



# DBMaster

---

## XML Solution Technique Document

---

P-E5002-XML Solution Technique Document

Version: 01.00

**Document No: 50/DBM50-T02222008-01-XMLT**

**Author: DBMaster Production Team, Syscom Computer Engineering CO.**

**Publication Date: February 22, 2008**

## Table of Content

1. Introduction.....	1-1
1.1 Additional Resources .....	1-1
1.2 Document Conventions .....	1-2
2. Overview.....	2-1
2.1 XML Language.....	2-1
2.2 XTT/XTM.....	2-1
2.3 XML Type Index and Predicate.....	2-1
2.4 XML Validate UDF .....	2-1
2.5 JData Transfer Tool .....	2-2
3. XML Learning .....	3-1
3.1 XML basic.....	3-1
3.1.1 WHAT IS XML.....	3-1
3.1.2 HOW CAN XML BE USED.....	3-1
3.2 XML Tree .....	3-2
3.2.1 AN EXAMPLE XML DOCUMENT.....	3-2
3.2.2 XML DOCUMENTS FORM A TREE STRUCTURE .....	3-2
3.3 XML Syntax .....	3-3
3.3.1 XML SYNTAX RULES.....	3-3
3.3.2 XML ELEMENTS .....	3-5
3.3.3 XML ATTRIBUTES.....	3-6
4. XML Transfer Template Tool .....	4-1
4.1 Getting to Know the XTT Tool .....	4-2
4.1.1 OPENING THE XTT TOOL AND LOGGING INTO A DATABASE....	4-2
4.1.2 THE MAIN CONSOLE.....	4-2
4.1.3 THE MENU BAR .....	4-3
4.1.4 THE TOOLBAR .....	4-5
4.1.5 THE XTT EDITING PANEL .....	4-6
4.1.6 THE DATABASE SCHEMA PANEL.....	4-7
4.1.7 THE DETAILED EDITING PANEL.....	4-8
4.1.8 THE CUSTOMIZE DIALOG .....	4-11
4.1.9 THE USER PREFERENCES DIALOG.....	4-11
4.1.10 THE TREE OPERATION OPTIONS DIALOG .....	4-11

4.2	Creating a New XTT .....	4-11
4.2.1	CREATING AN EMPTY XTT FILE.....	4-11
4.2.2	CREATING AN XTT FROM A DTD FILE .....	4-12
4.2.3	CREATING AN XTT FROM AN XSD FILE .....	4-13
4.2.4	CREATING AN XTT FROM AN XML FILE .....	4-13
4.3	Editing an XTT .....	4-14
4.3.1	ABOUT THE DESIGN VIEW .....	4-14
4.3.2	INSERTING A TABLE .....	4-14
4.3.3	ADDING NEW ELEMENTS AND ATTRIBUTES.....	4-15
4.3.4	MAPPING DATA TO ELEMENTS AND ATTRIBUTES .....	4-16
4.3.5	SAVING AN XTT .....	4-16
4.4	Generating a DTD .....	4-16
4.5	Generating an XSD .....	4-17
4.6	Generating XML data .....	4-18
5.	XTT API Functions .....	5-1
5.1	XTT API in Java.....	5-1
5.1.1	PUBLIC METHODS:.....	5-1
5.1.2	EXAMPLE: .....	5-4
5.2	XTT Stored Procedure.....	5-5
5.2.1	STORED PROCEDURE DEFINITION:.....	5-5
5.2.2	PRIVILEGE .....	5-5
5.2.3	EXAMPLES.....	5-5
6.	XML Transfer Mapping Tool .....	6-1
6.1	Getting to know the XTM Tool.....	6-1
6.1.1	THE MAIN CONSOLE.....	6-1
6.1.2	THE MENU BAR .....	6-2
6.2	The Toolbar .....	6-3
6.3	XTM Object Tree .....	6-4
6.4	XML Schema Tree .....	6-4
6.5	Database Schema Tree .....	6-4
6.6	Creating an XTM.....	6-4
6.6.1	ADDING A NEW JDBC DRIVER .....	6-5
6.7	Mapping xpath statements to XTM object nodes... ..	6-6
6.8	Executing an XTM .....	6-8
6.8.1	SAVING AN XTM AS AN SQL SCRIPT .....	6-8
6.8.2	SAVING AN XTM AS AN XSL FILE AND EXECUTING.....	6-9
7.	XTM API Functions.....	7-1

7.1	XTM API in Java .....	7-1
7.1.1	PUBLIC METHODS:.....	7-1
7.1.2	EXAMPLE: .....	7-4
7.2	XTM Stored Procedure .....	7-4
7.2.1	STORED PROCEDURE DEFINITION:.....	7-4
7.2.2	PRIVILEGE .....	7-4
7.2.3	EXAMPLE .....	7-5
8.	XML Type Index and Predicate.....	8-1
8.1	Managing Index .....	8-1
8.2	Creating Indexes on XML column.....	8-1
9.	XML Validate UDF .....	9-1
9.1	Create DTD/XML Validate UDF .....	9-1
9.1.1	FLEXML.....	9-1
9.1.2	DBMASTER DTD VALIDATION UDF GENERATOR .....	9-3
9.1.3	DEFAULT VALIDATOR.....	9-4
9.2	Add XMLType column .....	9-4
9.3	Query XMLType column .....	9-4
9.3.1	EXTRACT.....	9-4
9.3.2	EXTRACTVALUE .....	9-5
9.3.3	EXISTSNODE .....	9-6
9.4	Update XMLType column .....	9-6
9.4.1	INSERT-BEFORE .....	9-7
9.4.2	INSERT-AFTER .....	9-8
9.4.3	INSERT-ATTRIBUTE .....	9-8
9.4.4	INSERT-TEXT-BEFORE .....	9-9
9.4.5	INSERT-TEXT-AFTER .....	9-9
9.4.6	APPEND-TEXT .....	9-9
9.4.7	APPEND.....	9-9
9.4.8	UPDATE.....	9-9
9.4.9	REMOVE .....	9-10
9.4.10	RENAME .....	9-10
10.	JData Transfer Tool .....	10-1
10.1	Importing data from XML .....	10-1
10.2	Exporting data to XML .....	10-4

# 1. Introduction

Welcome to XML solution technique document. XML is an extensible markup language and is gradually becoming the standard in exchanging and representing data. Not surprisingly, effective and efficient querying of XML data has become an increasingly important issue.

DBMaster includes two Java-based, platform-independent tools for passing data between a database and XML documents. The XML Transfer Template tool and the XML Transfer Mapping tool allow you to create custom templates that determine how data maps from a database to XML files.

Once you have created a template, you can use one of the APIs provided by DBMaster to help automate the process of synchronizing data between the database and XML files. DBMaster provides stored procedures as well as APIs in Java to help you accomplish this task.

The XTT is designed for changing data template through xml format. The XTM maps the relationship between XML and table. Actually is equal to through xml format importing/exporting data, but the difference is that users can use XTT/XTM establishing xml data format. The follow sections describe the tools and introduce how to create or edit XTT/XTM file etc. If you want to know more information about how to use XTT /XTM Tools, please consult '*XTT/XTM Tool user's Guide*'.

## 1.1 Additional Resources

DBMaster provides a complete set of DBMS manuals in addition to this one. For more detailed information on a particular subject, consult one of the books listed below:

- For an introduction to DBMaster's capabilities and functions, refer to the *DBMaster Tutorial*.
- For more information on designing, administering, and maintaining a DBMaster database, refers to the *Database Administrator's Guide*.
- For more information on database management, refer to the *JDBA Tool User's Guide*.
- For more information on database server management, refer to the *JServer Manager User's Guide*.
- For more information on configuring DBMaster, refer to the *JConfiguration Tool Reference*.
- For more information on the native ODBC API, refer to the *ODBC Programmer's Guide*.

- For more information on the dmSQL interface tool, refer to the *dmSQL User's Guide*.
- For more information on the SQL language used in dmSQL, refer to the *SQL Command and Function Reference*.
- For more information on error and warning messages, refer to the *Error and Message Reference*.
- For more information on the DBMaster COBOL Interface, refer to the *DCI User's Guide*.

## 1.2 Document Conventions

This book uses a standard set of typographical conventions for clarity and ease of use. The NOTE, Procedure, Example, and Command Line conventions also have a second setting used with indentation.

CONVENTION	DESCRIPTION
<i>Italics</i>	Italics indicate placeholders for information that must be supplied, such as user and table names. The word in italics should not be typed, but is replaced by the actual name. Italics also introduce new words, and are occasionally used for emphasis in text.
<b>Boldface</b>	Boldface indicates filenames, database names, table names, column names, user names, and other database schema objects. It is also used to emphasize menu commands in procedural steps.
KEYWORDS	All keywords used by the SQL language appear in uppercase when used in normal paragraph text.
SMALL CAPS	Small capital letters indicate keys on the keyboard. A plus sign (+) between two key names indicates to hold down the first key while pressing the second. A comma (,) between two key names indicates to release the first key before pressing the second key.
NOTE	Contains important information.

CONVENTION	DESCRIPTION
➔ <b>Procedure</b>	Indicates that procedural steps or sequential items will follow. Many tasks are described using this format to provide a logical sequence of steps for the user to follow
➔ <b>Example</b>	Examples are given to clarify descriptions, and commonly include text, as it will appear on the screen. Other forms of this convention include Prototype and Syntax.
<b>CommandLine</b>	Indicates text, as it should appear on a text-delimited screen. This format is commonly used to show input and output for dmSQL commands or the content in the dmconfig.ini file

*Table 1-1 Document Conventions*

## 2. Overview

This chapter is a summarizer of this manual basic knowledge. It will help your to get the xml relatively description listed faster.

### 2.1 XML Language

If you do not quite understand xml language, you can refer to chapter 3 to get xml concept, grammar, and other related knowledge, it will let you easily understand xml.

### 2.2 XTT/XTM

The XML transfer mapping (XTM) tool allows you to pass XML data to a database using XSL transformations. The XML Transfer mapping tool consists of three parts: an XML schema part, which displays the schema of the XML file(s) that you are using as source data; an SQL database part, which displays the database tables; and an XTM part, which displays the mapping from the XML schema to the database tables.

The purpose of the XML Transfer Template (XTT) tool is to provide a customizable bridge between database data and XML documents. The bridge takes the form of a template file, the XML Transfer Template (XTT). The XTT file determines which database tables and columns to map to which XML elements and attributes. You determine the mapping using drag-and-drop operations in the XTT tool. The XTT tool ensures that XTT syntax is correct, and also aids in performing tasks such as generating schema documents (XSD) or document type definitions (DTD).

### 2.3 XML Type Index and Predicate

To improve XML queries performance, we can create special XML index on XML columns. The XML index supports XML UDF: **extract()** and **extractvalue()**. The chapter 8 will introduction **extract()** and **extractvalue()** UDF and show how to create an index on an XML column using dmSQL.

### 2.4 XML Validate UDF

XML has become a key technology of today's Internet. Its self-description and extensibility have offered the flexibility for data exchange. DBMaster will support this new data type 'xmltype' and some XML related functions for user to store and manage XML data. The chapter 9 will provide some example of how to create, add, query and update xml validate udf.

## 2.5 JData Transfer Tool

The Data Transfer Tool provides a user-friendly interface for transferring data in and out of the database. It is another way to import/export xml file. The tool performs the following types of data transformation:

- Importing from text
- Importing from XML
- Importing from ODBC
- Exporting to text
- Exporting to XML

Each type of data transformation is performed through a wizard. Each wizard guides the user through every step in the data transformation process, and gives descriptive information on the purpose of each step and the effect of different choices on the result. The chapter 10, we only provide detail information description about importing from XML and exporting to XML wizard.

