9 Using the Modulant Curtana Data Converter

The Modulant Curtana Data Converter lets you specify additional ways to transform the data associated with an ATS schema. Some fields in standard databases do not have direct counterparts in the Abstract Conceptual Model, such as dates and some names. Before you can map a date field, you must split it into the day, month, and year parts of the date. Then you can map each part separately.

The Modulant Curtana Data Converter makes this possible. When your source ATS contains composite fields, the Data Converter splits them into their component parts so that the Modulant Curtana Transformation Engine can properly populate the Abstract Conceptual Model. Then after the Transformation Engine extracts the data for output, the Data Converter recombines the fields as needed by the target ATS.

Note: The Data Converter requires you to use Oracle for the internal database.

You specify what you want the Data Converter to do in the XMP (Transformation MaP) file, which also stores the mapping information for an ATS schema. The following topics describe the Data Converter and how to modify the XMP file to specify the necessary conversions:

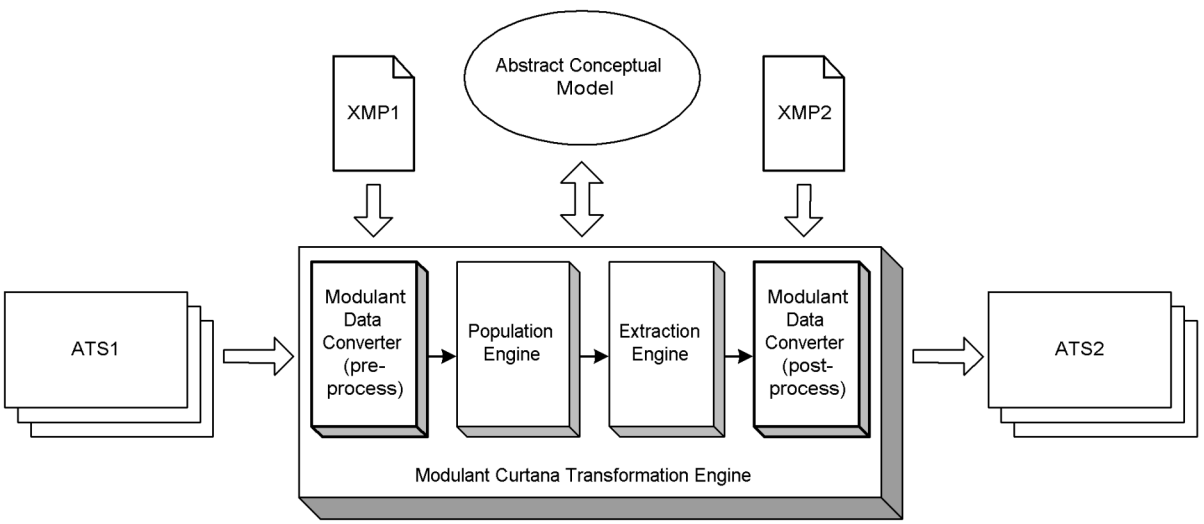
What is the Modulant Curtana Data Converter?

The Modulant Curtana Data Converter is part of the Modulant Curtana Transformation Engine. It has two parts, a preprocessor and a postprocessor:

- If the XMP file corresponding to an input ATS schema contains data conversion rules, the Transformation Engine calls the Data Converter before running the Population Engine to populate the Abstract Conceptual Model with data from the source ATS.
- If the XMP file corresponding to the output ATS schema contains data conversion rules, the Transformation Engine calls the Data Converter after running the Extraction Engine to output data from the Abstract Conceptual Model to the target ATS.

Figure 64 shows the role of the Data Converter in the overall flow of a transformation run.

Figure 64: Where the Data Converter Fits



What Does the Modulant Curtana Data Converter Do?

The Modulant Curtana Data Converter can perform three types of conversions, all of which can operate on only a single table:

- *Extraction:* Extract substrings from one column to another (for example, breaking dates into day, month, and year components, or breaking names into separate first name and last name parts).

  This operation extracts part of the data in a column and places it in a new column. When you define extractions in the preprocessor phase, the Data Converter creates a new column in the internal database to hold the result. In the postprocessor phase, the Data Converter places the final result of an extraction operation into a field of the target ATS.

