3 Creating Data Descriptor Files

Data descriptor files define the relationship between physical data fields in source or target application data and the logical ATS definition in a context map (CXM) file. In the ATS definition, each data element corresponds to an attribute of a logical ATS entity. Therefore, the data descriptor file syntax uses XML tags called **Entity** and **Attribute** to capture the relationship between application data fields and logical data elements.

Data descriptor files provide the ATS Manager component of the Interoperability Server with information about the internal structure of application data files and the correct mapping between data file fields and the data elements in the application schema.

During an interoperability run, the ATS Manager uses source data descriptor files to import application data into source ATS instances in the internal work space. At the end of the run, the ATS Manager uses target data descriptor files to export data from a target ATS instance in the internal work space into a file with target data in the correct format.

There are two types of descriptor files: one for application data in XML format, and the other for application data in delimited flat files. Each is in XML format; while similar in structure they have different DTDs.

This discussion covers the following topics:

- Analyzing and Describing XML Data
- Analyzing and Describing Flat-file Data

Each of these sections starts with sample data and describes how to create a data descriptor for that sample data. You can use any data descriptor file for either source application data or target application data. For source data, the ATS Manager uses the descriptor information to parse source data files on import. For target data, the ATS Manager uses the descriptor information to construct target data files on export.

Analyzing and Describing XML Data

This section describes the following steps for creating a data descriptor for XML data:

- Understanding the Data Formats
- Identifying the Entities
- Identifying the Attributes for Each Entity

For a complete description of the syntax of XML descriptors, see "Data Descriptors for XML Data" on page 69.

Understanding the Data Formats

Example 1 shows sample data from an accounting system. The rest of this section explains how to create a data descriptor file that describes the format of this data.

Example 1: Sample XML Data

```
<userData>
    <users>
        <employee>
            <lastName>Johnson</lastName>
            <firstName>Lee</firstName>
            <employeeID>7892213</employeeID>
            <costCenterInfo>
                <costCenterCode>043-032</costCenterCode>
                <costCenterDescr>Operations</costCenterDescr>
            </costCenterInfo>
        </employee>
        <user>
            <employeeID>7892213</employeeID>
            <userName>ljohnson1</userName>
            <roleCode>483</roleCode>
            <createDate>
                <day>29</day>
                <month>03</month>
                <year>2001</year>
            </createDate>
        </user>
    </users>
</userData>
```

The sample data contains two entities: **employee** and **user**. Most of the attributes of each entity are subelements of that entity. Each entity also has attributes that are more deeply nested, grouped together in another element:

- The **employee** entity has an element called **costCenterInfo** that contains two attributes, **costCenterCode** and **costCenterDescr**.
- The **user** entity has an element called **createDate** that contains three attributes, the **day**, **month**, and **year** parts of the date.

Using Tag Paths

To describe the location of the XML element that corresponds to a particular entity or attribute, you provide a *tag path*. The tag path looks like a file path, and lists the levels of nesting you must traverse to find that element.

For example, in Example 1, the tag path to the **employee** element is:

```
userData/users/employee
```

In some application data files, the data for attributes corresponding to an entity is not immediately inside the element for that entity, but is instead nested more deeply inside that element. For example, consider the **employee** data in Example 1:

```
<userData>
   <users>
      <employee>
         <lastName>Johnson</lastName>
         <firstName>Lee</firstName>
         <employeeID>7892213</employeeID>
         <costCenterInfo>
            <costCenterCode>043-032</costCenterCode>
            <costCenterDescr>Operations</costCenterDescr>
         </costCenterInfo>
      </employee>
      ...
   </users>
</userData>
```

In this example the **costCenterCode** and **costCenterDescr** appear inside the **costCenterInfo** element, one level more deeply nested than the other attributes of **employee**.