## 3. XML Learning

XML (Extensible Markup Language) has become a key technology of today's Internet. Its self-description and extensibility have offered the flexibility for data exchange.

### 3.1 XML basic

#### 3.1.1 WHAT IS XML

---

- XML stands for Extensible Markup Language
- XML is a **markup language** much like HTML
- XML was designed to **carry data**, not to display data
- XML tags are not predefined. You must **define your own tags**
- XML is designed to be **self-descriptive**
- XML is a **W3C** Recommendation

#### 3.1.2 HOW CAN XML BE USED

---

XML is used in many aspects of web development, often to simplify data storage and sharing.

##### **XML Separates Data from HTML**

If you need to display dynamic data in your HTML document, it will take a lot of work to edit the HTML each time the data changes. With XML, data can be stored in separate XML files. This way you can concentrate on using HTML for layout and display, and be sure that changes in the underlying data will not require any changes to the HTML.

##### **XML Simplifies Data Sharing**

In the real world, computer systems and databases contain data in incompatible formats.

XML data is stored in plain text format. This provides a software- and hardware-independent way of storing data. This makes it much easier to create data that different applications can share.

##### **XML Simplifies Data Transport**

With XML, data can easily be exchanged between incompatible systems.

One of the most time-consuming challenges for developers is to exchange data between incompatible systems over the Internet.

Exchanging data as XML greatly reduces this complexity, since the data can be read by different incompatible applications.

## XML Simplifies Platform Changes

Upgrading to new systems (hardware or software platforms), is always very time consuming. Large amounts of data must be converted and incompatible data is often lost.

XML data is stored in text format. This makes it easier to expand or upgrade to new operating systems, new applications, or new browsers, without losing data.

## XML Makes Your Data More Available

Since XML is independent of hardware, software and application, XML can make your data more available and useful.

Different applications can access your data, not only in HTML pages, but also from XML data sources.

## 3.2 XML Tree

XML documents form a tree structure that starts at "the root" and branches to "the leaves".

### 3.2.1 AN EXAMPLE XML DOCUMENT

---

XML documents use a self-describing and simple syntax:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

The first line is the XML declaration. It defines the XML version (1.0) and the encoding used (ISO-8859-1 = Latin-1/West European character set).

The next line describes the root element of the document (like saying: "this document is a note"):

```
<note>
```

The next 4 lines describe 4 child elements of the root (to, from, heading, and body):

```
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
```

And finally the last line defines the end of the root element:

```
</note>
```

### 3.2.2 XML DOCUMENTS FORM A TREE STRUCTURE

---

XML documents must contain a root element. This element is "the parent" of all other elements.

The elements in an XML document form a document tree. The tree starts at the root and branches to the lowest level of the tree.

All elements can have sub elements (child elements):

```
<root>
  <child>
    <subchild>....</subchild>
  </child>
</root>
```

The terms parent, child, and sibling are used to describe the relationships between elements. Parent elements have children. Children on the same level are called siblings (brothers or sisters).

All elements can have text content and attributes (just like in HTML).

## 3.3 XML Syntax

The syntax rules of XML are very simple and logical. The rules are easy to learn, and easy to use.

### 3.3.1 XML SYNTAX RULES

---

#### All XML Elements Must Have a Closing Tag

In HTML, you will often see elements that don't have a closing tag:

```
<p>This is a paragraph
<p>This is another paragraph
```

In XML, it is illegal to omit the closing tag. All elements must have a closing tag:

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
```

**NOTE** *You might have noticed from the previous example that the XML declaration did not have a closing tag. This is not an error. The declaration is not a part of the XML document itself, and it has no closing tag.*

#### XML Tags are Case Sensitive

XML elements are defined using XML tags.

XML tags are case sensitive. With XML, the tag <Letter> is different from the tag <letter>.

Opening and closing tags must be written with the same case:

```
<Message>This is incorrect</message>
<message>This is correct</message>
```

#### XML Elements Must be Properly Nested

In HTML, you will often see improperly nested elements:

```
<b><i>This text is bold and italic</b></i>
```

In XML, all elements must be properly nested within each other:

```
<b><i>This text is bold and italic</i></b>
```

In the example above, "Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.

## XML Documents Must Have a Root Element

XML documents must contain one element that is the parent of all other elements. This element is called the root element.

```
<root>
  <child>
    <subchild>....</subchild>
  </child>
</root>
```

## XML Attribute Values Must be Quoted

XML elements can have attributes in name/value pairs just like in HTML.

In XML the attribute value must always be quoted. Study the two XML documents below. The first one is incorrect, the second is correct:

```
<note date=12/11/2007>
<to>Tove</to>
<from>Jani</from>
</note>
```

```
<note date="12/11/2007">
<to>Tove</to>
<from>Jani</from>
</note>
```

The error in the first document is that the date attribute in the note element is not quoted.

## Entity References

Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

This will generate an XML error:

```
<message>if salary < 1000 then</message>
```

To avoid this error, replace the "<" character with an entity reference:

```
<message>if salary &lt; 1000 then</message>
```

There are 5 predefined entity references in XML:

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

**NOTE** Only the characters "<" and "&" are strictly illegal in XML. The greater than character is legal, but it is a good habit to replace it.

## Comments in XML

The syntax for writing comments in XML is similar to that of HTML.

```
<!-- This is a comment -->
```

## With XML, White Space is Preserved

HTML reduces multiple white space characters to a single white space:

HTML:	Hello      my name is Tove
Output:	Hello my name is Tove.

With XML, the white space in your document is not truncated.

## XML Stores New Line as LF

In Windows applications, a new line is normally stored as a pair of characters: carriage return (CR) and line feed (LF). The character pair bears some resemblance to the typewriter actions of setting a new line. In Unix applications, a new line is normally stored as a LF character. Macintosh applications use only a CR character to store a new line.

## 3.3.2 XML ELEMENTS

---

### What is an XML Element

An **XML element** is everything from (including) the element's start tag to (including) the element's end tag.

An element can contain other elements, simple text or a mixture of both. Elements can also have attributes.

```
<bookstore>
<book category="CHILDREN">
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
<book category="WEB">
  <title>Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>
```

In the example above, `<bookstore>` and `<book>` have element contents, because they contain other elements. `<author>` has text content because it contains text.

In the example above only `<book>` has an **attribute** (`category="CHILDREN"`).

## XML Naming Rules

XML elements must follow these naming rules:

- Names can contain letters, numbers, and other characters
- Names must not start with a number or punctuation character
- Names must not start with the letters xml (or XML, or Xml, etc)
- Names cannot contain spaces

Any name can be used, no word are reserved.

### 3.3.3 XML ATTRIBUTES

---

XML elements can have attributes in the start tag, just like HTML. Attributes provide additional information about elements.

#### XML Attributes

From HTML you will remember this: ``. The "src" attribute provides additional information about the `<img>` element.

In HTML (and in XML) attributes provide additional information about elements:

```
  
<a href="demo.asp">
```

Attributes often provide information that is not a part of the data. In the example below, the file type is irrelevant to the data, but important to the software that wants to manipulate the element:

```
<file type="gif">computer.gif</file>
```

#### XML Attributes Must be Quoted

Attribute values must always be enclosed in quotes, but either single or double quotes can be used. For a person's sex, the person tag can be written like this:

```
<person sex="female">
```

or like this:

```
<person sex='female'>
```

If the attribute value itself contains double quotes you can use single quotes, like in this example:

```
<gangster name='George "Shotgun" Ziegler'>
```

or you can use character entities:

```
<gangster name="George &quot;Shotgun&quot; Ziegler">
```

#### XML Elements vs. Attributes

Take a look at these examples:

```
<person sex="female">  
  <firstname>Anna</firstname>  
  <lastname>Smith</lastname>  
</person>
```

```
<person>
```

## XML Learning

```
<sex>female</sex>  
<firstname>Anna</firstname>  
<lastname>Smith</lastname>  
</person>
```

In the first example sex is an attribute. In the last, sex is an element. Both examples provide the same information.

## 4. XML Transfer Template Tool

XML Transfer Template Tool is XTT for short. The purpose of the XML Transfer Template (XTT) tool is to provide a customizable bridge between database data and XML documents. The bridge takes the form of a template file, the XML Transfer Template (XTT). The XTT file determines which database tables and columns to map to which XML elements and attributes. You determine the mapping using drag-and-drop operations in the XTT tool. The XTT tool ensures that XTT syntax is correct, and also aids in performing tasks such as generating schema documents (XSD) or document type definitions (DTD).

Using The XTT tool to transform data is a four-phase process — creating or importing the XTT structure, linking XTT objects to the database with SQL queries, generating DTD or XSD files if necessary, and finally generating the XML document.

Usually, linking XTT objects will only be necessary if you are importing an existing XML structure from an XML file, XSD, or DTD. Likewise, linking will not be necessary if you are creating an XTT based on the database. Hybrid situations may exist, however; for example, where you have an existing XML structure but need to add new elements for new data.

XTT files define a map from database tables and columns to the elements and attributes of an XML file. An XTT file is a document with syntax similar to a valid XML document. Elements define tables and columns, and attributes define SQL queries, attribute names and values, and element values for XML documents generated using the XTT.

So if you want to backup a database or a table, first you must know the tool and then you can create an xtt file or import an excised file.

### ➡ Example:

The following is a complete XTT file that maps data from the table CARD. It maps the columns FIRSTNAME, LASTNAME, and TITLE as attributes of the element CARD, and the column NUM as a child element:

```
<?xml version="1.0" encoding="UTF-8"?>
<xtt:template xmlns:xtt="urn:schema-dbmaster-com:xml-template">
  <root>
    <CARD xtt:query="CARD_SQL0" xtt:command="select NUM, FIRSTNAME, LASTNAME,
TITLE, BMP from SYSADM.CARD">
      <xtt:attribute name="FIRSTNAME" value="$CARD_SQL0.FIRSTNAME" />
      <xtt:attribute name="LASTNAME" value="$CARD_SQL0.LASTNAME" />
      <xtt:attribute name="TITLE" value="$CARD_SQL0.TITLE" />
      <NUM xtt:textvalue="$CARD_SQL0.NUM" />
    </CARD>
  </root>
</xtt:template>
```

If the XTT in the above example is run, the following XML file is generated:

```
<?xml version="1.0" encoding="US-ASCII" ?>
<root>
  <CARD FIRSTNAME="Eddie" LASTNAME="Brown" TITLE="Manager">
    <NUM>1</NUM>
  </CARD>
</root>
```

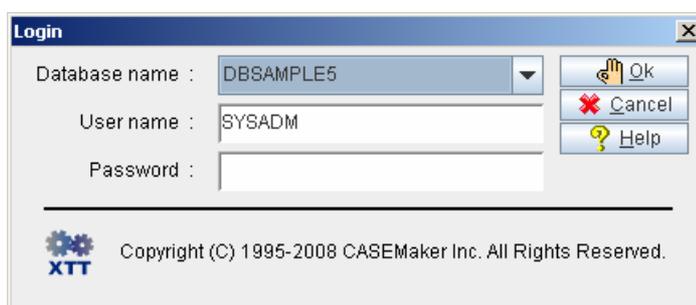
The XTT tool provides a simple user interface for scripting, validating and running XTT files. The following sections describe the user interface, and give procedures to help you quickly learn to start creating your own XTT files.

## 4.1 Getting to Know the XTT Tool

This section describes the elements of the XTT Tool user interface and how to log onto the database.

### 4.1.1 OPENING THE XTT TOOL AND LOGGING INTO A DATABASE

When you open the XTT tool from the Windows start menu you will automatically be prompted to log into the database. Select the database from which you want to export information. You must have an account on the database in order to log in. Be sure to use an account that has access to all the tables that you will need information from.



*Figure 4-1: The login dialog*

#### ➔ To open the XTT tool and log into a database:

1. **From the Windows Start menu, click Start > Programs > DBMaster 5.0> XML Transfer Template.**
2. In the **Login** dialog, select a database and enter a user name and password
3. Click **Ok**. The XTT tool will display database tables in the Database Schema Panel.

### 4.1.2 THE MAIN CONSOLE

The Main Console can be divided into five logical areas. Refer to figure below.

# XML Transfer Template Tool

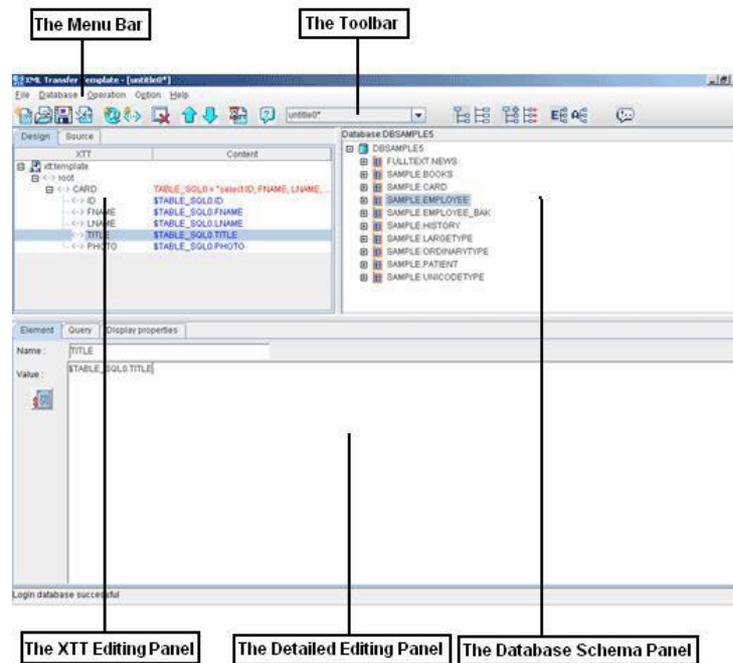


Figure 4-2: Elements of the Main Console

## 4.1.3 THE MENU BAR

The Menu Bar consists of five menus: **File**, **Database**, **Operation**, **Options**, and **Help**. Menu items are disabled if they cannot be used. Refer to the following sections for each menu item's function.

### **File**

The **File** menu consists of the following items:

- **New XTT > Empty XTT:** Creates a new empty XTT. Refer to [Creating an empty XTT file](#) for more information.
- **New XTT > With imported DTD:** Creates a new XTT based on a DTD file. Refer to [Creating an XTT from a DTD file](#) for more information.
- **New XTT > With imported XSD:** Creates a new XTT based on an XSD file. Refer to [Creating an XTT from an XSD file](#) for more information.
- **New XTT > With imported XML:** Creates a new XTT based on an XML file. Refer to [Creating an XTT from an XML file](#) for more information.
- **Open XTT:** opens the **Open** dialog with .XTT as the default file extension filter.
- **Close:** closes the XTT currently open in the XTT editing panel. If the XTT has been modified a confirmation dialog will ask to save changes.
- **Save:** saves the XTT currently open in the XTT editing panel. If the XTT has not been saved before, the **Save as** dialog will open.
- **Save as:** Opens the **Save as** dialog with .XTT as the default file extension.
- **Generate DTD:** Opens the **Save as** dialog with .DTD as the default file extension. Refer to [Generating a DTD](#) for more information.

- **Generate XSD:** Opens the **Save as** dialog with .XSD as the default file extension. Refer to [Generating an XSD](#) for more information
- **Recent files:** displays the most recently opened files
- **Exit:** exits the XML Transfer Template tool

## Database

---

The **Database** menu consists of the following items:

- **Connect:** Opens the **Login** dialog. A list of running databases appears in the drop down menu.
- **Disconnect:** stops the session with the database. The content in the **database schema panel** is cleared.
- **Refresh:** refreshes the **database schema panel** if a session is active.

## Operation

---

The **Operation** menu consists of the following items:

- **Insert > Element:** inserts a new empty element into the XTT object tree. Refer to [Adding New Elements and Attributes](#) for more information
- **Insert > Attribute:** inserts a new empty attribute into the XTT object tree. Refer to [Adding New Elements and Attributes](#) for more information
- **Undo:** returns the XTT object tree to the state it was in before the last modification
- **Copy:** copies the selected node of the XTT object tree and all descendants
- **Cut:** cuts the selected node of the XTT object tree and all descendants
- **Paste:** Pastes the last cut or copied node of the XTT object tree and all descendants.
- **Remove:** removes the selected node of the XTT object tree and all descendants
- **Run:** executes the XTT file.
- **Validate:** checks if variable references for elements exist in the parent element, and checks if SQL commands in elements are valid in the database

## Options

---

The **Options** menu consists of the following items:

- **Preferences:** opens the user preferences dialog
- **Tree operation options:** opens the tree operation options dialog

## Help

---

The **Help** menu consists of the following items:

- **Help:** opens the online help.
- **Web site:** opens a browser window to the web site [www.dbmaker.com](http://www.dbmaker.com).

- **About:** displays information about the XML transfer template including the build date and number, the CASEMaker technical support e-mail address, and a link to [www.dbmaker.com](http://www.dbmaker.com).

## 4.1.4 THE TOOLBAR

---

This section shows the toolbar items with their equivalent menu bar operations.

### File Operations

---

-  New empty XTT= menubar > File > New XTT > Empty XTT
-  Open XTT= menubar > File > Open XTT
-  Save= menubar > File > Save
-  Close= menubar > File > Close

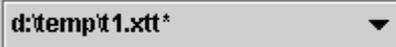
### XTT Tree Operations

---

-  Add Attribute Object = menubar > Operation > Insert > Attribute
-  Add Element Object = menubar > Operation > Insert > Element
-  Remove Tree Node = menubar > Operation > Remove
-  Move Up = Move current selected node before its previous sibling node. The element can be moved before another element but cannot be moved before an attribute.
-  Move Down = Move current selected node after its next sibling node. The attribute can only be moved after another attribute but can't be moved after an element.

### Opened Files

---

-  d:\temp\1.xml\*

The combo box displays all open XTT filenames. If the file has been edited, then there will be an asterisk (\*) after the filename. When you choose a different filename, the XTT edit panel will reload and show the XTT tree of the newly selected file.

### Operation Options

---

-  Run Transfer = menubar > Operations > Run
-  Help = menubar > Help > Help
-  Insert as Child = Insert elements as child elements (when performing drag-and-drop operations)
-  Inset as Sibling = Insert elements as sibling elements (when performing drag-and-drop operations)
-  = Add a new node (when performing drag-and-drop operations)
-  = Link by source structure (when performing drag-and-drop operations)
-  = Add a new element below the selected element (follows **Insert as ...** rule)

-  = Add a new attribute within the selected element
-  = Operation options – when selected, causes **Customize** dialog to appear when performing drag-and-drop operations on tables or views.

## 4.1.5 THE XTT EDITING PANEL

The XTT editing panel consists of two views: the design view and the source view. The design view contains the XTT object tree and is displayed by default when an XTT is created or opened. The XTT object tree is a logical representation of the XTT file itself, which is a well-formed XML document. The XTT object tree consists of five types of elements, described in the following table.

XTT object type	Tree node name	Content description
<xtt:template>	xtt:template	(none)
<root>	root	(none)
<xtt:attribute>	value of attribute 'name'	value of attribute 'value'
user-defined element without query or text value	element's tag name	(none)
user-defined element with query	element's tag name	value of attribute 'xtt:query' + '=' + value of attribute 'xtt:command'
user-defined element with text value	element's tag name	value of attribute 'textvalue'

Table 4-3: Element types in the XTT object tree.

The five XTT element types as they appear in the design view are illustrated in the following table.

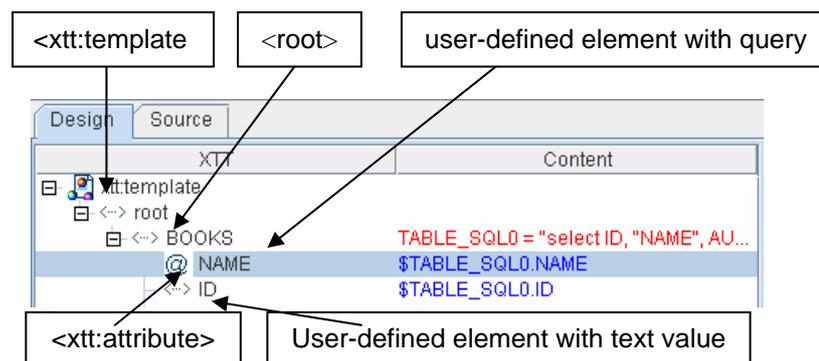


Table 4-4: Design view of element types in the XTT object tree

To select an XTT tree object, left-click on it and it will be highlighted in blue. If you right click on the design view, a pop-up menu will appear depending on the object that is highlighted. Right-clicking on an object will not cause it to be selected. The following table summarizes the pop-up menu contents for different selected elements.

Node type	Pop-up menu
<xtt:template>	N/A
attribute	<ul style="list-style-type: none"> <li>● 'Change to ELEMENT' - the selected attribute node will be replaced by an element with the same name. The data in the <b>value</b> attribute will be copied to the <b>textvalue</b> attribute of the new element</li> <li>● Cut – same as ctrl-X</li> <li>● Copy – same as ctrl-C</li> <li>● 'Remove' - removes the current selected node from the tree</li> </ul>
user-defined element	<ul style="list-style-type: none"> <li>● Element – creates a new element below the selected element. Whether the new element is inserted as a sibling or child element depends on whether the <b>Insert as Child</b> or <b>Insert as Sibling</b> option has been selected</li> <li>● Attribute – creates a new attribute for the selected element</li> <li>● 'Change to ATTRIBUTE' – only provided if the object has no sub node (either attribute or element). The selected element will be replaced by an attribute with the same name. The data in <b>text value</b> will be copied to the <b>value</b> attribute of the new attribute</li> <li>● Cut – same as ctrl-X</li> <li>● Copy – same as ctrl-C</li> <li>● Paste – same as ctrl-V</li> <li>● 'Remove' - to remove the current selected node from the tree</li> </ul>

*Table 4-5: pop-up menus available in the XTT editing panel design view*

4. The Source tab displays the source code for the XTT file.

## 4.1.6 THE DATABASE SCHEMA PANEL

The database schema panel shows the table/views of the connected database as a schema tree. By expanding the table/view node, it will show each column as its sub node. The nodes in the schema tree can be dragged into the XTT tree panel. It will then add a new node or link the schema information into XTT tree depending on the settings selected.

### 4.1.7 THE DETAILED EDITING PANEL

Detail editing panel shows the detailed properties of the selected node in the XTT tree. The general rule for text fields in the editing panel is that when an object outside the text field or area is selected the value is set. For example, the name field will be set after the next field is selected or the XTT tree selection is changed. The change can be seen on the XTT tree.

#### *xtt:template*

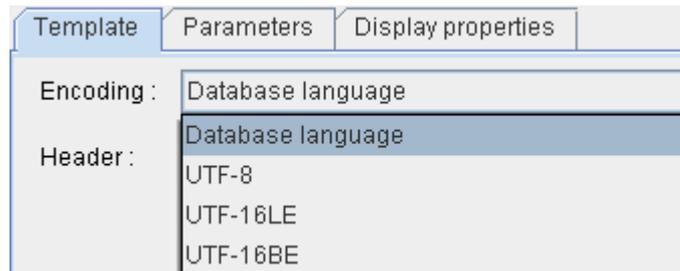


Figure 4-6: The language encoding menu of the detailed editing panel

**Encoding** – The **Encoding** menu specifies the text encoding for any output XML file. The choices are database local (the text encoding specified in the database), UTF-8, UTF-16LE and UTF-16BE. The default is database local. If database local is Big5 and the encoding setting is database local, then the output XML will be encoded in Big5.

**Header** – The header box it is where you add information like a schema file or applicable XSL in the output XML file. The XTT engine will print the content in this block to the output XML file after `<?xml version="1.0"...?>`. Be sure to type valid XML content in this header block.

Name		Default
teacher_id		10000

**Parameters** – you may add as many parameters as you want. The parameter name must be unique. Press the 'delete' key to remove a selected row. The last empty row cannot be removed. The parameter default can be empty.

Discription	Value
Empty abbreviation	system default (0 - always show start and end tag)
Indent	
Line break	system default (2 - add line break after start and end tag)
Line break character	system default (2 - {CR}{LF})
Display mode	system default (0 - display XML text content with escaped character)
NULL handling	system default (0 - skip null data)
LO export mode	system default (0 - dump large object into XML file)

There are a few display settings available for the output XML file. The default value for each setting is 'system default'. The display settings will apply to all XTT objects unless specified otherwise.

Empty abbreviation –

0 – always show start and end tags for an element, even if its content is empty.

# XML Transfer Template Tool

1 – to use the abbreviated form of the end tag if no sub element is produced. For example, **<Department id="1001" />**.

2 – hide the start and end tags if the element content is empty; no text, no attribute, no child element. If the element only has attributes, it will use type 1 abbreviation.

**Indent** – the number of indent spaces in the source document for each sub level as displayed in a text editor. For example, if the number is 2, then the start tag of the root will be indented 2 spaces, and a sub node of the root will be indented 4 spaces.

Line break –

0 – do not add any line break.

1 – add a line break after an end tag.

2 – add a line break after every start and end tag.

Line break char – the character(s) to be added as line break.

0 – {CR}

1 – {LF}

2 – {CR} {LF}.

**Display mode** – the way to display text data. Generally, the text character '<' will be replaced with '&lt;.' in the text content. But '<' can be used if it is enclosed in the CDATA section, or the text content itself is already an XML fragment and can be added into the output XML as a part of the XML content. There are 3 different ways to display text data.

0 – Display XML text content with the escape character.

1 – Use a CDATA section to enclose the text value.

2 – Make the content native XML text.

**Null handling** – specify how to handle null data.

0 – skip null data. If it's an attribute, then do nothing.

1 – Show empty content. For example, <NAME></NAME>.

2 – Display 'NULL'. For example, <NAME>NULL</NAME>.

**LO mode** – specify how to handle large objects.

0 – dump large object data directly into the XML file. Print a string if it is CLOB type data or print in hexadecimal format if it is BLOB type data.

1 – To store large object data in external file.

## ***User-defined element***

---

The following represents the properties of a user-defined element in the XTT object tree.

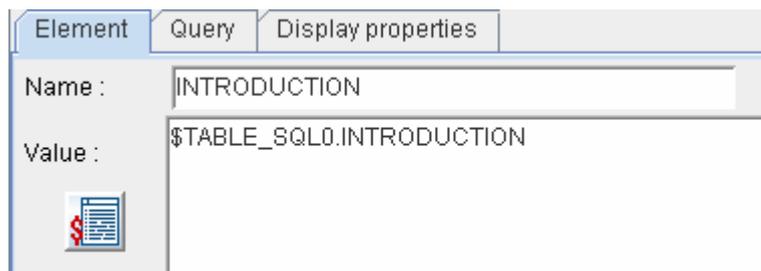


Figure 4-7: The detailed editing panel for a user-defined element

