service writes a message to the session log that indicates whether the validation is successful. The log message contains the location of the schema the XML Parser accessed to validate the XML. When XML streaming is enabled and the XML is invalid, the Integration Service truncates the XML and passes it to the Invalid_Payload port. The Integration Service logs the invalid XML in the session log.

To configure the XML Parser transformation to validate the XML, enable the Route Invalid Payload Through Data Flow option on the Midstream XML Parser tab. The Designer adds the following ports to the XML Parser transformation:

- **Invalid_Payload**. Returns invalid XML messages to the pipeline. If the XML payload is valid, the Invalid_Payload port contains a null value. This port has the same precision as the DataInput port.
- **Error_Status**. Contains the error string or status returned from the XML validation. If the XML is valid for the current row, Error_Status contains a null value. This port has the same precision as the DataInput port.

The following figure shows an XML Parser transformation that routes invalid XML messages to an Errors target table:



The mapping contains the following objects:

- **MQSeries source definition.** Contains employee XML data in the message data field.
- **Source Qualifier transformation**. Reads data from WebSphere MQ. Contains a set of ports that represent the message header fields and the message data field.
- **XML Parser transformation**. Receives the XML message data in the DataInput port. When the XML is valid, the XML Parser transformation returns the employee data and passes it to a target. When the XML is not valid, the XML Parser transformation returns the XML in the Invalid_Payload port. It returns an error message in the Error_Status port.
- **Employees target definition**. Receives rows of valid employee data.
- **XML_Errors target definition.** Receives invalid XML and error messages.

Configure the XML Schema Location attribute in the session properties for the transformation. Enter the name and location of the schema to validate the XML against. You can configure workflow, session, or mapping variables and parameters for the XML schema definition. You can configure multiple schemas for validation if you separate them with semi-colons.

You can use a DTD for validation if you include it in the input XML payload. You cannot configure a DTD in the XML Schema Location attribute or use it to route invalid XML data to the Invalid Payload port.

If you enable XML streaming, verify that the precision for the Invalid_Payload port matches the maximum message size. If the port precision is less than the message size, the XML Parser transformation returns truncated XML in the Invalid_Payload port, and writes an error in the session log.

Stream XML to the XML Parser Transformation

You can configure a session to stream the XML from an Unstructured Data transformation, JMS source, or WebSphere MQ source to the XML Parser transformation. When the PowerCenter Integration Service streams XML data, it splits XML data into multiple segments.

You can configure a smaller input port in the XML Parser transformation and reduce the amount of memory that the XML Parser transformation requires to process large XML files. You can parse XML files that are larger than 100 MB.

When you enable XML streaming, the XML Parser transformation receives data in segments that are less than or equal to the port size. When the XML file is larger than the port size, the PowerCenter Integration Service passes more than one row to the XML Parser transformation. Each XML row has a row type of streaming. The last row has a row type of insert.

The input port precision must be equal to or greater than the output port precision of the object that passes the XML to the XML Parser transformation. When most of the XML documents are small, but some messages are large, set the XML Parser transformation port size to the size of the smaller messages for best performance.

If you enable XML streaming, you must also enable XML streaming for the source or transformation that is passing the XML data to the XML Parser transformation. If you do not enable streaming, the XML Parser receives the XML in one row, which might slow performance.

To enable XML streaming in the XML Parser transformation, select Enable XML Input Streaming in the XML Parser transformation session properties. If you enable XML streaming in the source or transformation, but you do not enable it for the XML Parser transformation, the XML Parser transformation cannot process the XML file.

When you enable XML streaming and an error occurs in the XML document, the PowerCenter Integration Service writes the XML document to the session log by default. You can configure the session to write the XML document to the error log file when an error occurs.

Enable Log Source Row Data in the session properties. When you enable logging, and an error occurs in the XML document, the PowerCenter Integration Service generates a row error. The PowerCenter Integration Service writes the XML document to the error log file and it increments the error count.

For information about XML sizing in PowerCenter, see "Using XML with PowerCenter Overview" on page 32. For more information about the limitations that apply to XML handling in PowerCenter, see "Limitations" on page 33.

To create a transformation with other element types, and to transform larger XML input files, use a Data Processor transformation. For more information about how to create Data Processor transformations, see the *Informatica Data Transformation User Guide* and the *Informatica Data Transformation Getting Started Guide*.

XML Decimal Datatypes

When you define the precision of an XML decimal element to be greater than 34 digits, the Integration Service calls an external function to convert the XML decimal datatype to a Double in the XML Parser transformation. The function returns a Double with a precision length that is dependent on the node that is running the Integration Service. On all platforms, the precision is guaranteed to be 17 digits before the number is rounded, but the precision might be more on some platforms.

For example, on Windows 32-bit, the Integration Service rounds the number after 17 digits.

```
1234.567890123456789 is converted to 1234.567890123460800.
```

On HP-UX 32-bit, the Integration Service rounds the number after 34 digits.