  You can specify this extraction in two ways:

  - by index position in the column
  - by delimiter character

- *Concatenation:* Concatenate separate columns to form a single column (for example, recombining dates or names).

  This operation extracts strings from multiple columns, concatenates them, and places the result into a new column. When you define concatenations in the preprocessor phase, the Data Converter creates a new column in the internal database to hold the result. In the postprocessor phase, the Data Converter places the final result of a concatenation operation into a field of the target ATS.

- *Calculation:* Perform arithmetic operations (addition, subtraction, multiplication, division) on one or more numeric fields (for example, to create derived attributes such as a total price, by calculating the sales tax on a price and adding the tax to the price).

  This operation performs the requested operations on values from multiple columns and places the result into a new column. When you define arithmetic calculations in the preprocessor phase, the Data Converter creates a new column in the internal database to hold the result. In the postprocessor phase, the Data Converter places the final result of a calculation operation into a field of the target ATS.

When you identify fields in your source data that require any of these types of conversion, you must use the Mapping Tool to create data elements for each of the "virtual" columns that the Data Converter will create in its internal database during a transformation run, and map each of these data elements separately. For instructions, see "Defining and Mapping Virtual Fields" on page 179.

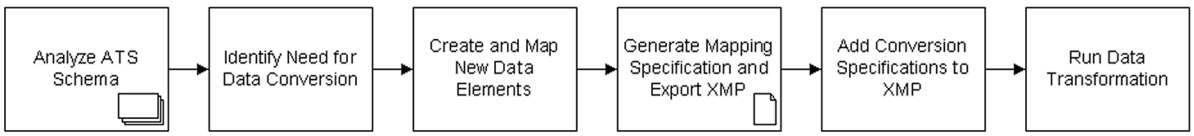
You specify which types of conversions you want the Data Converter to perform on your ATS schema in the **dataConverter** section of the XMP file. For instructions, see "Specifying Data Conversions" on page 181.

Planning for Data Conversion

As you develop your mapping strategy, you might discover that your ATS schema contains fields for which there is no direct mapping target in the Abstract Conceptual Model. For each of these fields, analyze the structure to determine how you want to map it.

Figure 65 shows the steps you take to implement data conversion, beginning with analyzing your ATS schema. The following sections describe the remaining phases.

Figure 65: Implementing Data Conversion



Identifying Extraction Conversions

For example, if you have a field that contains dates, you must use different mapping targets for the month part, the day part, and the year part. Therefore, you must have a separate data element for each part before you can map it. In cases like this, you define an extraction conversion to create each of the individual fields during the transformation process.

Identifying Concatenation Conversions

As another example, suppose your ATS schema stores the year and the century in separate fields. Because the Abstract Conceptual Model only has an attribute for mapping years (see "Date and Time" on page 77), you must combine the century and the year information into a single data element that you can map. In this case, you define a concatenation conversion to create a composite century-and-year field.

Identifying Arithmetic Conversions

In addition, your ATS schema might have a group of fields from which you derive a value that your application needs. If you want to map the derived field separately, you can use the data converter to specify an arithmetic calculation.

For example, you might have information about sales figures, from which you calculate a salesperson's commission. Rather than store the commission amount in the database, your application calculates it separately. But in this case, you might want to map the commission value to its own mapping target in the Abstract Conceptual Model. To do this, you define an arithmetic conversion.