**Name** – the element name. It is case-sensitive and cannot be empty.

**Value** – the text value. This is a text expression, which can have both constant text and a variable reference. For example, you might want to add the country code to all phone number data, such as "886 - \$SQL1.PHONE", where "886 –" is the constant text and "\$SQL1.PHONE" is the variable reference. The XTT engine will concatenate both into the output XML file as the text value of the element.

**Browse button** – the browse button will list all the available parameters and variable references at the current level. You can choose the variable name(s) and insert them into the text expression field next to the browse button.

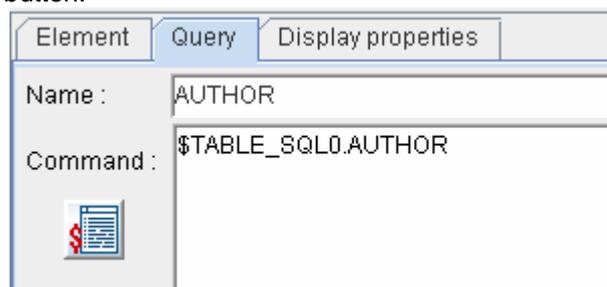


Figure 4-8: The query view for a user-defined element in the detailed editing panel

An element can have embedded query properties in it. A valid XTT element will have both fields (name and command) specified or both left as empty.

**Name** – the query name. It will be used as a variable reference in the sub nodes.

**Command** – SQL query statement. The statement must be able to generate a result set. For example, you cannot type a **delete table** statement here.

The display settings are similar to the ones in xtt:template, But the default is 'follow template setting'.

**Null handling** – there are only two choices instead of three. If 'follow template setting' is selected and the template has the null handling setting as '0 - skip null data', then it will be treated as '1 show empty content'.

1 – Show empty content. For example, <NAME></NAME>.

2 – Display 'NULL'. For example, <NAME>NULL</NAME>.

## **Attribute Node**

---

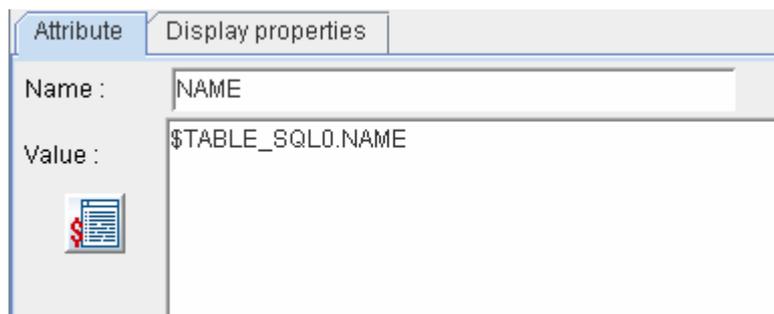


Figure 4-9: An attribute viewed in the detailed editing panel

Name – attribute name. The Name field cannot be empty. Multiple attributes must have names unique to the same parent element.

Value – the attribute value; a text expression field.

There are only two display settings for an attribute node; null handling and LO mode. The default for both is 'follow template setting'.

## 4.1.8 THE CUSTOMIZE DIALOG

---

The customize dialog appears when performing drag-and-drop operations if you have selected **show customize dialog**. The appearance of the customize dialog depends on the settings selected in the tree operation options dialog. There are two main views for the customize dialog depending on whether you have selected **Link by source structure** or **Add a new node**.

## 4.1.9 THE USER PREFERENCES DIALOG

---

The user preferences dialog is where you select the user interface language, and the method by which you wish to view the results. You may select English, Chinese, or Japanese as the user interface language. You may also choose to select your default XML browser or your default text editor to view the output when running the XTT.

## 4.1.10 THE TREE OPERATION OPTIONS DIALOG

---

The tree operation options dialog is where you select the behavior of drag-and-drop operations. You may select to add database objects as elements or attributes; add objects as child nodes or sibling nodes; and to add objects as new trees, or to link data to existing elements or attributes.

## 4.2 Creating a New XTT

An XTT is the map by which XML data files are produced. An XTT may be created in one of four ways: from an empty XTT file, from a DTD file, From an XSD file, and from an XML file.

### 4.2.1 CREATING AN EMPTY XTT FILE

---

Creating an empty XTT file is useful if you have data in a database that you want to display in XML form, but have no preconditions for how the XML data must be formatted. An empty XTT file consists only of the root node. After creating an empty XTT file, you may add elements and attributes. To learn about adding elements and attributes refer to [Editing an XTT](#).

➤ **To create an empty XTT file:**

1. Open the XTT tool and log in to the database that you want to use.
2. Click **File > New XTT > Empty XTT**. The root node will appear in the XTT object tree.

## 4.2.2 CREATING AN XTT FROM A DTD FILE

---

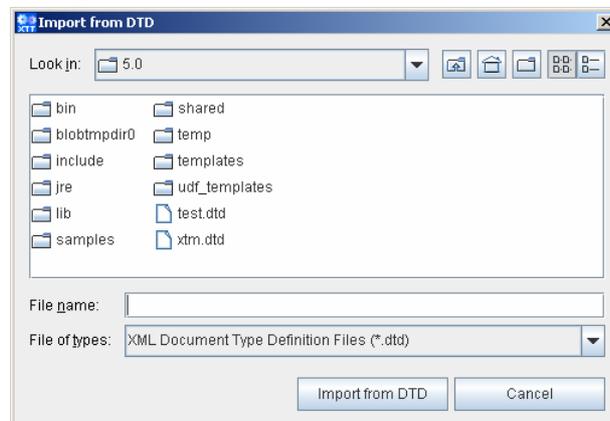
You may want to define the structure of your XML documents based on an existing Document Type Definition (DTD) file. If you have an existing XML document based on an external DTD and want to produce XML files from database data that conforms to that DTD, then you can create an XTT from a DTD file.

A root element must be specified for an XTT. During the import process a dialog allows you to select any valid element definition as the root for the XTT.

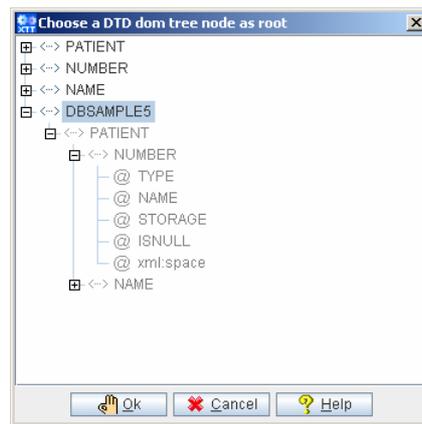
After creating the XTT, all the elements and attributes under the selected root node of the DTD will appear in the XTT object tree, however, none of the element or attribute definitions contain values – there is no method to pass SQL data to an XML file using the XTT. This may be accomplished by editing element nodes in the XTT object tree. For details on how to modify nodes in the XTT object tree, refer to [Mapping Data to Elements and Attributes](#).

➤ **To create an XTT from a DTD:**

1. Open the XTT tool and log in to the database that you want to use.
2. Click **File > New XTT > With imported DTD**.
3. In the **Import from DTD** dialog, select the DTD file to import and click **Import from DTD**.



4. In the **Choose a DTD dom tree node as root** dialog, select an element definition to be the root element in the XTT object tree and click **Ok**.

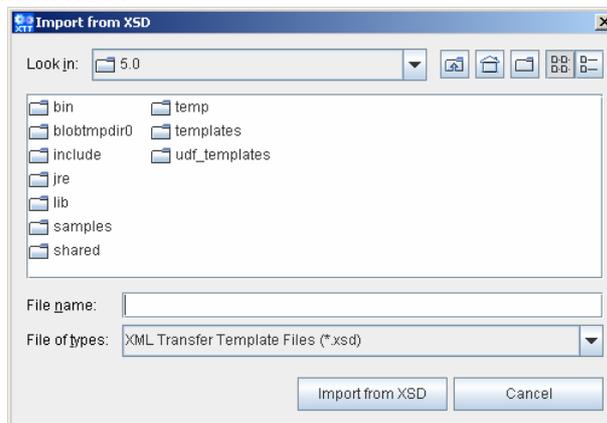


## 4.2.3 CREATING AN XTT FROM AN XSD FILE

It is also possible to base the XTT object tree structure on an XML schema defined by an XML Schema Definition (XSD) document file. As with DTD files, a root element must be specified, you may select any valid element definition as the root for the XTT. Also, none of the newly created attribute or element definitions in the XTT object tree has a value; you must modify the element and attribute definitions in order to get SQL data into an XML file. For details on how to modify nodes in the XTT object tree, refer to [Editing an XTT](#).

### ➤ To create an XTT from an XSD file:

1. Open the XTT tool and log in to the database that you want to use.
2. Click File > New XTT > With imported XSD.
3. In the **Import from XSD** dialog, select the DTD file to import and click **Import from XSD**.



4. In the **Choose an XSD Dom tree node as root** dialog, select an element definition to be the root element in the XTT object tree and click **Ok**.

## 4.2.4 CREATING AN XTT FROM AN XML FILE

If you do not have a DTD or XSD file to base your XTT structure on, but need to maintain consistency with an existing XML structure, then you may create an XTT directly from an XML file. The XTT tool will parse the structure of your XML document to generate an XTT object tree. The primary difference between this method of creating an XTT versus using a DTD or XSD is that you are not given an option as to which element will constitute the root node of the XTT object tree.

- ☛ **To create an XTT from an XML file:**
  1. Open the XTT tool and log in to the database that you want to use.
  2. Click **File > New XTT > With imported XML**.
  3. In the **Import from XML** dialog, select the DTD file to import and click **Import from XML**.

## 4.3 Editing an XTT

After you have created the root node and structure (if creating an XTT from XML, DTD, or XSD), you will want to provide content for the generated XML file. For blank XTT, this is primarily a drag-and-drop operation from the tables in the database schema panel into the XTT object tree. The sections [Inserting a Table](#) and [Adding New Elements and Attributes](#), describe the primary tasks you will need to accomplish if creating a new XTT.

For XTT based on an XML, DTD, or XSD file, you will need to add query statements and values to the attribute and element definitions in the XTT object tree. These tasks are described the section [Mapping Data to Elements and Attributes](#).

These tasks are not mutually exclusive and the above guidelines are only provided to enable you to quickly understand how to create a valid XTT document. At times you may find it useful to modify an element definition in a new XTT – one example being if you only wish to select values from an SQL table that meet some conditions. Or you may not need to conform precisely to the XML schema that your XTT is based on – in which case it is possible to use drag-and-drop operations to build your XTT object tree.

### 4.3.1 ABOUT THE DESIGN VIEW

---

The design view of the XTT editing panel displays the XTT object tree. For a new, blank XTT, the XTT object tree contains only the XTT template node and an empty root node. An XTT object tree that has been created from an XML, XSD, or DTD will have a different root. Refer to [The XTT Editing Panel](#) for detailed information on the objects and functions available in the design view.

### 4.3.2 INSERTING A TABLE

---

You can insert a table as a child or sibling. The first table you insert must be a child of the root node; attempting to add an element to the xtt:template node will return an error.

Before a table is added to the XTT object tree, you can choose which columns are to be added as elements, which columns to add as attributes, which columns to select but not add to the XTT object tree, and which columns not to select. This is accomplished in the customize dialog. Click the **show customize dialog** button if you want the customize dialog to appear when adding object to the XTT object tree.

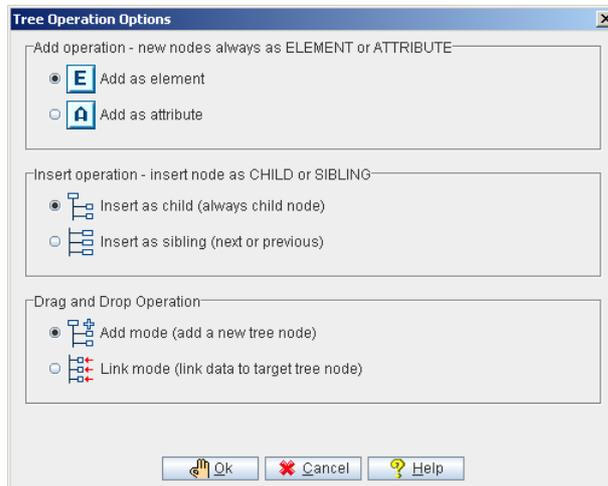
When inserting a table, the customize dialog will display a query object name, and the structure of the database object as it will appear after it is entered into the XTT object tree. By default both parent and child objects are inserted as elements. You may also choose to insert child objects as attributes, to select database objects without inserting them as attribute objects or element objects, or to not select the database objects.

After you have inserted the first table, subsequent tables may be added as children or siblings of the first table. Tables must always be represented as elements in the XTT object tree. If you have

created in the XTT an existing file, any tables you insert can be children or siblings of any of the existing XTT elements.

➔ **To insert a table as a child node:**

1. Click Option > Tree Operation Options.
2. In the Tree Operation Options dialog click Add as element, Insert as a child, and Add mode.



3. Select a table from the database schema panel.
4. Drag the table from the database schema panel to the XTT object tree node that is to be the parent of the new node.
5. In the customize dialog, select any columns that you do not want to include and click remove. Select any columns that you wish to select in the SQL command, but do not want to export to the XML file and click **hidden**. Select any columns that you wish to add as attributes and click **attribute**. Select any columns that you wish to add as elements and click **element**.
6. Click **Ok**. The table will appear as a new element in the XTT object tree. Columns will appear as elements and attributes, depending on how they were selected in the customize dialog.

### 4.3.3 ADDING NEW ELEMENTS AND ATTRIBUTES

---

You may need to add elements or attributes to the XTT object tree in order to create the desired data structure. Note that the following procedure only describes how to add empty elements and attributes. Refer to defining elements and defining attributes for information on how to map data from the database to empty elements in the XTT object tree.

➔ **To add new elements or attributes:**

1. Select the element to which you will add a new element or attribute.
2. Determine the relationship between the selected element and your new object: click **insert as sibling** to make the new object a sibling of the selected element, click **insert as child** to make the new object a child of the selected element.
3. Click **add attribute object** to add an attribute. Click **add element object** to add a new element.
4. Type a name for the new object in the name box of the **detailed editing panel** and press **Enter**.

### 4.3.4 MAPPING DATA TO ELEMENTS AND ATTRIBUTES

---

Elements should be associated with data. To associate an element object with data, the parent element object must contain an SQL query. The SQL query must select the table and column that will map to the child element object. After associating the parent element with an SQL query, you must associate a child element with the column.

Use the link by source structure tree operation option to associate elements with SQL queries. When you associate an element with an SQL query, use the customize dialog to associate child elements or attributes with the selected columns of the SQL query. Dragging a table from the database schema panel into a parent element in the XTT object tree will cause the customize dialog to open.

When linking element objects by source structure, the customize dialog will show the query object name, and two columns: the XTT object column, and the mapping column. The XTT object column displays all child elements and attributes of the element that you have dragged the table into. The mapping column displays any existing content, and is where you select the column content that you want to map from. Click a row in the mapping column to select an SQL data source for the corresponding XTT object.

➔ **To map data to an empty XTT element object:**

1. Click a table in the database schema panel.
2. Drag the table to an element object in the XTT object tree. The element you dragged the table into will be the parent element.
3. In the **Customize dialog**, select the mapping that you wish to perform. For each element and attribute object:
4. Click the Mapping column that corresponds to the XTT object that you wish to map database data to.
5. Select the value for the XTT object from the drop-down menu. The value is the equivalent of the column data in the database. It appears in the format of the name of the SQL query followed by a dot, and then the column name.
6. Click OK when you have completed selecting mapping values for all the XTT object nodes that you wish to map to database columns.

### 4.3.5 SAVING AN XTT

---

After you have completed editing the XTT object tree, you should save the XTT file. The saved XTT file can be later recalled for modification, or called in a stored procedure to automatically pass database data to XML files.

To save an XTT click the **Save** icon, or click **File > Save** from the menu bar.

## 4.4 Generating a DTD

If your development requirements determine that you will need to create a Document Type Definition (DTD) file to describe the structure of generated XML files, then you can use the **Generate DTD** function in the XML Transfer Template tool.

The DTD file structure will be consistent with the structure of the source XTT.

➔ **Example:**

**Given the following XTT file:**

# XML Transfer Template Tool