XML Generator Transformation

Use an XML Generator transformation to combine input that comes from several sources to create an XML document. For example, use the transformation to combine the XML data from two TIBCO sources into one TIBCO target. One source might contain employee and salary information, and the other might have employee phone and email information. The XML Generator transformation is an active and a connected transformation.

The XML Generator transformation is similar to an XML target definition. When the Integration Service processes an XML Generator transformation, it writes rows of XML data. The Integration Service can also process pass-through fields containing non-XML data in the transformation.

The XML Generator transformation has one or more input groups and one output group. The output group has one port, “DataOutput,” which generates a string data BLOB XML document. The output group contains the pass-through port when you create pass-through fields.

Creating a Midstream XML Transformation

When you create a midstream XML transformation, you use the XML Wizard and XML Editor to define the XML groups. You can create the transformation in the Transformation Developer and the Mapping Designer.

To create a midstream XML transformation:

1. Open the Transformation Developer or Mapping Designer.
2. Click Transformation > Create.

   The Create Transformation dialog box appears.
3. Select the XML Parser or XML Generator transformation type.
4. Enter a transformation name, and click Create.

   The Import XML Definition dialog box appears.
5. Choose a file to import, and click Open.

   The XML Wizard appears.
6. Create the XML definitions using the XML Wizard.
7. Click Finish in the XML Wizard.

   The midstream XML transformation appears in the workspace.
8. To edit the midstream XML transformation properties, double-click the transformation in the workspace.

Synchronizing a Midstream XML Definition

You can synchronize a midstream XML transformation using a different version of the schema or source file that you imported to create the transformation. For example, you might add new elements to the schema file you imported to create an XML Parser transformation. You can update the XML Parser transformation with the new schema instead of deleting the transformation and re-creating the transformation.

To synchronize a midstream XML transformation, use the Transformation Developer or Mapping Designer.

To synchronize a midstream XML transformation:

1. Open the Transformation Developer or Mapping Developer.
2. Drag the midstream XML transformation or the mapping you want to update into the workspace.
3. Right-click the top of the midstream XML transformation.
4. Select Synchronize XML transformation.

   The Import XML Definition dialog box appears.
5. Navigate to the repository definition or file that you used to create the XML Parser or XML Generator transformation.
6. Click Open to update the transformation.

The Designer cannot synchronize a transformation with a file that you did not use to create the transformation.

You use the Source Analyzer and Target Designer to synchronize source and target XML definitions.

Editing Midstream XML Transformation Properties

You can edit some of the midstream XML transformation properties. However, because you use the XML Wizard and XML Editor to define the transformation, you must use these tools to change the XML definition.

When you create a midstream XML transformation in the Mapping Designer, the following rules apply:

- If you make the transformation reusable, you can change some of the transformation properties from the Mapping Designer. You cannot add pass-through ports or metadata extensions.
- If you create a non-reusable transformation, you can edit the transformation from the Mapping Designer.

When you configure a midstream XML transformation, you can configure components on the following tabs:

- **Transformation tab.** Rename the transformation and add a description on the Transformation tab.
- **Ports tab.** Display the transformation ports and attributes that you create on the XML Parser or XML Generator tab.
- **Properties tab.** Update the tracing level.
- **Initialization Properties tab.** Create run-time properties that an external procedure uses during initialization.
- **Metadata Extensions tab.** Extend the metadata stored in the repository by associating information with repository objects, such as an XML transformation.
- **Port Attribute Definitions tab.** Define port attributes that apply to all ports in the transformation.
- **Midstream XML Parser or XML Generator tab.** Create pass-through ports using this tab. Pass-through ports enable you to pass non-XML data through the transformation. For the XML Parser transformation, you can choose to reset sequence numbers if you use sequence numbering to generate XML column names. For the XML Generator transformation, you can choose to create a new XML document on commits.

Properties Tab

Configure the midstream XML transformation properties on the Properties tab.

The following table describes the options you can change on the Properties tab:

Transformation	Description
Runtime Location	Location that contains the DLL or shared library. Default is $PMExtProcDir. Enter a path relative to the Integration Service node that runs the XML session. If this property is blank, the Integration Service uses the environment variable defined on the Integration Service node to locate the DLL or shared library. You must copy all DLLs or shared libraries to the runtime location or to the environment variable defined on the Integration Service node. The Integration Service fails to load the procedure when it cannot locate the DLL, shared library, or a referenced file.
Tracing Level	Amount of detail displayed in the session log for this transformation. Default is Normal.
Transformation Scope	Indicates how the Integration Service applies the transformation logic to incoming data. You can choose one of the following transformation scope values for the XML Parser transformation: - Row. Applies the transformation logic to one row of data at a time. Flushes the rows generated for all the output groups before processing the next row. - Transaction. Applies the transformation logic to all rows in a transaction. Flushes generated rows at transaction boundaries, when output blocks fill up, and at end of file. - All Input. Applies the transformation logic to all incoming data. Flush generated rows only when the output blocks fill up and at end of file. For the XML Generator transformation, the Designer sets the transformation scope to all input when you set the On Commit setting to Ignore Commit. The Designer sets the transformation scope to the transaction level if you set On Commit to Create New Doc.
Output is Repeatable	Indicates if the order of the output data is consistent between session runs. - Never. The order of the output data is inconsistent between session runs. - Based On Input Order. The output order is consistent between session runs when the input data order is consistent between session runs. - Always. The order of the output data is consistent between session runs even if the order of the input data is inconsistent between session runs. Default is Based on Input Order for the XML Parser transformation. Default is Always for the XML Generator transformation.
Requires Single Thread per Partition	Indicates if the Integration Service must process each partition with one thread.
Output is Deterministic	Indicates whether the transformation generates the same output data between session runs. You must enable this property to perform recovery on sessions that use this transformation. Default is enabled.

Warning: If you configure a transformation as repeatable and deterministic, it is your responsibility to ensure that the data is repeatable and deterministic. If you try to recover a session with transformations that do not produce the same data between the session and the recovery, the recovery process can result in corrupted data.

Midstream XML Parser Tab

Use the Midstream XML Parser tab to modify the size of the DataInput port. You can also add pass-through ports on this tab.

You can access the XML Editor from the Midstream XML Parser Tab. Click the XML Editor button.

Note: When you access the XML Editor, you cannot update Edit Transformations until you exit the XML Editor.

The following table describes the options you can change on the Midstream XML Parser tab:

Transformation	Description
Precision	Length of the column. Default DataInput port precision is 64K. Default precision for a pass-though port is 20. You can increase the precision.
Restart	Always start the generated key sequence at 1. Each time you run a session, the key sequence values in all groups of the XML definition start over at 1.
Reset	At the end of a session, reset the value sequence for all generated keys in all groups. Reset the sequence numbers back to where they were before the session.
Route Invalid Payload Through Data Flow	Validate the XML against a schema. If the XML is not valid for the schema, a row error occurs. The XML Parser transformation returns the XML and associated error messages to a separate output group.
Description	Describes the transformation.

Note: If you do not select Reset or Restart, the sequence numbers in the generated keys increase from session to session. If you select the Restart or Reset option, you update the Restart or Reset property that appears on the Initialization Properties tab. You cannot change these options from the Initialization Properties tab, however.

Midstream XML Generator Tab

Use the XML Generator tab to modify the size of the DataOutput port. You can also add pass-through ports on this tab.

You can access the XML Editor from the Midstream XML Generator Tab. Click the XML Editor button. When you access the XML Editor, you cannot edit transformation properties until you exit the XML Editor.

The following table describes the options you can change on the XML Generator transformation tab:

Transformation Setting	Description
Precision	Length of the column. Default DataOutput port precision is 64K. Default precision for a pass-though port is 20. You can increase the precision.
On Commit	The Integration Service can generate multiple XML documents after a commit. Use one of the following options: - Ignore Commit. The Integration Service creates the XML document and writes data to the document at end of file. Use this option if two different sources are connected to the XML Generator transformation. - Create New Document. Creates a new XML document at each commit. Use this option if you are running a real-time session. When a session uses multiple partitions, the Integration Service generates a separate XML document for each partition, regardless of On Commit settings. If you select Create New Document, the Integration Service creates new documents for each partition.
Description	Describes the transformation.

Note: The Designer sets the transformation scope to all input when you set the On Commit setting to Ignore Commit. The Designer sets the transformation scope to the transaction level if you set On Commit to Create New Doc.

Generating Pass-Through Ports

Pass-through ports are columns that pass non-XML data through a midstream XML transformation. For example, you can pass message IDs with XML for MQSeries sources and targets. Use the message ID to correlate input and output messages for requests and replies.

When you define a pass-through port in the midstream transformation, you add the pass-through port to either the DataInput group in the XML Parser transformation or the DataOutput group in the XML Generator transformation.

Once you generate the port, you use the XML Editor to add a corresponding reference port to another view in the XML definition. In the XML Parser transformation, the pass-through port is an input port, and the corresponding reference port is an output port. In the XML Generator transformation, the pass-through port is an output port and the associated reference port is an input port.

To create a pass-through port in a midstream XML transformation:

1. Open the transformation in the Transformation Developer or Mapping Designer.
2. Double-click the transformation to open Edit Transformations.
3. Click the Midstream XML Generator or Midstream XML Parser tab.

   The DataInput or DataOutput port appears, depending on the transformation type.
4. Click the Add button to add an output port for the pass-through.