Defining and Mapping Virtual Fields

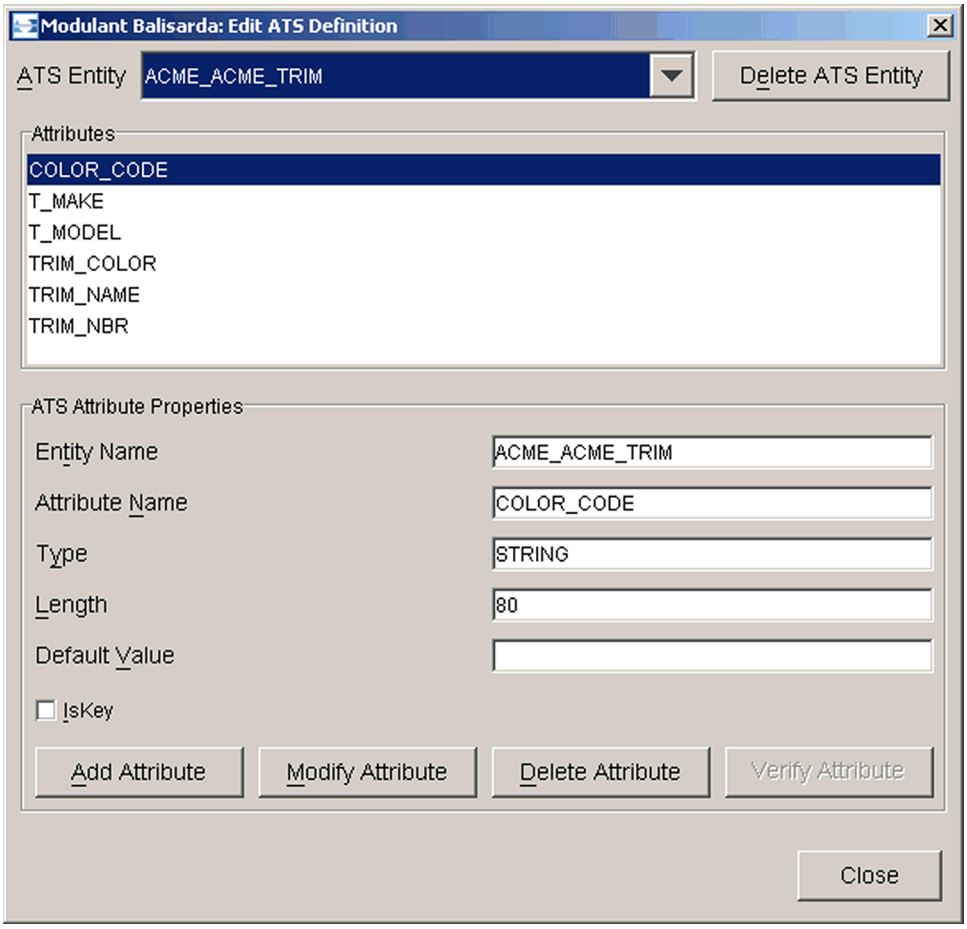
Once you have identified the fields in your source ATS schema that require conversion, and you have chosen the mapping targets you want to use, you must use the Mapping Tool to define new data elements that correspond to the "virtual" columns the Data Converter will create in its internal database during the transformation run.

To define data elements corresponding to virtual fields:

1 To create a new attribute:

a Select **Edit > ATS Definition**.

The **Edit ATS Definition** dialog box appears:



b From the **ATS Entity** list at the top, select the ATS entity associated with this conversion.

c In the **Attribute Name** field, type the name of the new data element.

Tip: If you are defining a new data element to hold the result of an extraction from an existing data element, select the existing data element first. This lets you add a suffix to the current name; for example, **DATE** can become **DATE_MM**, to indicate the month part.

d Make any necessary changes to the remaining properties of the new data element in the **Type**, **Length**, **Default Value**, and **IsKey** fields.

e Click **Add Attribute**, and then click **Close**.

2 To add the new data element to a data element group:

a Select **Mapping > Add > Data Element**.

b In the **Add Data Element** dialog box, select the data element group.

c In the **Data Element Name** field, type the name of the new data element.

Tip: To see exactly how you spelled the new data element's name, you can select it from the **ATS Attribute for Data Element** drop-down list before you type it.

d From the **ATS Entity for Data Element** and **ATS Attribute for Data Element** drop-down lists, select the corresponding entity and attribute names.

e In the **Data Element Mapping ID** field, type a mapping ID for the new data element.

Tip: To determine the next available mapping ID for this data element group, scroll through the list of data elements at the top. Use the same prefix for the new mapping ID, and assign the next sequential number. Prefixes are case-sensitive.

f Leave the **Data Element Mapping Usage** set to the default value **Input/Output**.

g Click **Apply**, and then click **Close**.