```
<?xml version="1.0" encoding="US-ASCII"?>
<xtt:template xmlns:xtt="urn:schema-dbmaster-com:xml-template">
  <root>
    <CARD xtt:query="CARD_SQL0" xtt:command="select NUM, FIRSTNAME, LASTNAME,
TITLE, BMP from SYSADM.CARD">
      <NUM xtt:textvalue="$CARD_SQL0.NUM" />
      <FIRSTNAME xtt:textvalue="$CARD_SQL0.FIRSTNAME" />
      <LASTNAME xtt:textvalue="$CARD_SQL0.LASTNAME" />
      <TITLE xtt:textvalue="$CARD_SQL0.TITLE" />
      <BMP xtt:textvalue="$CARD_SQL0.BMP" />
    </CARD>
  </root>
</xtt:template>
```

The resultant DTD file will be:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT root (CARD*) >
<!ELEMENT CARD (NUM, FIRSTNAME, LASTNAME, TITLE, BMP) >
<!ELEMENT NUM (#PCDATA) >
<!ELEMENT FIRSTNAME (#PCDATA) >
<!ELEMENT LASTNAME (#PCDATA) >
<!ELEMENT TITLE (#PCDATA) >
<!ELEMENT BMP (#PCDATA) >
```

## ➔ To generate a DTD file from an XTT object tree:

1. Ensure that the XTT that you want to convert to a DTD is open.
2. Click **File > Generate DTD**.
3. In the **Generate DTD** dialog, select an output path and specify a file name and encoding type. Possible encoding types include: **UTF-8**, **UTF-6LE**, **UTF-16BE**, and Local Code.
4. Click **Generate DTD**, a DTD file will be created in the selected folder.

## 4.5 Generating an XSD

You may wish to generate an XML schema file from the logical structure represented in the XTT file. Some tools may require a schema file to be able to parse XML data. The schema file structure is consistent with the XTT object tree structure. In the example below, a small XTT file is used to generate a schema file.