You can use a special character, **#**, in your tag paths to preserve the nesting when the ATS Manager exports (serializes) **employee** data. In other words, if you set the tag paths as follows:

costCenterCode	**userData/users/employee/costCenterInfo/costCenterCode**
costCenterDescr	**userData/users/employee/costCenterInfo/costCenterDescr**

Then the exported data would look like:

```
<userData>
   <users>
      <employee>
         <lastName>Johnson</lastName>
         <firstName>Lee</firstName>
         <employeeID>7892213</employeeID>
         <costCenterInfo>
            <costCenterCode>043-032</costCenterCode>
         </costCenterInfo>
```

```
            <costCenterInfo>
               <costCenterDescr>Operations</costCenterDescr>
            </costCenterInfo>
         </employee>
         ...
      </users>
</userData>
```

Using the **#** in the tag path instead, you would end up with:

costCenterCode	**userData/users/employee/costCenterInfo#costCenterCode**
costCenterDescr	**userData/users/employee/costCenterInfo#costCenterDescr**

This format would combine the data for **costCenterCode** and **costCenterDescr** into a single **costCenterInfo** element.

Identifying the Entities

The format for data descriptors for XML data contains a tag called **Entity** that describes each logical entity in your data structure. The accounting system contains two entities, **employee** and **user**.

For each of these entities, you start by specifying where that entity appears in the ATS definition for your context map and how to identify records associated with that entity. For each **Entity** element, you also supply the following information:

- The name of the entity as it appears in the ATS definition in the context map file you are using. To specify this, use the **name** attribute of the **Entity** element.
- Where to find the beginning of this entity within the XML structure. To specify this, you add an **EntityLocation** element inside the **Entity** element with a **tagPath** attribute.

  The tag path shows how to navigate the hierarchy of XML tags. It looks like a file path. For example, in the accounting example, the **employee** entity is a sub-element of **users**, which is a sub-element of **userData**. Therefore, the tag path to find **employee** is **userData/users/employee**.
- Where to find the data for this entity. To do this you add an **EntityInstance** element inside the **Entity** element. In this element, you add two kinds of information:
  - An **EntityInstanceLocation** element with a **tagPath** attribute to point to the beginning of the record of data.
  - A separate **Attribute** element for each attribute of this entity. For more information, see "Identifying the Attributes for Each Entity" on page 31.

Therefore, the following XML code identifies the entities in the accounting system data:

```
<!DOCTYPE XMLDescriptor SYSTEM "xmldescriptor.dtd">
<XMLDescriptor>
    <Entity name="employee">
        <EntityLocation tagPath="userData/users/employee" />
        <EntityInstance>
            <EntityInstanceLocation
                tagPath="userData/users/employee" />
                ...
        </EntityInstance>
    </Entity>
    <Entity name="user">
        <EntityLocation tagPath="userData/users/user" />
        <EntityInstance>
            <EntityInstanceLocation
                tagPath="userData/users/user" />
                ...
        </EntityInstance>
    </Entity>
</XMLDescriptor>
```

The next step is to fill in information about where to find the data for each attribute of each entity.

Identifying the Attributes for Each Entity

Inside each **EntityInstance** element, you must add an **Attribute** element for each attribute of the entity with the name of the corresponding data element in the ATS definition. In addition, inside each **Attribute** element you place an **AttributeLocation** element with the tag path to the location of that attribute in the XML data file. For information about tag paths and the special character **#**, see "Using Tag Paths" on page 29.

Describing the Attributes of the employee Entity

The **Attribute** elements for **lastName**, **firstName**, and **employeeID** are:

```
<Attribute name="lastName">
    <AttributeLocation
        tagPath="userData/users/employee/lastName" />
</Attribute>
<Attribute name=firstName">
    <AttributeLocation
        tagPath="userData/users/employee/firstName" />
</Attribute>
<Attribute name="employeeID">
    <AttributeLocation
        tagPath="userData/users/employee/employeeID" />
</Attribute>
```

Because the **costCenterCode** and **costCenterDescr** are nested inside the **costCenterInfo** element, you can use the special character **#** in the tag path for these attributes:

```
<Attribute name="costCenterCode">
    <AttributeLocation
       tagPath="userData/users/employee/costCenterInfo#costCenterCode" />
</Attribute>
<Attribute name="costCenterDescr">
    <AttributeLocation
       tagPath="userData/users/employee/costCenterInfo#costCenterDescr" />
</Attribute>
```