3 To map each new data element:

a Select **Mapping > Add > Mapping Target**.

b Specify the mapping target, as described in Chapter 7, "Mapping Data Elements."

c Complete the structure mapping, as described in "Defining Structure Mappings" on page 157.

4 Map the remaining data elements in your ATS schema.

5 Connect the data elements to the structure mappings, as described in Chapter 8, "Defining the Transformation."

6 Select **Tools > Generate Transformation Map** to validate the mapping specification.

7 Select **File > Export > XMP** to create an XMP file with mapping information.

Once you have an XMP file, you can specify the data conversion information for the Transformation Engine.

WARNING: If you import an XMP file that contains information in the **dataConverter** section, the Mapping Tool only preserves the data conversion specifications if you export mapping information to the same XMP file. If you export to a different file, you must copy the **dataConverter** section and paste it to the newly exported file.

Specifying Data Conversions

To define data conversion specifications, you add conversion definitions to the **dataConverter** section of the XMP file, after the **mappingStatement** elements. The Data Converter executes the conversions in the order in which the conversion definitions appear in the XMP file.

Note: As you add conversion specifications, be careful not to make any changes to the **mappingStatemement** elements generated by the Mapping Tool.

Defining "Cascading" Conversions

Depending on the complexity of your database, you might encounter situations where you want to define a data conversion that creates one result, and then use that result in a subsequent conversion operation.

For example, suppose your database stores hotel rates in U.S. dollars and airline fares in Swiss francs. To find out how much a person's total expenses will be, you must first convert the airline fare to dollars. Then you can add up the rates to compute the total. To do this, you define two conversions in the XMP file: first the currency conversion, followed by the calculation.

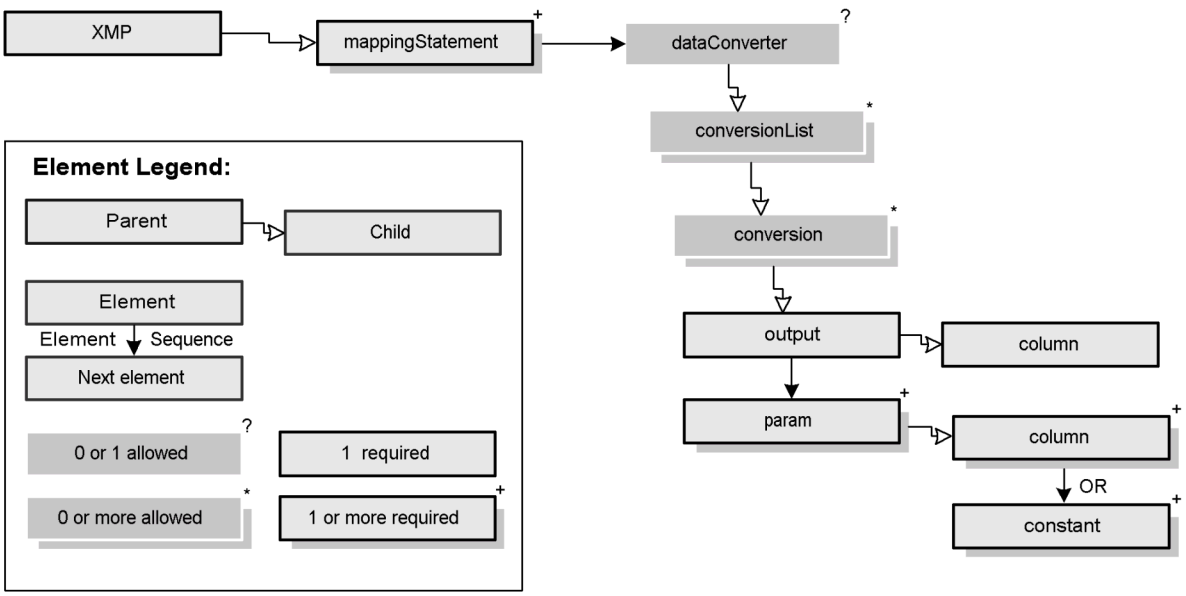
If the goal is to derive the result of the calculation, you might only need to define a new data element for the end result and map that data element. Or, depending on the structure of your data, you might determine that it is helpful to define mapping targets for both the intermediate field and the end result.