   A default field appears in the Field Name column.
5. Modify the field name. You can also modify type, precision, and scale depending on the file you used to create the definition.
6. Click XML Editor to open the XML definition for the transformation.

   The XML views in the definition appear in the workspace.
7. Right-click the top of a view to add the reference port.
8. Select Add a Reference Port.

   The Reference Port dialog box opens.

   The dialog box lists the pass-through ports you added in the transformation.
9. Select the pass-through port that will correspond to the new reference port in the view and click OK.

   The corresponding output reference port appears in the view. You can rename the port to a meaningful name in the Columns window.
10. Click Apply Changes and exit the XML Editor.
11. Click OK in the transformation.

Non-XML data comes through the input port called Pass_thru_field and passes through the corresponding COL_0 reference output port.

Troubleshooting Midstream XML Transformations

I need to extract XML files from a database table that contains an XML CLOB. Each XML file can be up to 2 GB. If I create an XML Parser transformation, I need to define a fixed maximum length for the CLOB column. However, the maximum length for the CLOB datatype is 104 MB.

The XML data is too large to pass directly from the table to an XML Parser transformation. You need to stage the CLOB table data to a flat file and create an XML source definition from the file.

For information about XML sizing in PowerCenter, see "Using XML with PowerCenter Overview" on page 32. For more information about the limitations that apply to XML handling in PowerCenter, see "Limitations" on page 33.

To create a transformation with other element types, and to transform larger XML input files, use a Data Processor transformation. For more information about how to create Data Processor transformations, see the *Informatica Data Transformation User Guide* and the *Informatica Data Transformation Getting Started Guide*.

APPENDIX A

XML Datatype Reference

This appendix includes the following topic:

XML and Transformation Datatypes

PowerCenter supports all XML datatypes specified in the W3C May 2, 2001 Recommendation. The following table lists the XML datatypes and compares them to the transformation datatypes in the XML Source Qualifier transformation. For more information about XML datatypes, see the W3C specifications for XML datatypes at http://www.w3.org/TR/xmlschema-2.

You can change the datatypes in XML definitions and in midstream XML transformations if you import an XML file to create the definition. You cannot change XML datatypes when you import them from an XML schema. You cannot change the transformation datatypes for XML sources within a mapping.

The following table describes the XML and corresponding transformation datatypes:

Datatype	Transformation	Range
anySimpleType	String	1 to 104,857,600 characters
anyURI	String	1 to 104,857,600 characters
base64Binary	Binary	1 to 104,857,600 bytes
boolean	Small Integer	Precision 5; scale 0
byte	Small Integer	Precision 5; scale 0
date	Date/Time	Jan 1, 0001 A.D. to Dec 31, 9999 A.D. (precision to the nanosecond)
dateTime	Date/Time	Jan 1, 0001 A.D. to Dec 31, 9999 A.D. (precision to the nanosecond)
decimal	Decimal	Precision 1 to 28; scale 0 to 28
double	Double	Precision 15, scale 0
duration	String	1 to 104,857,600 characters

Datatype	Transformation	Range
ENTITIES	String	1 to 104,857,600 characters
ENTITY	String	1 to 104,857,600 characters
float	Double	Precision 15, scale 0
gDay	Integer	-2,147,483,648 to 2,147,483,647 Precision 10; scale 0
gMonth	Integer	-2,147,483,648 to 2,147,483,647 Precision 10; scale 0
gMonthDay	Date/Time	Jan 1, 0001 A.D. to Dec 31, 9999 A.D. (precision to the nanosecond)
gYear	Integer	Precision 10; scale 0
gYearMonth	Date/Time	Jan 1, 0001 A.D. to Dec 31, 9999 A.D. (precision to the nanosecond)
hexBinary	Binary	1 to 104,857,600 bytes
ID	String	1 to 104,857,600 characters
IDREF	String	1 to 104,857,600 characters
IDREFS	String	1 to 104,857,600 characters
int	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision 19; scale 0
integer	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision 19; scale 0
language	String	1 to 104,857,600 characters
long	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision 19; scale 0
Name	String	1 to 104,857,600 characters
Ncname	String	1 to 104,857,600 characters
negativeInteger	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision 19; scale 0
NMTOKEN	String	1 to 104,857,600 characters
NMTOKENS	String	1 to 104,857,600 characters
nonNegativeInteger	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision 19; scale 0

Datatype	Transformation	Range
nonPositiveInteger	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision 19; scale 0
normalizedString	String	1 to 104,857,600 characters
NOTATION	String	1 to 104,857,600 characters
positiveInteger	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision 19; scale 0
QName	String	1 to 104,857,600 characters
short	Small Integer	Precision 5; scale 0
string	String	1 to 104,857,600 characters
time	Date/Time	Jan 1, 0001 A.D. to Dec 31, 9999 A.D. (precision to the nanosecond)
token	String	1 to 104,857,600 characters
unsignedByte	Small Integer	Precision 5; scale 0
unsignedInt	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision 19; scale 0
unsignedLong	Bigint	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 Precision 19; scale 0
unsignedShort	Integer	-2,147,483,648 to 2,147,483,647 Precision 10; scale 0

XML Date Format

PowerCenter supports the following format for date, time, and datetime datatypes:

```
CCYY-MM-DDThh:mm:ss:sss
```

Use this format or any portion of this format in an XML file. PowerCenter does not support negative dates for datetime format.