Describing the Attributes of the user Entity

The **Attribute** elements for **employeeID**, **userName**, and **roleCode** are:

```
<Attribute name="employeeID">
    <AttributeLocation
       tagPath="userData/users/user/employeeID" />
</Attribute>
<Attribute name=userName">
    <AttributeLocation
       tagPath="userData/users/user/userName" />
</Attribute>
<Attribute name="roleCode">
    <AttributeLocation
       tagPath="userData/users/user/roleCode" />
</Attribute>
```

Because the **day**, **month**, and **year** are nested inside the **createDate** element, you can use the special character **#** in the tag path for these attributes:

```
<Attribute name="createDate-day">
    <AttributeLocation
       tagPath="userData/users/user/createDate#day" />
</Attribute>
<Attribute name="createDate-month">
    <AttributeLocation
       tagPath="userData/users/user/createDate#month" />
</Attribute>
<Attribute name="createDate-year">
    <AttributeLocation
       tagPath="userData/users/user/createDate#year" />
</Attribute>
```

Analyzing and Describing Flat-file Data

This section describes the following steps for creating a data descriptor for flat-file data:

- Understanding the Data Formats
- Identifying the Entities
- Identifying the Attributes for Each Entity

For a complete description of the syntax of flat-file descriptors, see "Data Descriptors for Flat-file Data" on page 78.

Understanding the Data Formats

Example 2 shows sample data from an accounting system. The rest of this section explains how to create a data descriptor file that describes the format of this data.

Example 2: Sample Flat-file Data

```
EP001Johnson/Lee7892213043-032Operations
EP001Davis/Terry8323023043-132Maintenance
EP001Franklin/Dana3820139233-202Accounting
EP0027892213,ljohnson1,483,20010329
EP0028323023,tdavis23,421,19981202
EP0023820139,dfranklin,291,20020113
```

It is not clear just from looking at this file what its structure is. Discussions with domain experts reveal the following information about the sample data:

- Two entities are represented: **Employee** and **User_Audit**.
- The first three lines of data are **Employee** records.
- The last three lines of data are **User_Audit** records.
- The **Employee** entity has the following attributes:
  - The first five characters identify which entity this data belongs to.
  - The characters from the end of the entity indicator to the next slash (/) character represent the employee's last name, which appears in the ATS definition in the context map as **Last_Name**.
  - The next group of alphabetic characters represent the employee's first name, which is **First_Name** in the ATS definition.
  - The next seven characters represent the employee number, which is **Employee_ID** in the ATS definition.
  - Following the employee number is a number that represents the cost center assigned to the employee, which is **Cost_Center** in the ATS definition. The cost center number is three digits, a dash, and three more digits.
  - The text from the end of the cost center to the end of the record is the name of the cost center, which is **Cost_Center_Name** in the ATS definition.

- The **User_Audit** entity has the following attributes, separated by commas:
  - The first five characters again identify which entity the data belongs to.
  - The characters from the beginning of the line to the first comma represent an employee number, which is again **Employee_ID** in the ATS definition.
  - The characters up to the next comma represent the name of a user of the accounting system, which is **Username** in the ATS definition.
  - The characters up to the next comma identify this user's role in using the accounting system, which is **Role_Code** in the ATS definition.
  - The characters from here to the end of the line indicate the date on which this record was created, which is **Create_Date** in the ATS definition.

Identifying the Entities

The XML format for flat-file data descriptors contains a tag called **Entity** that describes each logical entity in your data structure. The accounting system contains two entities, **Employee** and **User_Audit**.

For each of these entities, you start by specifying where that entity appears in the ATS definition for your context map and how to identify records associated with that entity. The **Entity** element has three attributes you can use to do this:

- The **name** attribute specifies the name of the logical ATS entity corresponding to this physical entity in the ATS definition.
- The **entityIndicator** attribute specifies a text string in the data that identifies data for a particular entity.

  Based on the analysis of the sample data, the string **EP001** identifies **Employee** data and the string **EP002** identifies **User_Audit** data.
- The **delimiter** attribute specifies one or more characters that signal the end of data for each attribute of this entity type. This delimiter is global; you can override it for each attribute.

Each **Entity** element contains an element called **EntityInstance**, which contains one or more **Attribute** elements. The **EntityInstance** element has no attributes.