Anatomy of the dataConverter Element

An XMP file can have only one **dataConverter** element, following the **mappingStatement** elements. Inside this element, you define **conversionList** elements, each of which specifies a set of conversions you want the Data Converter to perform.

Figure 66 shows the structure of the **dataConverter** element as it occurs in the XMP file, based on the DTD (document type definition). For a full description of the XMP file, see Appendix B, "XMP Reference."

Figure 66: Structure of the dataConverter Element



The Data Converter expects two **conversionList** elements, one for conversions to perform on the input data from a source ATS during the preprocessing phase (the **PREPROCESSOR** list), and one for conversions to perform on the output data to the target ATS during the postprocessing phase (the **POSTPROCESSOR** list).

Example 1 shows the standard **conversionList** elements and how they appear in the XMP file.

Example 1: Standard conversionList Elements

```
<xmp>
    ...
    <dataConverter>
        <conversionList name="PREPROCESSOR">
        </conversionList>
        <conversionList name="POSTPROCESSOR">
        </conversionList>
    </dataConverter>
</xmp>
```

Note: The Data Converter ignores any **conversionList** elements whose **name** attribute has a value other than **PREPROCESSOR** or **POSTPROCESSOR**. These values are case-sensitive.

Testing Mappings that Require Data Conversion

As a general practice, the best way to test a mapping specification is to perform a transformation run with the same ATS data as both the source and the target of the transformation. In this case, you use the same XMP file to specify the mappings for both the source ATS and the target ATS.

If your ATS schema contains fields that require data conversion during either the preprocessing phase or the postprocessing phase, you must account for both phases as you do your initial testing.

In other words, for this test transformation run, if you perform extractions to split some fields in the preprocessor phase before running the Population Engine, you must define concatenations that recombine those fields during postprocessing after running the Extraction Engine.

To define a data conversion, you add one or more **conversion** elements to a **conversionList**. Each **conversion** element defines one conversion operation, using a set of attributes and two subelements that define the conversion itself:

- One **output** element that specifies the column to hold the result of the conversion.

  For preprocessor conversions and intermediate results of cascading conversions, this element refers to a column that the Data Converter will create. For the final results of postprocessor conversions, this element refers to a column in the target ATS.
- One or more **param** elements that define the parameters for the conversion.

In the **conversion** element, use the **type** attribute to specify this type of extraction: **EXTRACTSUBSTR**, **EXTRACTDELIM**, **CONCAT**, or **ARITHMETIC**. Note that these values of the **type** attribute must be in upper-case letters.

The **conversion** element also contains an attribute that defines the name of the database table that contains the fields to convert.

For example, the following **conversion** element defines a substring extraction:

```
<conversion type="EXTRACTSUBSTR" table="Employee"
</conversion>
```

The following sections describe the ways to create conversion definitions:

- Defining Substring Extractions
- Defining Data Concatenations
- Defining Arithmetic Operations

Defining Substring Extractions

A substring extraction separates part of the data in a field into a separate column in the internal database. For example, you use substring extractions to split date fields into month, day, and year components. You can also use this type of conversion to split name fields into first and last names.

The Data Converter provides two types of string extraction:

- Extraction by Index Position

  Use extraction by index position if you know the offset position in a string where you want to split a field.

- Extraction by Delimiter

  Use extraction by delimiter if you want to split a field after a particular character or character string. In this type of conversion, the delimiter can be more than a single character.

Extraction by Index Position

In the **conversion** element, you specify this type of extraction as **EXTRACTSUBSTR**:

```
<conversion type="EXTRACTSUBSTR" ...>
```

Inside the **conversion** element, you provide the following elements:

- One **output** element that specifies the column to hold the result of the conversion.
- One or more **param** elements, each of which has one of the following **name** attributes:
  - **name="input"**: the field that contains the data to split

    Use a **column** element with a **name** attribute to specify the name of the input field.
  - **name="start"**: the starting position of the string to extract

    Use a **constant** element with a **value** attribute to specify the position.

    Note: The Data Converter counts index positions starting with 1.
  - **name="length"**: the number of characters to extract

    Use a **constant** element with a **value** attribute to specify how many characters.