Use a date, time, or datetime element in either of the following formats within a session:

```
CCYY-MM
```

- or -

```
CCYY-MM-DD/Thh
```

The format of the first datetime element in an XML file determines the format of all subsequent values of the element. If the Integration Service reads a value for the same date, time, or datetime element that has a different format, the Integration Service rejects the row.

For example, if the first value of a datetime element is in the following format:

```
CCYY-MM-DDThh:mm:ss
```

The Integration Service rejects a row that contains the element in the following format:

```
CCYY-MM-DD
```

The XML parser converts the datetime value in the input XML to a value in the local time zone of the machine that hosts the Integration Service. If you enable the option to adjust the clock for daylight saving changes on Windows, the XML parser adds an hour to the datetime value. For consistent datetime value conversions, do not enable the option on Windows to adjust the clock for daylight saving changes.

APPENDIX B

XPath Query Functions Reference

This appendix includes the following topics:

XPath Query Functions Overview

XPath is a language that describes a way to locate items in an XML document. XPath uses an addressing syntax based on the path through the XML hierarchy from a root component. You can create an XPath query predicate for elements in the view row or a column that has an XPath that includes the view row.

Use an XPath query predicate in an XML view to filter XML source data. In a session, the Integration Service extracts data from a source XML file based on the query. If all queries return TRUE, the Integration Service extracts data for the view.

An XPath query predicate includes an element or attribute to extract, and the query that determines the criteria. You can verify the value of an element or attribute, or you can verify that an element or attribute exists in the source XML data.

This appendix describes each function used in an XPath query predicate. Functions accept arguments and return values. When you create a function, you can include components from the elements and attributes in the XML view, and you can add literal values. When you specify a literal, you must enclose the literal in single or double quotes.

Function Quick Reference

Use the following types of function in an XPath query predicate:

- **String.** Use string functions to test substring values, concatenate strings, or translate strings into other strings. For example, the following XPath query predicate determines if an employee's full name is equal to the concatenation of last name and first name:

  ```
  EMPLOYEE[./FULLNAME=concat(./ENAME/LASTNAME,./ENAME/FIRSTNAME)]
  ```

- **Numeric.** Use numeric functions with element and attribute values. Numeric functions operate on numbers and return integers. For example, the following XPath query predicate rounds discount and tests if the result is greater than 15:

  ```
  ORDER_ITEMS[round(./DISCOUNT) > 15]
  ```

- **Boolean.** Use Boolean functions to test elements, check the language attribute, or force a true or false result. For example, the following XPath query predicate returns true if the value is greater than zero:

  ```
  boolean(string)
  ```

The following table describes XPath query predicate string functions:

Function	Syntax	Description
concat	concat (string1, string2)	Concatenates two strings.
contains	contains (string, substring)	Determines a string contains another string.
normalize-space	normalize-space (string)	Strips leading and trailing white space from a string.
starts-with	starts-with (string, substring)	Determines if string1 starts with string2.
string	string (value)	Converts a number or Boolean to a string.
string-length	string-length (string)	Returns the number of characters in a string, including trailing blanks.
substring	substring (string, start [,length])	Returns a portion of a string starting at a specified position.
substring-after	substring-after (string, substring)	Returns a portion of a string starting at a specified position.

Function	Syntax	Description
substring-before	substring-before (string, substring)	Returns the characters in a string that occur before a substring.
translate	translate (string1, string2, string3)	Converts the characters in a string to other characters.

The following table describes XPath query predicate number functions:

Function	Syntax	Description
ceiling	ceiling (number)	Rounds a number to the smallest integer that is greater than or equal to the passed number.
floor	floor (number)	Rounds a number to the largest integer that is less than or equal to the passed number.
number	number (value)	Converts a string or Boolean value to a number.
round	round (number)	Rounds a number to the nearest integer.

The following table describes XPath query predicate Boolean functions:

Function	Syntax	Description
boolean	boolean (object)	Converts an object to Boolean.
false	false ( )	Always returns FALSE.
lang	lang (code)	Determines if an element has an xml:lang attribute that matches the code argument.
not	not (condition)	Returns TRUE if a Boolean condition is FALSE and FALSE if the Boolean condition is TRUE.
true	true ( )	Always returns TRUE.

boolean

Converts a value to Boolean.

Syntax

```
boolean ( object )
```

The following table describes the boolean argument:

Argument	Description
object	Numeric or character string datatype. Passes a number or string to test.