### ➔ Example

Given an XTT file with the following object tree structure:

```
<?xml version="1.0" encoding="US-ASCII"?>
<xtt:template xmlns:xtt="urn:schema-dbmaster-com:xml-template">
  <root>
    <CARD xtt:query="CARD_SQL0" xtt:command="select NUM, FIRSTNAME, LASTNAME,
TITLE, BMP from SYSADM.CARD">
      <xtt:attribute name="NUM" value="$CARD_SQL0.NUM" />
      <xtt:attribute name="FIRSTNAME" value="$CARD_SQL0.FIRSTNAME" />
      <xtt:attribute name="LASTNAME" value="$CARD_SQL0.LASTNAME" />
      <xtt:attribute name="TITLE" value="$CARD_SQL0.TITLE" />
      <BMP xtt:textvalue="$CARD_SQL0.BMP" />
    </CARD>
  </root>
</xtt:template>
```

```

        </CARD>
    </root>
</xtt:template>

```

The resultant schema file structure will be:

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="root">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="CARD" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="CARD">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="BMP" />
      </xsd:sequence>
      <xsd:attribute name="NUM" type="xsd:int"/>
      <xsd:attribute name="FIRSTNAME" type="xsd:string"/>
      <xsd:attribute name="LASTNAME" type="xsd:string"/>
      <xsd:attribute name="TITLE" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="BMP" type="xsd:string"/>
</xsd:schema>

```

## ☛ To generate an XSD file from an XTT object tree:

1. Ensure that the XTT that you want to convert to an XSD is open.
2. Click **File > Generate XSD**.
3. In the Generate **XSD** dialog, select an output path and specify a file name and encoding type. Possible encoding types include: **UTF-8**, **UTF-6LE**, **UTF-16BE**, and Local Code.
4. Click Generate **XSD**, an XSD file will be created in the selected folder.

## 4.6 Generating XML data

After creating the required transfer template and optional DTD or XSD file, you are ready to generate an XML file using the data stored in the database.

The **Run transfer** function is most useful for testing the XTT. After you have created an XTT, you can use it to generate XML documents on demand and pass data to your XML application.

The following example shows a completed XTT template file and corresponding output file.

```

<?xml version="1.0" encoding="US-ASCII"?>
<xtt:template xmlns:xtt="urn:schema-dbmaster-com:xml-template">
  <root>
    <CARD xtt:query="CARD_SQL0" xtt:command="select NUM, FIRSTNAME, LASTNAME,
    TITLE, BMP from SYSADM.CARD">
      <NUM xtt:textvalue="$CARD_SQL0.NUM" />

```

# XML Transfer Template Tool

```
<FIRSTNAME xtt:textvalue="$CARD_SQL0.FIRSTNAME" />
<LASTNAME xtt:textvalue="$CARD_SQL0.LASTNAME" />
<TITLE xtt:textvalue="$CARD_SQL0.TITLE" />
<BMP xtt:textvalue="$CARD_SQL0.BMP" />
</CARD>
</root>
</xtt:template>
```

The corresponding output file for the preceding XTT file:

```
<?xml version="1.0" encoding="US-ASCII" ?>
- <root>
- <CARD>
  <NUM>1</NUM>
  <FIRSTNAME>Eddie</FIRSTNAME>
  <LASTNAME>Chang</LASTNAME>
  <TITLE>Manager</TITLE>
  <BMP>lobdir5\blobfile0.tmp</BMP>
</CARD>
- <CARD>
  <NUM>2</NUM>
  <FIRSTNAME>Hook</FIRSTNAME>
  <LASTNAME>Hu</LASTNAME>
  <TITLE>Software Engineer</TITLE>
  <BMP>lobdir5\blobfile1.tmp</BMP>
</CARD>
</CARD>
- <CARD>
  <NUM>7</NUM>
  <FIRSTNAME>Oscar</FIRSTNAME>
  <LASTNAME>Tseng</LASTNAME>
  <TITLE>Software Engineer</TITLE>
  <BMP>lobdir5\blobfile6.tmp</BMP>
</CARD>
- <CARD>
  <NUM>8</NUM>
  <FIRSTNAME>Jerry</FIRSTNAME>
  <LASTNAME>Liu</LASTNAME>
  <TITLE>Manager</TITLE>
  <BMP>lobdir5\blobfile7.tmp</BMP>
</CARD>
</root>
```

## 5. XTT API Functions

The XTT API is provided by DBMaster for automating the output of XML data from the database to files or a location in memory. APIs is provided in Java, and also in the form of a stored procedure. A few common properties are required for XTT to process:

- Database connection - either with a connection string or pre-connected handle.
- XTT - either the XTT filename or XTT content in memory buffer.
- Output XML - either the XML filename or a pre-allocated buffer for output XML content.
- Parameters - a string with the parameter values for the template file. The string might contain several pairs of parameter names and values. Each pair is separated by comma. The text on the left of the equal sign is the name of parameter, on the right is the value. The values may be enclosed by doubles quote or single quotes. For example, "*student\_id='10012',class\_id='103'*".

### 5.1 XTT API in Java

#### 5.1.1 PUBLIC METHODS:

---

##### Constructor and destructor:

```
XTT( )
```

Constructs a XTT object.

```
finalize ( )
```

Before destroying a XTT object.

##### setDatabaseConnection

This function sets the database connection.

##### Syntax

```
int setDatabaseConnection( dbmaster.sql.JdbcOdbcConnection handle )
```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
conn	Input	DBMaster jdbc connection object (cannot be null).

RETURNS	
SUCCESS	Setting the database connection with hdbc will always return SUCCESS. If setting the database connection with a connection string, it will try to connect using the connection specified, and will return SUCCESS if successful.
SQLConnect Error	The function will return this string if setting the database connection with a connection string fails.

## setXtt

Sets the XTT filename or content.

### Syntax

```
void setXtt( String xttFileName );
```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
xttFileName	Input	XTT filename

```
void setXtt( byte [] xttValue );
```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
xttValue	Input	Byte array with XTT content.

## setOutputXml

Sets the output XML filename or a memory buffer for output content.

### Syntax

```
void setOutputXml( String xmlFileName );
```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
xmlFileName	Input	Output XML filename

```
void setOutputXml( byte [] xmlValue );
```

The output buffer is null terminated.

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
xmlValue	Input	Byte array with output XML content.

## setParameteres

Sets parameter values for the template file.

## Syntax

```
void setParameters ( String parameters );
```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
parameters	Input	The parameters for the template file, in the format of: [param_name="value" [,param_name="value"...]

## getError

If the return code from run() is not SUCCESS, then use this method to get the error message. If no error is returned, then this method will return NULL.

## Syntax

```
String getError();
```

## run

Starts the XML transformation.

## Syntax

```
int run ( );
```

RETURNS	DESCRIPTION
SUCCESS	Transformation successful.
ERR_XTT_INV_ARG	Missing required arguments/properties for XTT engine to process. For example, no XTT filename or XTT content is set.
ERR_XTT_XERCES_PARSER	Error returned by apache xerces parser. XTT is not in valid XML format.
ERR_XTT_INV_SYNTAX	Invalid XTT syntax. Possible causes: No recognized xtt element or attribute. Not exactly one user-defined element in xtt:template element. <xtt:attribute> is declared after any user-defined element in its parent. <xtt:parameter> is declared after a user-defined element under <xtt:template> Missing required attribute.
ERR_XTT_PROCESS	XTT process errors: Failed to create folder or file.

	Invalid variable references.
SQL Error	An SQL error occurred during processing.

### 5.1.2 EXAMPLE:

```

String xtt = "c:\\temp\\casel.xtt";
String xml = "c:\\temp\\casel.xml";
String error = "c:\\temp\\casel.log";
String param = null;
String dbname = "DBSAMPLE";
String user = "SYSADM";
String password = "";
try
{
    /* connect to dbmaster database via dbmaster jdbc bridge */
    Class.forName("dbmaster.sql.JdbcOdbcDriver");
    System.setProperty("DM_DRIVER_MODE", "CLIENT_SERVER");
    System.setProperty("DM_CONNECT_MODE", "CONNECT_DB");
    Connection conn = DriverManager.getConnection("jdbc:dbmaster:"+dbname,
user, password);
    /* new XTT object */
    XTT transfer = new XTT();

    /* set database connection with previous connected Jdbc connection. */
    transfer.setDatabaseConnection((dbmaster.sql.JdbcOdbcConnection)conn);

    /* set XTT filename */
    transfer.setXtt(xtt);

    /* set output XML filename */
    transfer.setOutputXml(xml);

    /* set parameter values for XTT template file */
    transfer.setParameters(param);

    /* start XML transformation */
    int rc = transfer.run();

    if( rc != 0 )
    {
        /* print out error message */
        System.out.println(transfer.getError());
    }
    conn.close();
}

```

```

catch( ClassNotFoundException e )
{
    e.printStackTrace();
}
catch( SQLException sqle )
{
    sqle.printStackTrace();
}

```

## 5.2 XTT Stored Procedure

The XTT stored procedure provides XML transformation functionality similar to the APIs in Java. The major differences are:

- It only accepts XTT and output XML filenames.
- The filenames specified are on the server site.

### 5.2.1 STORED PROCEDURE DEFINITION:

```

CREATE PROCEDURE XTT(
    VARCHAR(257)  XTTFILE           INPUT,
    VARCHAR(257)  OUTPUTFILE       INPUT,
    VARCHAR(257)  PARAMETERS       INPUT );

```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
XTTFILE	Input	XTT filename
OUTPUTFILE	Input	Output XML filename
PARAMETERS	Input	Parameter values for XTT template.

### 5.2.2 PRIVILEGE

The owner of this stored procedure is SYSTEM. The privileges of the stored procedure are the same as the user executing the stored procedure. For example, if there is a query in an XTM file to a table on which the currently logged on user does not have the privilege to select, then an SQL error will be returned.

### 5.2.3 EXAMPLES

```

dmSQL> call XTT('d:\document\test\xtt\case1.xml', 'c:\temp\case1.xml', NULL);