Note: The values of the **name** attribute must be in all lower-case letters.

Table 8 shows an example of a date field and the day, month, and year fields after extraction. Because the month and day each always have two digits, you can define each substring to extract based on its starting position in the original field.

Table 8: Example of Substring Extraction by Position

Input	Output		
HDate	**HDay**	**HMonth**	**HYear**
'07/09/2001'	9	7	2001

Example 2 shows **conversion** elements that define this extraction.

Example 2: Substring Extraction by Position

```
<conversion type="EXTRACTSUBSTR" table="Employee">
    <output>
        <column name="HMonth" type="varchar(2)"/>
    </output>
    <param name="input">
        <column name="HDate"/>
    </param>
    <param name="start">
        <constant value="1"/>
    </param>
    <param name="length">
        <constant value="2"/>
    </param>
</conversion>
<conversion type="EXTRACTSUBSTR" table="Employee">
    <output>
        <column name="HDay" type="varchar(2)"/>
    </output>
    <param name="input">
        <column name="HDate"/>
    </param>
    <param name="start">
        <constant value="3"/>
    </param>
    <param name="length">
        <constant value="2"/>
    </param>
</conversion>
<conversion type="EXTRACTSUBSTR" table="Employee">
    <output>
        <column name="HYear" type="varchar(4)"/>
    </output>
    <param name="input">
        <column name="HDate"/>
    </param>
```

```
    <param name="start">
        <constant value="7"/>
    </param>
    <param name="length">
        <constant value="4"/>
    </param>
</conversion>
```

Extraction by Delimiter

In the **conversion** element, you specify this type of extraction as **EXTRACTDELIM**:

```
<conversion type="EXTRACTDELIM" ...>
```

Inside the **conversion** element, you provide the following elements:

- One **output** element that specifies the column to hold the result of the conversion.
- One or more **param** elements, each of which has one of the following **name** attributes:
  - **name="input"**: the field with the data to split

    Use a **column** element with a **name** attribute to specify the input field.
  - **name="delim"**: the delimiter string (one or more characters long) that separates the strings to be extracted

    Use a **constant** element with a **value** attribute to specify the string.
  - **name="token"**: specifies which extracted string goes in which new field; in other words, when the value of **token** is **1**, the first substring (before the first delimiter) goes in the field defined by this **conversion** element

    Note: You can use negative numbers to identify strings extracted from the end of the string, rather than from the beginning. In other words, a token value of **–1** extracts the first delimited substring from the end.

    Use a **constant** element with a **value** attribute to specify the extraction order.

  Note: The values of the **name** attribute must be in all lower-case letters.

Table 9 shows an example of extracting the first name and last name from a name field. In this example, the name field contains the last name followed by the first name, separated by a number sign (**#**).

Table 9: Example of Substring Extraction by Delimiter

Input	Output	
Name	**FirstName**	**LastName**
'Doe#Jane'	'Jane'	'Doe'
'Katana#Jean-Paul'	'Jean-Paul'	'Katana'

Example 3 shows **conversion** elements that define this extraction.

Example 3: Substring Extraction by Delimiter

```
<conversion type="EXTRACTDELIM" table="Employee">
    <output>
        <column name="FirstName" type="varchar(100)"/>
    </output>
    <param name="input">
        <column name="Name"/>
    </param>
    <param name="delim">
        <constant value="#"/>
    </param>
    <param name="token">
        <constant value="2"/>
    </param>
</conversion>
<conversion type="EXTRACTDELIM" table="Employee">
    <output>
        <column name="LastName" type="varchar(100)"/>
    </output>
    <param name="input">
        <column name="Name"/>
    </param>
    <param name="delim">
        <constant value="#"/>
    </param>
    <param name="token">
        <constant value="1"/>
    </param>
</conversion>
```

Defining Data Concatenations

Concatenation combines two or more data fields into a new field. For example, you can use concatenation to recreate date fields from individual month, day, and year fields after extraction. You can also use this type of conversion to combine separate fields to create a new field you want to map. If your ATS schema has separate fields for the century and the year, you can combine them into a single field that you can map to **date.year_component**.