Return Value

Boolean.

The function returns a Boolean as follows:

- A string returns TRUE if its length is not zero, otherwise it returns FALSE.
- A number returns FALSE if it is zero or not a number (NaN), otherwise it returns TRUE.

Examples

The following example verifies that a name has characters:

```
boolean ( NAME )
```

The following table includes example arguments and return values:

NAME	RETURN VALUE
Lilah	TRUE
-	FALSE

The following example verifies that a zip code is numeric:

```
boolean ( ZIP_CODE )
```

The following table includes example arguments and return values:

ZIP_CODE	RETURN VALUE
94061	TRUE
94005	TRUE
9400g	FALSE

ceiling

Rounds a number to the smallest integer that is greater than or equal to the passed number.

Syntax

```
ceiling ( number )
```

The following table describes the argument for this function:

Argument	Description
number	Numeric value. The number you want to round.

Return Value

Integer.

Example

The following expression returns the price rounded to the smallest integer:

```
ceiling ( PRICE )
```

The following table contains example arguments and return values:

PRICE	RETURN VALUE
39.79	40
125.12	126
74.24	75
NULL	NULL
-100.99	-100
100	100

concat

Concatenates two strings.

Syntax

```
concat ( string1, string2 )
```

The following table describes the arguments for this function:

Argument	Description
string1	String datatype. Passes the first string to concatenate.
string2	String datatype. Passes the second string to concatenate.

Return Value

String.

If one of the strings is NULL, concat ignores it and returns the other string.

Example

The following expression concatenates FIRSTNAME and LASTNAME:

```
concat( FIRSTNAME, LASTNAME )
```

The following table includes example arguments and return values:

FIRSTNAME	LASTNAME	RETURN VALUE
John	Baer	JohnBaer
NULL	Campbell	Campbell

FIRSTNAME	LASTNAME	RETURN VALUE
Greg	NULL	Greg
NULL	NULL	NULL

Tip

The concat function does not add spaces to strings. To add a space between two strings, you can write an expression that contains nested concat functions. For example, the following expression adds a space to the end of the first name and concatenates first name to the last name:

```
concat ( concat ( FIRST_NAME, " " ), LAST_NAME )
```

The following table shows example arguments and return values:

FIRST_NAME	LAST_NAME	RETURN VALUE
John	Baer	John Baer
NULL	Campbell	Campbell *(includes leading space)*
Greg	NULL	Greg

contains

Determines if a string contains another string.

Syntax

```
contains( string, substring )
```

The following table describes the arguments for this function:

Argument	Description
string	String datatype. Passes the string to examine. The argument is case sensitive.
substring	String datatype. Passes the string to search for in the string. The argument is case sensitive.

Return Value

Boolean.

Example

The following expressions returns TRUE if the NAME contains SHORTNAME:

```
contains( NAME, SHORTNAME )
```

The following table includes example arguments and return values:

NAME	SHORTNAME	RETURN VALUE
John	Baer	FALSE
SuzyQ	Suzy	TRUE

NAME	SHORTNAME	RETURN VALUE
WorldPeace	World	TRUE
CASE_SENSITIVE	case	FALSE

false

Always returns FALSE. Use this function to set a Boolean to FALSE.

Syntax

```
false ()
```

The false function does not accept arguments.

Return Value

FALSE.

Example

Combine the false function with other functions to force a FALSE result.

The following table includes expressions that return FALSE:

EXPRESSION	RETURN VALUE
(salary) = false()	FALSE
A/B = false ()	FALSE
starts-with (name, 'T') = false()	FALSE

floor

Rounds a number to the largest integer that is less than or equal to the passed number.

Syntax

```
floor( number )
```

The following table describes arguments for this function:

Argument	Description
number	Numeric value. Use a numeric expression.

Return Value

Integer.

NULL if a value passed to the function is NULL.

Example

The following expression returns the largest integer less than or equal to the value in BANK_BALANCE:

```
floor( BANK_BALANCE )
```

The following table contains example arguments and return values:

BANK_BALANCE	RETURN VALUE
39.79	39
NULL	NULL
-100.99	-101
5	5

lang

Returns TRUE if the element has an xml:lang attribute that is the same language as the code argument. Use the lang function to select XML by language. The xml:lang attribute is a code that identifies the language of the element content. An element might include text in several languages.

Syntax

```
lang ( code )
```

The following table describes arguments for this function:

Argument	Description
code	String datatype. Passes the element content language code.

Return Value

Boolean.

Example

The following expression examines the element content language code:

```
lang ( 'en' )
```

The following table contains example arguments and return values:

XML	RETURN VALUE
<Phrase xml:lang="es"> El perro esta en la casa. </Phrase>	FALSE
<Phrase xml:lang="en"> The dog is in the house. </Phrase>	TRUE

normalize-space

Removes leading and trailing white space from a string. White space contains characters that do not display, such as the space character and the tab character. This function replaces sequences of white space with a single space.