Therefore, the following XML code identifies the entities in the accounting system data file:

```
<!DOCTYPE FlatFileDescriptor SYSTEM "ffdescriptor.dtd">
<FlatFileDescriptor>
   <Entity name="Employee" entityIndicator="EP001">
      <EntityInstance>
         ...
      </EntityInstance>
   </Entity>
```

```
    <Entity name="User_Audit" entityIndicator="EP002"
           delimiter=",">
        <EntityInstance>
            ...
        </EntityInstance>
    </Entity>
</FlatFileDescriptor>
```

The next step is to fill in information about how to identify the data corresponding to each attribute of each entity.

Identifying the Attributes for Each Entity

Inside the **EntityInstance** element for each **Entity** element, you add an **Attribute** element for each attribute of that entity. The **Attribute** element serves two purposes:

- It links this attribute to a logical attribute of an entity in the ATS definition.
- It describes how to identify data associated with that attribute.

The **Attribute** elements must appear in the **Entity** element in the same order as the data for each attribute appears in the file.

You use a combination of the attributes on the **Attribute** element and the position subelement and its attributes to describe how to parse the data for each attribute. For a complete list of the attributes of these elements, see "Attribute Element" on page 83 and "position Element" on page 84.

Using the **position** element you can specify the physical location in an entity record of the data corresponding to the parent **Attribute** element. The value can be a fixed offset from the beginning of the line or a relative offset from the end of the last attribute found.

Using combinations of the attributes for this element, you can describe how to identify the corresponding data in a number of ways, including:

- Start at a specific character position and match a particular pattern.
- Start at a specific character position and read up to a particular delimiter.
- Read a specific number of characters, beginning immediately after the end of the preceding attribute.

The following sections show examples of some of these options.

Describing the Attributes of the Employee Entity

Consider again the first **Employee** record in the sample data:

```
EP001Johnson/Lee7892213043-032Operations
```

Based on this example, the following sections explain how to write **Attribute** elements to describe each of the attributes of **Employee**.

Describing Last_Name

Following the string that identifies this entity, the first attribute is the employee's last name. It appears as a text string terminated by a slash (/). To identify data for the **Last_Name** attribute, you read the characters up to the delimiter character.

Therefore, the following XML element describes the **Last_Name** attribute:

```
<Attribute name="Last_Name" delimiter="/" />
```

Describing First_Name

The first name follows the last name, and ends at the beginning of the employee number. The easiest way to identify the entire first name is to say "use all of the characters from here until you encounter a number."

You can express this in a regular expression. The **matches** attribute of the **Attribute** element lets you provide a regular expression that describes the format of the data for that attribute. In this case, the regular expression **/D+** says to search for characters that are not digits and continue until you find something that does not match.

The following XML element describes the **First_Name** attribute:

```
<Attribute name="First_Name" matches="/D+" />
```

Using Regular Expressions

The regular expression parser in the Interoperability Server recognizes the following ways to describe text to match.

Special Characters:

\	Used as an "escape" character before another special character, to negate the special meaning of the character.
\\	Matches a single backslash character.
\t	Matches an AXCII tab character.
\n	Matches a newline character.
\r	Matches a return character.
^	Looks for matches only at the beginning of a line.
$	Looks for matches only at the end of a line.

Groups of Characters:

[abc]	Matches any of the characters inside the brackets.
[a-zA-Z]	Matches any of the characters in the ranges in the brackets.
[^xyz]	Matches any characters *not* in the brackets.
\d	Matches any digit; equivalent to [0-9].

Using Regular Expressions (Continued)	
\D	Matches any non-digit; equivalent to [^0-9].
\s	Matches any whitespace character, including tabs, spaces, returns, and newlines.
\S	Matches any non-whitespace character.
\w	Matches any alphanumeric (“word”) character, including _; equivalent to [a-zA-Z0-9_].
\W	Matches any non-alphanumeric character.
Repeated Characters:	
*	Matches the preceding character zero or more times.
+	Matches the preceding character one or more times.
?	Matches the preceding character zero or one time.

Describing Employee_ID

The employee number is seven digits long and follows the employee’s first name. To specify the length of an attribute, you add a **position** element inside the **Attribute** element. The position element has attributes that let you specify where the data starts and how many characters long it is. In this case, you only need the **length** attribute. For a complete list of attributes of the position element, see “position Element” on page 84.