In the **conversion** element, you specify concatenation as **CONCAT**:

```
<conversion type="CONCAT" ...>
```

Inside the **conversion** element, you provide the following elements:

- One **output** element that specifies the column to hold the result of the conversion.

- One or more **param** elements, each of which has one of the following **name** attributes:
  - **name="input"**: a field with data to combine

    Use a **column** element with a **name** attribute to specify each input field, in the order they appear in the concatenation.
  - **name="delim"**: a delimiter string to include between concatenated values

    Use a **constant** element with a **value** attribute to specify the string.
  - **name="trim"**: specifies whether to trim white space (spaces, tabs) from the end of string fields before concatenating them

    Use a **constant** element with a **value** attribute of **Y** (for Yes) or **N** (for No) to specify whether to trim blanks.

Note: The values of the **name** attribute must be in all lower-case letters.

Table 10 shows an example of combining day, month, and year fields to create a date. In this example, the Data Converter placed slashes (/) between the concatenated fields.

Table 10: Example of Concatenation

Input			Output
HDay	**HMonth**	**HYear**	**HDate**
'12'	'09'	'1987'	'09/12/1987'

Example 4 shows **conversion** elements that define this concatenation.

Example 4: Concatenation

```
<conversion type="CONCAT" table="Employee">
    <output>
        <column name="HDate" type="char(10)"/>
    </output>
    <param name="input">
        <column name="HDay"/>
        <column name="HMonth"/>
        <column name="HYear"/>
    </param>
    <param name="delim">
        <constant value="/"/>
    </param>
    <param name="trim">
        <constant value="Y"/>
    </param>
</conversion>
```

Defining Arithmetic Operations

You can define conversions that perform calculations using the values of one or more fields in a table and store the result in a new column in the internal database. Your calculations can include combinations of the standard arithmetic operations: addition, subtraction, multiplication, and division. All calculations operate on string data whose data type is **varchar**.

In the **conversion** element, you specify calculations as **ARITHMETIC**:

```
<conversion type="ARITHMETIC" ...>
```

Inside the **conversion** element, you provide the following elements:

- One **output** element that specifies the column to hold the result of the conversion.
- One or more **param** elements, each of which has one of the following **name** attributes:
  - **name="expression"**: a control string that specifies the calculation you want to perform

    Use a **constant** element with a **value** attribute to specify the expression, using the following symbols:

?	a data value from an input field
+	addition
–	subtraction
*****	multiplication
/	division
( )	for grouping operations

  - **name="input"**: a field with data to operate on

    Use a **column** element with a **name** attribute to specify each input field, in the order they appear in the arithmetic expression.

  Note: The values of the **name** attribute must be in all lower-case letters.

Table 11 shows an example of calculating the sale price of an item with a 10% discount.

Table 11: Example of Arithmetic Calculation

Input		Output
Price	**Discount**	**SalePrice**
50	10	45

Example 5 shows **conversion** elements that define this calculation.

Example 5: Arithmetic Operations

```
<conversion type="ARITHMETIC" table="Product">
    <output>
        <column name="SalePrice" type="varchar(4)"/>
    </output>
    <param name="expression">
        <constant value="? * (1 - ?/100)"/>
    </param>
    <param name="input">
        <column name="Price"/>
        <column name="Discount"/>
    </param>
</conversion>
```

Testing Your Conversion Specification

After adding conversion definitions to your XMP file, you can test the results of the conversion specifications by running a data transformation, using either the Modulant Curtana Analyst Tool or an application you write using the Lifecycle Manager API. For information about the Lifecycle Manager API, see the *Modulant Curtana Developer's Guide*.

To test your conversion specification using the Analyst Tool, you must specify the locations of the XMP files that contain the conversion definitions. To do this, you modify the file **dataconverter.xml** in the **conf** directory of your Modulant Curtana platform installation. For information about the format of this file, see Appendix C, "dataconverter.xml Reference."

The Transformation Engine validates structure of the XMP file against the DTD before either data conversion phase. If the Data Converter encounters an error during either data conversion phase, the transformation run stops at that point. If this happens, check the log file **curtana.log** in the **logs** directory of your Modulant Curtana installation.