Syntax

```
normalize-space ( string )
```

The following table describes the argument for this function:

Argument	Description
string	String datatype. Passes a string that contains white space.

Return Value

String.

NULL if the string is NULL.

Example

The following expression removes excess white space from a name:

```
normalize-space ( NAME )
```

The following table contains example arguments and return values:

NAME	RETURN VALUE
Jack Dog	Jack Dog
Harry Cat	Harry Cat

not

Returns the inverse of a Boolean condition. The function returns TRUE if a condition is false, and returns FALSE if a condition is true.

Syntax

```
not ( condition )
```

The following table describes the argument for this function:

Argument	Description
condition	Boolean expression or value.

Return Value

Boolean.

NULL if the condition is NULL.

Example

The following expression returns the inverse of a Boolean condition:

```
not ( EMPLOYEE = concat ( FIRSTNAME, LASTNAME ))
```

The following table contains example arguments and return values:

EMPLOYEE	FIRSTNAME	LASTNAME	RETURN
Fullname	Full	Name	FALSE
Lastname	Lastname	First	TRUE

number

Converts a string or Boolean value to a number.

Syntax

```
number ( value )
```

The following table describes the argument for this function:

Argument	Description
value	Use a Boolean or string value.

Return Value

The function returns a number for the following data:

- A string converts to a number if the string contains a numeric character. The string can contain white space and include a minus sign followed by a number. White space can follow the number in the string. Any other string is Not a Number (NaN).
- A Boolean TRUE converts to 1. A Boolean FALSE converts to 0.

If a value passed as an argument to the function is not a number, the function returns Not A Number (NaN).

Example

The following expression converts payment to a number:

```
number ( PAYMENT )
```

The following table contains example arguments and return values:

PAYMENT	RETURN VALUE
‘850.00’	850.00
TRUE	1
FALSE	0
AB	NaN

round

Rounds a number to the nearest integer. If the number is between two integers, round returns the higher number.

Syntax

```
round ( number )
```

The following table describes the argument for this function:

Argument	Description
number	Numeric value. Passes a numeric datatype or an expression that results in a number.

Return Value

Integer.

Example

The following expression rounds BANK_BALANCE:

```
round( BANK_BALANCE )
```

The following table contains example arguments and return values:

BANK_BALANCE	RETURN VALUE
12.34	12
12.50	13
-18.99	-19
NULL	NULL

starts-with

Returns TRUE if the first string starts with the second string. Otherwise, returns FALSE.

Syntax

```
starts-with ( string, substring )
```

The following table describes the arguments for this function:

Argument	Description
string	String datatype. Passes the string to search. The string is case sensitive.
substring	String datatype. Passes the substring to search for in the string. The substring is case sensitive.

Return Value

Boolean.

Example

The following expression determines if NAME starts with FIRSTNAME:

```
starts-with ( NAME, FIRSTNAME )
```

The following table contains example arguments and return values:

NAME	FIRSTNAME	RETURN VALUE
Kathy Russell	Kathy	TRUE
Joe Abril	Mark	FALSE

string

Converts a number or Boolean to a string.

Syntax

```
string ( value )
```

The following table describes the argument for this function:

Argument	Description
value	Numeric or Boolean value. Passes a number or Boolean value.

Return Value

String.

Returns an empty string if no value is passed. Returns NULL if a null value is passed.

The string function converts a number to a string as follows:

- If the number is an integer, the function returns a string in decimal form with no decimal point and no leading zeros.
- If the number is not an integer, the function returns a string including a decimal point with at least one digit before the decimal point, and at least one digit after the decimal point.
- If the number is negative, the function returns a string that contains a minus sign (-).

The string function converts a Boolean to a string as follows:

- If the Boolean is FALSE, the function returns the string “false.”
- If the Boolean is TRUE, the function returns the string “true.”

Example

The following expression returns a string from the numeric argument speed:

```
string( SPEED )
```

The following table contains example arguments and return values:

SPEED	RETURN VALUE
10.99	‘10.99’

SPEED	RETURN VALUE
15.62567	'15.62567'
0	'0'
10	'10'
50	'50'
1.3	'1.3'

The following expression returns a string from the Boolean argument STATUS:

```
string( STATUS )
```

The following table shows example arguments and return values:

STATUS	RETURN VALUE
TRUE	'true'
FALSE	'false'
NULL	NULL

string-length

Returns the number of characters in a string, including trailing blanks.

Syntax

```
string-length ( string )
```

The following table describes the argument for this function:

Argument	Description
string	String datatype. The string you want to evaluate.

Return Value

Integer.

NULL if a value passed to the function is NULL.