The following XML element describes the **Employee_ID** attribute:

```
<Attribute name="Employee_ID">
   <position length="7" />
</Attribute>
```

Describing Cost_Center

The cost center number is three digits, a dash, and three more digits. Because it is always seven characters long, you can use a **position** element with a **length** attribute to describe the data for **Cost_Center**.

The following XML element describes the **Cost_Center** attribute:

```
<Attribute name="Cost_Center">
   <position length="7" />
</Attribute>
```

Describing Cost_Center_Name

The name of the cost center starts immediately after the cost center number and goes to the end of the line. Therefore, the only thing you need to specify for this attribute is its name.

The following XML element describes the **Cost_Center_Name** attribute:

```
<Attribute name="Cost_Center_Name" />
```

You have now completely described the **Employee** entity:

```
<Entity name="Employee" entityIndicator="EP001">
   <EntityInstance>
      <Attribute name="Last_Name" delimiter="/" />
      <Attribute name="First_Name" matches="/D+" />
      <Attribute name="Employee_ID">
         <position length="7" />
      </Attribute>
      <Attribute name="Cost_Center">
         <position length="7" />
      </Attribute>
   </EntityInstance>
</Entity>
```

Describing the Attributes of the User_Audit Entity

Now consider the first **User_Audit** record in the sample data:

```
EP0027892213,ljohnson1,483,20010329
```

When you defined the **Entity** element for **User_Audit**, you specified a comma (**,**) as the default delimiter for attributes of that entity. Notice that the format of the data in this record is uniform—every attribute but the last one ends with a comma. Therefore, the only thing you need to specify is the name of each attribute:

```
<Attribute name="Employee_ID" />
<Attribute name="Username" />
<Attribute name="Role_Code" />
```

The last attribute is a date, so you can also use the **dataType** and **mask** attributes of the **Attribute** element to specify the format of the date. The following XML element describes the **Create_Date** attribute:

```
<Attribute name="Create_Date" dataType="date" mask="YYYYMMDD"/>
```

For more information, see the next section, "About Masks."

You have now completely described the **User_Audit** entity:

```
<Entity name="User_Audit" entityIndicator="EP002">
   <EntityInstance>
      <Attribute name="Employee_ID" />
      <Attribute name="Username" />
      <Attribute name="Role_Code" />
      <Attribute name="Create_Date" dataType="date"
         mask="YYYYMMDD"/>
   </EntityInstance>
</Entity>
```

About Masks

Format masks are specific patterns that specify how numeric data and dates are formatted in your data files. You can specify format masks for any attribute whose data type is **integer**, **decimal**, **float**, or **date**. The Interoperability Server uses this information to validate the formats when it exports data from the internal work space to target application data files.

Masks for Numbers

Format masks for numeric data types use combinations of the following symbols:

0	a digit
#	a digit, zero shows as absent
.	placeholder for decimal separator
,	placeholder for grouping separator
;	separates formats
-	default negative prefix
%	multiply by 100 and show as percentage
?	multiply by 1000 and show as per mille
¤	currency sign, replaced by currency symbol; if doubled, replaced by international currency symbol; if present in a pattern, the monetary decimal separator is used instead of the decimal separator
X	any other characters can be used in the prefix or suffix
'	used to quote special characters in a prefix or suffix

The following table shows examples of format masks and what each one produces:

Value	Mask	Produces
123456.789	###,###.###	123,456.789
123456.789	###.##	123456.79
123.45	000000.000	000123.450
123.456	$#####.######	$123.456

Masks for Dates

Format masks for dates use combinations of the following symbols:

y	year
M	month in year; **M** produces a number (**3**), **MM** produces a number with leading zeroes (**03**), and **MMM** produces a word in English (**March**)
d	day in month
h	hour, using 12-hour clock
H	hour, using 24-hour clock (0 is midnight)
m	minute in hour
s	second in minute
S	millisecond

The following table shows examples of format masks for dates and how each one represents the date March 7, 2002:

Format	Produces
MM/DD/YYYY	03/07/2002
DDMMYYYY	07032002
YYYYMMDD	20020307