If the transformation run is successful, check the output for accuracy. If the results do not match what you expect, review the corresponding conversion specifications in the XMP file.

Log File Messages

As the Data Converter runs, it writes status messages to the log file. This section lists the informational, warning, and error messages you might see, and explains what they mean.

Informational Messages

The Data Converter logs the following messages to inform you of its progress through a transformation run:

- **Loading conversions from list {<conversionList> *name*}**

  The Data Converter has identified a **conversionList** element in your XMP file with the specified ***name***, and is reading the conversion specifications in that list.

- **Preparing Data Converter for {<conversionList> *name*}**

  The Data Converter is preparing to execute the conversion specifications in the **conversionList** element with the specified ***name***.

- **Identifying useless conversions**

  The Data Converter is looking for conversion definitions that are either redundant or could potentially conflict with other conversions.

- **Creating input column in ATS for {<conversionList> *name*}**

  The Data Converter is creating columns in the internal database to process one of the conversion specifications in the **conversionList** element with the specified ***name***.

- **Checking input column in ATS for {<conversionList> *name*}**

  The Data Converter is verifying columns it has created in the internal database to process one of the conversion specifications in the **conversionList** element with the specified ***name***.

- **Aggregating conversion rules**

  The Data Converter is combining the conversion specifications it has read, in preparation for processing them.

- **Executing conversions in {<conversionList> *name*}**

  The Data Converter is performing the conversions defined in the conversion specifications in the **conversionList** element with the specified ***name***.

Warning Messages

The Data Converter logs the following warnings to inform you of situations it encounters during a transformation run:

- **<dataConverter> tag not found. Data Converter will be ignored.**

  The XMP file does not contain a **dataConverter** element with conversion specifications; therefore, the Transformation Engine will not call the Data Converter for this transformation run.

- **Conversion list named {<conversionList> *name*} not found in XMP. Data Converter will be ignored.**

  The XMP file does not contain a **conversionList** element with the specified ***name*** (either **PREPROCESSOR** or **POSTPROCESSOR**); therefore, the Transformation Engine will not call the Data Converter to perform conversions for the specified phase.

- **Overwritten conversion: {*code for conversion*} is overwritten by {*code for conversion*}**

  Two conversion specifications conflict. The Data Converter performs only the last of the listed conversions. This message shows you the a description of the conflicting conversion specifications.

Error Messages

The Data Converter logs the following errors to inform you of problems it encounters during a transformation run:

- **Cannot drop column {*column_name*} in table {*table_name*}.**

  During database cleanup, Oracle has returned an error message after attempting to delete the specified column from the internal database after performing data conversions. The error log will also contain the SQL error message from Oracle.

- **Error when executing the conversion [SQL = {*SQL*}]**

  Oracle encountered an error executing the specified SQL code during a conversion. The error log will also contain the SQL error message from Oracle.

- **SQL error on colExist**

  Oracle could not find a column that the Data Converter expected to be there. The error log will also contain the SQL error message from Oracle.

- **ATS source tables not ready for processing**

  Either Oracle couldn't create one or more requested columns or the input for a conversion is invalid. If the Data Converter encounters this error, the error log will also contain one or more of the following errors:

  - **Missing column *colName* in table *tableName***
  - **Cannot create column *colName* in table *tableName* with type *dataType***

- **Errors occurred when parsing the XMP**

  The Data Converter encountered an error while reading the XMP file, usually an invalid parameter (either a missing parameter or too many parameters). In this case, the error log will contain one or more of the following errors:

  - **Cannot instantiate conversion from XMP; check the params: *XML code***
  - **Missing output column**
  - **Missing input(s)**
  - **The trim value must be "Y" (default) or "N"**
  - **The number of inputs must match the number of '?' in the expression**
  - **Missing expression**
  - **The token must have a numeric value, different from 0**
  - **Missing delimiter**
  - **Missing token**
  - **Token must have a numeric value**
  - **Missing start**
  - **Start must have a numeric value**
  - **Missing length**
  - **Length must have a numeric value**
  - **Missing parameter name='*value*' in: *XML code***

    This message indicates that something is missing or incorrect in the conversion specification; for example, you might have defined an arithmetic conversion without specifying an expression.