Example

The following expression returns the length of the customer name:

```
string-length ( CUSTOMER_NAME )
```

The following table contains example arguments and return values:

CUSTOMER_NAME	RETURN VALUE
Bernice Davis	13

CUSTOMER_NAME	RETURN VALUE
NULL	NULL
John Baer	9

substring

Returns a portion of a string starting at a specified position. Substring includes blanks as characters in the string.

Syntax

```
substring ( string, start [ ,length ] )
```

The following table describes arguments for this function:

Argument	Description
string	String datatype. The string to search.
start	Integer datatype. Passes the position in the string to start counting. If the start position is a positive number, substring locates the start position by counting from the beginning of the string. The first character is one. If the start position is a negative number, substring locates the start position by counting from the end of the string.
length	Integer datatype. Must be greater than zero. Passes the number of characters to return in a string. If you omit the length argument, substring returns all of the characters from the start position to the end of the string.

Return Value

String.

When the string contains a numeric value, the function returns a character string.

If you pass a negative integer or zero, the function returns an empty string.

NULL if a value passed to the function is NULL.

Examples

The following expression returns the area code in PHONE:

```
substring( PHONE, 1, 3 )
```

PHONE	RETURN VALUE
809-555-3915	809
NULL	NULL

The following expression returns the PHONE without the area code:

```
substring ( phone, 5, 8 )
```

The following table includes example arguments and return values without the area codes:

PHONE	RETURN VALUE
808-555-0269	555-0269
NULL	NULL

You can pass a negative start value to start from the right side of the string. The expression reads the source string from left to right for the *length* argument:

```
substring ( PHONE, -8, 3 )
```

The following table includes example arguments and return values when the expression reads the source string from left to right:

PHONE	RETURN VALUE
808-555-0269	555
809-555-3915	555
357-687-6708	687
NULL	NULL

When the *length* argument is longer than the string, substring returns all the characters from the start position to the end of the string. For example:

```
substring ( 'abcd', 2, 8 )
```

returns ‘bcd.’

```
substring ( 'abcd', -2, 8 )
returns ‘cd.’
```

substring-after

Returns the characters in a string that occurs after a substring.

Syntax

```
substring-after ( string, substring )
```

The following table describes the arguments for this function:

Argument	Description
string	String datatype. Passes the string to search.
substring	String datatype. Passes a substring to search for in the string.

Return Value

String.

Empty string if the substring is not found.

NULL if a value passed to the function is NULL.

Example

The following expression returns the string of characters in PHONE that occurs after the area code (415):

```
substring-after ( PHONE, (415) )
```

The following table includes examples of arguments and return values:

PHONE	RETURN VALUE
(415)555-1212	555-1212
(408)368-4017	-
NULL	NULL
(415)366-7621	366-7621

substring-before

Returns the part of a string that occurs before a substring.

Syntax

```
substring-before ( string, substring )
```

The following table describes the arguments for this function:

Argument	Description
string	String datatype. Passes the string to search.
substring	String datatype. Passes the substring to search for in the string.

Return Value

String.

Empty string if the substring is not found.

NULL if a value passed to the function is NULL.

Example

The following expression returns the number that occurs in a Third Street address:

```
substring-before ( ADDRESS, Third Street )
```

The following table contains example attributes and return values:

ADDRESS	RETURN VALUE
100 Third Street	100
250 Third Street	250
600 Third Street	600

ADDRESS	RETURN VALUE
NULL	NULL

translate

Converts the characters in a string to other characters. The function uses two other strings as translation pairs.

Syntax

```
translate ( string1, string2, string3 )
```

The following table describes the arguments for this function:

Argument	Description
string1	String datatype. Passes the string to translate.
string2	String datatype. Passes the string that defines which characters to translate. Translate replaces each character in *string1* with a number indicating its position in *string2*.
string3	String datatype. Passes the string that defines what the characters from encrypted *string1* should translate to. Translate replaces each character in encrypted *string1* with a character in *string3* at the position number from *string2*.

Return Value

String.

Example

The following expression translates a string using two other strings:

```
translate ( EXPRESSION, STRING2, STRING3 )
```

The following table contains example arguments and return values:

EXPRESSION	STRING2	STRING3	RETURN VALUE
A Space Odissei	i	y	A Space Odyssey
rats	tras	TCAS	CATS
bar	abc	ABC	BAr

Translate does not change a character in EXPRESSION if the character does not occur in string2. If a character occurs in EXPRESSION and string2, but does not occur in string3, the character does not occur in the returned string.

true

Always returns TRUE. Use this function to set a Boolean to TRUE.

Syntax

```
true ()
```

The true function does not accept arguments.

Return Value

Boolean TRUE.

Example

The following table contains example expressions that return TRUE:

EXPRESSION	RETURN VALUE
`( decision ) = true ()`	TRUE
`A/B = true ()`	TRUE
`( starts-with ( name, 'T' ))= true`	TRUE

INDEX

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

X