dmSQL> call XTT('d:\document\test\xtt\case1.xml', NULL, NULL);
ERROR (5612): [DBMaster] xtt invalid argument :
    xml filename or xml destination buffer is required.

```

## 6. XML Transfer Mapping Tool

The XML transfer mapping (XTM) tool allows you to pass XML data to a database using XSL transformations. The XML Transfer mapping tool consists of three parts: an XML schema part, which displays the schema of the XML file(s) that you are using as source data; an SQL database part, which displays the database tables; and an XTM part, which displays the mapping from the XML schema to the database tables.

Using the tool can be summarized in five basic steps: opening a source XML or XSD file to create a source XML schema, connecting to a database, creating an XTM structure from the database table, mapping elements and attributes from the source XML schema, and finally storing the XTM structure as an XSL file.

Once the XSL file is created, it can be used to transform any XML file that conforms to the source schema to database data.

### 6.1 Getting to know the XTM Tool

Unlike the XTT tool, the XTM tool does not require a connection to the database when it is opened. The tool creates a database connection when you create or open an XTM, and is used to display the database schema tree, described in [XML Schema Tree](#).

This chapter describes all of the major screen elements in the XTM tool.

#### 6.1.1 THE MAIN CONSOLE

---

The main console of the XTM tool can be broken down into five major areas as illustrated in the following figure.

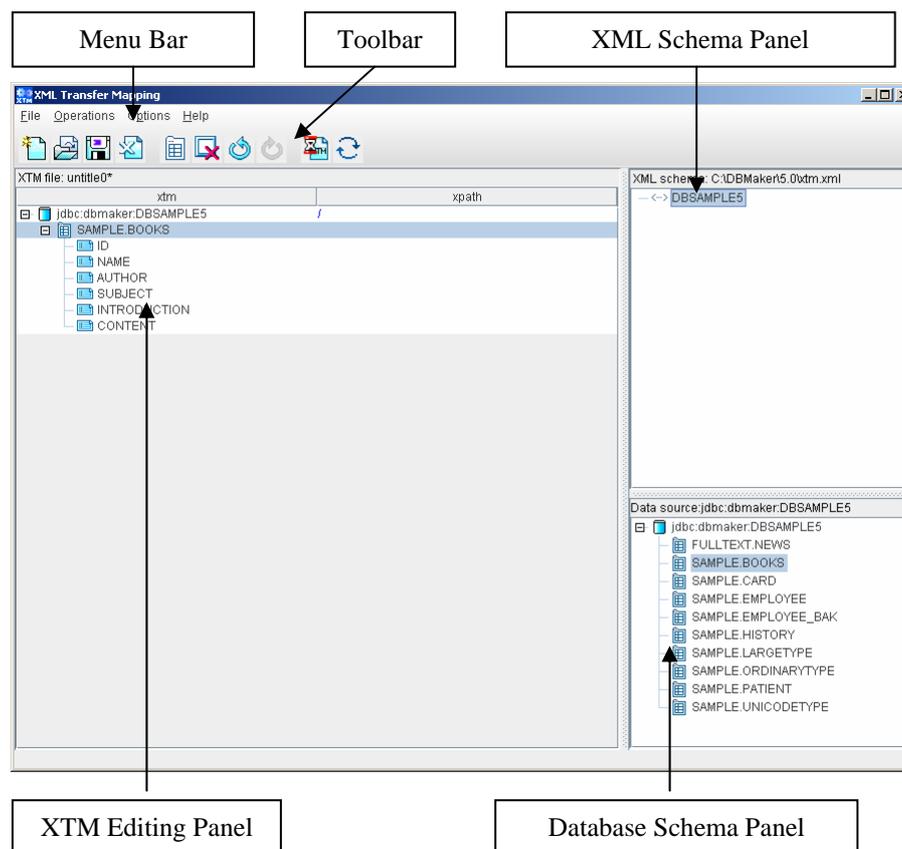


Figure 6-1: Elements of the XTM main console

## 6.1.2 THE MENU BAR

The menu bar consists of four menus: **File**, **Operations**, **Options**, and **Help**. Menu items are disabled if they cannot be used. Refer to the following sections for each menu item's function.

The **File** menu consists of the following items:

- **New**: opens the **New XTM** dialog, which prompts you to enter source schema and database information.
- **Open**: opens the **Open** dialog with .XSL as the default file extension filter.
- **Save**: saves the XTM currently open in the XTM editing panel as an XSL file. If the XTM has not been saved before, the **Save as** dialog will open.
- **Save as**: Opens the **Save as** dialog with .XSL as the default file extension.
- **Close**: closes the XTM currently open in the XTM editing panel. If the XTM has been modified, a confirmation dialog will ask to save changes.
- **Recent files**: Lists the XSL files opened during the current session.
- **Exit**: Exits the XTM tool. Opens the **Save as** dialog with .XSL as the default file extension if the current XTM has not been saved.

## Operation

The **Operation** menu consists of the following items:

- **Undo**: returns the XTT object tree to the state it was in before the last modification
- **Redo**: Executes the last performed action again
- **Insert**: Inserts a new XTM control node. Opens the **New XTM Control Node** dialog
- **Remove**: Removes the selected node of the XTM object tree and all descendants
- **Run**: Opens the **Execute XTM** dialog. The XTM dialog offers the option to immediately execute the XTM and send data to the database or to generate an SQL script for later use
- **Refresh**: Queries the database to refresh database schema

## Options

The **Options** menu consists of the following items:

- **Preferences**: opens the **Preferences** dialog, which allows you to choose the language that the UI is displayed in, and to specify different syntax for the connect section of the XSL file
- **JDBC Drivers**: opens the **JDBC Drivers** dialog, which allows you to connect to other data sources

## Help

The **Help** menu consists of the following items:

- **Help**: opens the online help.
- **web site**: opens a browser window to the web site [www.dbmaker.com](http://www.dbmaker.com).
- **about**: displays information about the XML transfer template including the build date and number, the CASEMaker technical support e-mail address, and a link to [www.dbmaker.com](http://www.dbmaker.com).

## 6.2 The Toolbar

-  New File: Menu bar > File > New
-  Open File: Menu bar > File > Open
-  Save File: Menu bar > File > Save
-  Close File: Menu bar > File > Close
-  Add new Table/Statement Node: Menu bar > Operations > Insert
-  Delete Node: Menu bar > Operations > Remove
-  Undo: Menu bar > Operations > Undo {command}
-  Redo: Menu bar > Operations > Redo

-  Run: Menu bar > Operations > run
-  Refresh Database: Menu bar > Operations > Refresh

## 6.3 XTM Object Tree

The XTM Object Tree is a logical representation of the structure of the XTM file. It contains two columns: the XTM column and the xpath column. The XTM column contains a graphical tree representation of objects that have been inserted from the Database Schema Tree. The xpath column contains a path to address the corresponding location in the XML schema tree.

## 6.4 XML Schema Tree

The XML Schema Tree provides a graphical representation of the schema of the XML, DTD, or XSD file from which the XTD will create addresses. Any XML file that conforms to the schema represented in the XML Schema Tree can be used as a data source after the XTM has been created and saves as an XSL.

## 6.5 Database Schema Tree

The Database Schema Tree displays a logical representation of database tables within the selected database.

## 6.6 Creating an XTM

Creating an XTM requires selecting a data source and connecting to a database.

The data source must be an XML, XSD, or DTD file. You must choose one of the elements in the source file to be the root. Only child elements and attributes of the selected element will be visible in the XML schema panel after you create the XTM file. If you want to include other elements or attributes, you will need to create a new XTM file.

Connecting to a database requires a driver and a connection to the database. The standard driver for DBMaster databases is **dbmaster.sql.JdbcOdbcDriver**. For more information about drivers refer to [Adding a New JDBC Driver](#). The connection to the database requires that the database is started, that a channel of communications over TCP/IP is open, and that a username and password for an account with privileges on the tables you wish to display is available.



### To create a new XTM:

1. Select **File > New** from the menu bar. The **New XTM file** dialog opens.

# XML Transfer Mapping Tool



2. In the **File path** box, type a file path or select the browse button to locate a source schema file (XML, XSD or DTD).
3. To select a root element different from the one indicated in the Root element box, click the browse button to the right. The **Choose root element** dialog appears.
4. Select the element you want to be the root from the tree. Double-click an element in the tree to expand the node and see its child elements.
5. Click **Ok**.
6. In the **Database** box, select a JDBC driver and a data source from the menus.
7. Enter the User name and password of an account on the database.
8. Click **Ok**. The selected XML schema and database tables will appear in the XML Schema Panel and Database Schema panel, respectively.

## ➔ To open an existing XTM:

1. Select **File > Open** from the menu bar.
2. Select the XSL file that you want to open using the file chooser.
3. In the **File path** box, type a file path or select the browse button to locate a source schema file (XML, XSD or DTD).
4. To select a root element different from the one indicated in the Root element box, click the browse button to the right. The **Choose root element** dialog appears.
5. Select the element you want to be the root from the tree. Double-click an element in the tree to expand the node and see its child elements.
6. Click **Ok**.
7. In the **Database** box, select a JDBC driver and a data source from the menus.
8. Enter the User name and password of an account on the database.
9. Click **Ok**. The selected XML schema and database tables will appear in the XML Schema Panel and Database Schema panel, respectively.

## 6.6.1 ADDING A NEW JDBC DRIVER

---

It may be necessary to use a different JDBC driver than the ones provided. It is possible to connect to databases provided by different vendors as long as the JDBC driver is present. The XTM tool provides this function through the Options menu. It is not necessary to have an XTM open to add a new driver.

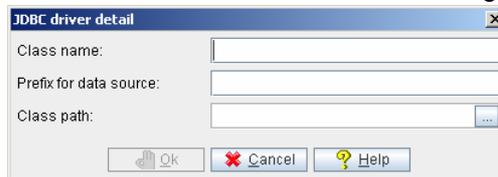
You may also edit or remove drivers that you have added. The default drivers cannot be removed or edited.

➔ **To add a new JDBC driver**

1. Select **Options > JDBC Drivers** from the menu bar. The **JDBC Drivers** window opens.



2. Click **Add**. The **JDBC driver detail** dialog opens.



3. Type a class name and prefix for data source into the appropriate text boxes.
4. Type a class path or click the browse button to navigate to the location of the driver.
5. Click **Ok**.
6. The new driver will appear in the list of JDBC drivers. Click **Ok**.

## 6.7 Mapping xpath statements to XTM object nodes

After creating a new XTM, the next step is to create a map between the nodes in the XML schema and the database tables. First you will need to add database tables to the XTM object tree, and then you will need to add XML schema paths to properly map from the address in the XML schema where the data is located.

To create the structure of the XTM object tree, drag-and-drop tables from the database into the desired location in the XTM object tree. The first node must be dropped onto the root node; subsequent nodes can be dropped into the root node or any other node that represents a table. Tables cannot be dropped into a node representing a column. When you drop a table, it will appear in the XTM column with all of the columns visible.

After the XTM object tree structure is complete, drag-and-drop nodes from the XML schema panel onto the desired node in the XTM object tree. A statement will appear in the xpath column that corresponds to the xpath representation of the element or attribute address in the XML schema. Furthermore, xpath statements will be preceded by a data property if they are numerical or binary data types. A data property statement will not precede character data types. The xpath data property and corresponding data types are summarized in the following table.

# XML Transfer Mapping Tool

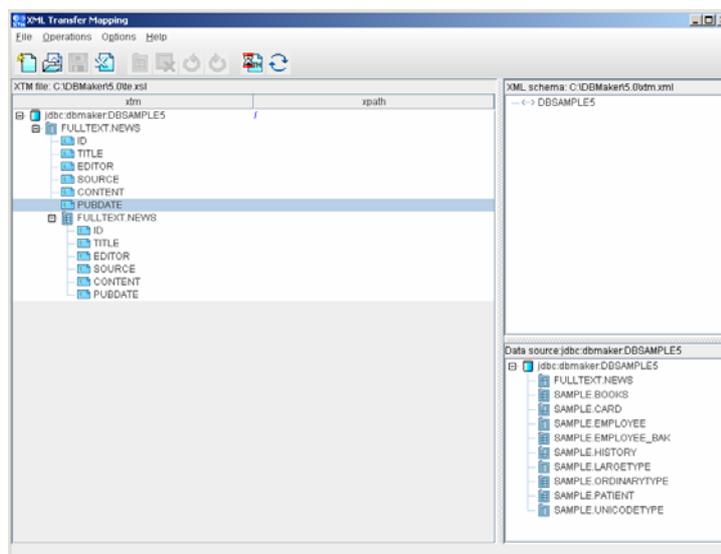
XPATH DATA PROPERTY	SQL DATA TYPE
Character data	Char
	Varchar
	Longvarchar
	Longvarbinary
	File
	Nchar
	Nvarchar
	Nclob
Numerical data	Serial
	Smallint
	Int
	Float
	Double
	Decimal
Normalize-space (binary) data	Binary
	Date
	Time
	Timestamp

Table 6-2: xpath data properties and corresponding SQL data types



## To build the structure of an XTM:

1. Create a new XTM or open an existing XTM:
2. Drag the table that you want to import data from the Database schema panel and drop to the xtm column of the XTM editing panel.



3. Drag-and drop the elements or attributes to the XTM nodes that you want to map the data to. The xpath statement appears in the xpath column.
4. When you are finished, click **File > Save**.
5. Browse to the folder where you want to store the XTM and type a file name. The file will automatically be saved with the extension .XSL.

## 6.8 Executing an XTM

After creating the XTM and saving it as an XSL file, you can pass data from the source XML file to the database by executing the XTM. When executing the XTM, you can choose to save the data as an XSL file and run the XTM, or save the transformation as an SQL script. Saving as an SQL script will not enter any data into the database. You must execute the script to enter the data into the database.

If you are trying to automate data transfer using a data transfer API or stored procedure, then you should save the transformation as an XSL file.

### 6.8.1 SAVING AN XTM AS AN SQL SCRIPT

Saving an XTM as an SQL script allows you to create an SQL script that will perform the same operation on the database as executing the XTM file, only it stores the equivalent SQL commands in a script file. This method will not actually store any data in the database. Be sure to save the XTM as an XSL file before performing this operation, as it will not save either XML data or the transformation other than in the form of the SQL script.

#### ➔ To execute an XTM and save output as an SQL script:

1. Click **Operations > Run**. The **Execute XTM** window will appear.
2. Click **Save as SQL Script**.
3. In the **Save as SQL Script** box, enter the full path and file name for the XSL file, or select a file and path by clicking the browse button.
4. In the **Source XML** box, enter a full path and file name for the XML file to import data from, or select a file and path by clicking the browse button.

- Click **OK**. The XTM Tool will create an SQL script. You can run the SQL script from the dmSQL prompt or using the JDBC tool to enter data into the database.

➔ **Example:**

SQL script output:

```
INSERT INTO DELPHI.CHINESE (ID,TEXT) VALUES (?,?);
1, 'lobdir1\clobfile0.txt';
2, 'lobdir1\clobfile1.txt';
3, 'lobdir1\clobfile2.txt';
4, 'lobdir1\clobfile3.txt';
5, 'lobdir1\clobfile4.txt';
...
```

## 6.8.2 SAVING AN XTM AS AN XSL FILE AND EXECUTING

Saving the XTM as an XSL file and executing performs the same operation as the XTM API or stored procedure. Executing the XTM file allows you to find errors in a transformation before you have automated it, and allows you to test the output for a given transformation to ensure that it produces the desired result.

➔ **To execute an XTM and save output as an XSL file:**

- Click **Operations > Run**. The **Execute XTM** window will appear.
- Click **Save as XTM and run**.
- In the **Save as XTM and Run** box, enter the full path and file name for the XSL file, or select a file and path by clicking the browse button.
- In the **Source XML** box, enter a full path and file name for the XML file to import data from, or select a file and path by clicking the browse button.
- Click **OK**. The XTM Tool will create an XSL file and add new data to the selected tables in the database.

➔ **Example:**

XSL output:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0"
xmlns:xtm="dbmaster.xml.xtm.XMLTransferMap"
extension-element-prefixes="xtm">
<xsl:template match="/">
<xtm:connect driver="dbmaster.sql.JdbcOdbcDriver"
datasource="jdbc:dbmaster:DBSAMPLE4">
<xtm:table owner="DELPHI" name="CHINESE" select="/root/CHINESE">
<xtm:column name="ID" select="number(@ID)"/>
<xtm:column name="TEXT" select="@TEXT"/>
</xtm:table>
</xtm:connect>
</xsl:template>
</xsl:stylesheet>
sel.xsl ', 'c:\temp\case1.xml');
```

## 7. XTM API Functions

After you have created an XTM file you are ready to automate the transfer process. The XTM API allows you to automate the process. DBMaster provides Java XTM transfer API and the XTM API stored procedure.

The XTM API requires a few common objects to process:

- Source XML file – the data source for XTM is kept in an XML file.
- XTM file – the file specifies how to retrieve data from source XML and save into the database.

In addition, the following properties may be requires depending on how you choose to implement the API:

- Database connection – you may optionally pass the JDBC/ODBC connection to the XTM engine. If specified, then it will use this connection and ignore the element 'xtm:connect' in the XTM file. A JDBC connection can be established using a JDBC driver provided by one of several vendors, but an ODBC connection should use the DBMaster ODBC connection handle.
- User name – the user name for database connection. In some cases, you might not want to add the user name to 'xtm:connect'. Alternatively, you may pass the user name when processing XTM file. If specified, then the user name in the XTM file be used instead of the one in 'xtm:connect'.
- Password – the password used to log into the database. Like the user name, you may pass the password to the XTM engine in an API argument.

The search rule for a database connection in XTM processing is:

1. The connection details are passed in the **setConnection** method.
2. Connection with the user/password passed in the **setUser / setPassword** method.
3. Connection information is specified in <xtm:connect> in the XTM file.

If the same property is set more than once before calling the run method, then only the last input value will be used.

### 7.1 XTM API in Java

The XTM processor also provides the Java class `dbmaster.xml.XTM`.

#### 7.1.1 PUBLIC METHODS:

---

##### Constructor :

Constructs a XTM object.

## Syntax

```
XTM( )
```

## setConnection

Sets the database connection. This functions accepts the java.sql.Connection object as an argument. The connection can be established via the DBMaster JDBC driver or a JDBC driver from other vender.

## Syntax

```
void setConnection( Connection conn );
```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
conn	Input	Object of Class java.sql.Connection (cannot be null).

## setUser

Sets the user for a database connection.

## Syntax

```
void setUser ( String user);
```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
user	Input	User name

## setPassword

Sets the user's password for a database connection.

## Syntax

```
void setPassword ( String password );
```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
password	Input	Password used for database connection

## setParameter

Set parameter for XSL processor. Parameter value will be available during XTM processing. It can be referenced in the syntax as ***\$parameter\_name***.

## Syntax

```
void setParameters ( String name, Object value );
```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
name	Input	Parameter name. Parameter name is a qualified name as a two-part string, the namespace URI enclosed in curly braces ({}), followed by the local name.

## XTM API Functions

		<p>If the name has a null URL, the String only contain the local name. An application can safely check for a non-null URI by testing to see if the first character of the name is a '{' character. For example, if a URI and local name were obtained from an element defined with <code>&lt;xyz:foo xmlns:xyz="http://xyz.foo.com/yada/baz.html"&gt;</code>, then the qualified name would be <code>{http://xyz.foo.com/yada/baz.html}foo</code>. Note that no prefix is used.</p>
value	Input	The parameter value.

### getError

If return code from `run()` is not SUCCESS, then use this method to get the error message. If no error, then this method will return NULL.

#### Syntax

```
String getError();
```

### run

The method `run()` will transform the XTM content. The XTM itself is actually an XSL file. The XTM engine is an extension of the Apache Xalan XSLT processor. The transformer will transform the source XML file according to the specification in the XTM file.

#### Syntax

```
int run ( String xtmfile,
         String xmlfile);
```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
xtmfile	Input	XTM filename. It cannot be null.
xmlfile	Input	Source XML filename. It cannot be null.

RETURNS	DESCRIPTION
SUCCESS	Engine successfully processed source XML file with XTM specification.

RETURNS	DESCRIPTION
ERR_XTM_PROCESS	Possible reasons for XTM process errors include:  XSTL process error – XTM or source XML file is not valid XML, or the XTM is not a valid XSL file.  XTM syntax error.
SQL Error	The SQL command produced an error.

## 7.1.2 EXAMPLE:

```

/* create new XTM object */
XTM transfer = new XTM();

/* start transformation */
int rc = transfer.run(xtm, xml);

/* check if process fails.
   if so, getError for error detail. */
if ( rc != 0 )
{
    System.out.println(transfer.getError()+"\n");
}

```

## 7.2 XTM Stored Procedure

The XTM stored procedure provides an XTM transformation function similar to the Java APIs. The main differences are

- It only accepts XTM and source XML filenames.
- It only uses a current database connection.
- The filenames specified will be on the server site.

### 7.2.1 STORED PROCEDURE DEFINITION:

#### Syntax

```

CREATE PROCEDURE XTM(
    VARCHAR(257) XTMFILE INPUT,
    VARCHAR(257) XMLFILE INPUT );

```

ARGUMENT	INPUT/OUTPUT	DESCRIPTION
XTMFILE	Input	XTM filename
XMLFILE	Input	Source XML filename

### 7.2.2 PRIVILEGE

The owner of this stored procedure is SYSTEM. The privileges of the stored procedure are the same as the user executing the stored procedure. For example, if there is an insert statement in an

XTM file to a table on which the user does not have the privilege to insert, then an SQL error will be returned.

### 7.2.3 EXAMPLE

---

```
dmSQL> call XTM('c:\temp\case1.xml ', 'c:\temp\case1.xml');
```

## 8. XML Type Index and Predicate

### 8.1 Managing Index

An index is a mechanism that provides fast access to specific rows in a table based on the values of one or more columns from the table (known as the key). Indexes contain the same data as the key columns, but the data is structured and sorted to make retrieval much faster. Once an index is created on a table, its operation is transparent to users of the database. The DBMS uses the index to improve query performance whenever possible.

An index can be composed of more than one column, up to a maximum of 32. All the columns in a table can be used in an index.

### 8.2 Creating Indexes on XML column

The CREATE INDEX command creates a new index on an existing table. Use indexes to increase the performance of queries by quickly locating specific rows in a table without examining the entire table.

The following figure is creating index syntax.

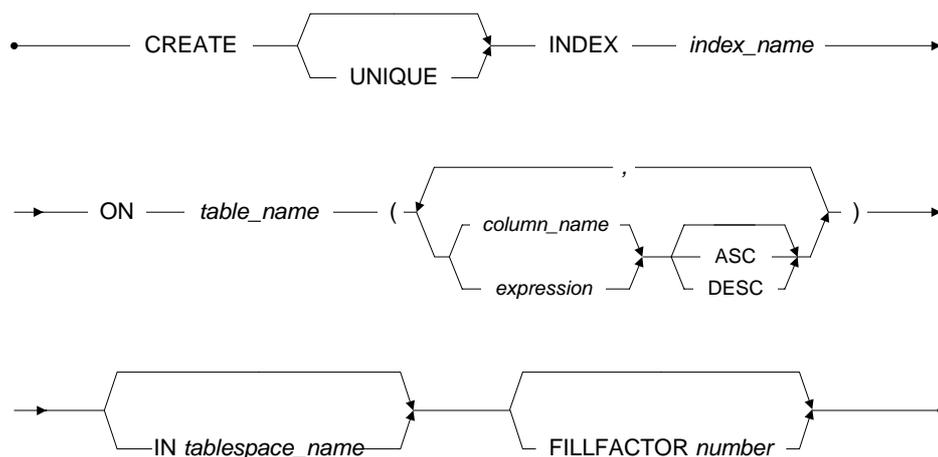


Figure 8-1 CREATE INDEX syntax

When creating an index specify the index name, the name of the table creating the index on, and the name of the key columns in the table. Create an index on one or more columns, up to a maximum of 32 columns. Any column in a table can be used in an index. DBMaster limits indexes to a maximum record size of 4000 bytes.

Creating indexes for frequently used expressions will improve query performance. For XML columns, create the index on XML UDF: `extract()` and `extractvalue()` to speed up xpath queries. Please note the primary differences between `extract()` and `extractvalue()`. **Extract()** allows multi-value, one value, or zero value results, however, `asc/desc` and unique index are not allowed. **Extractvalue()** only allows UDF results having one value or zero values. If the UDF result is multi-value, then the create index fails for the existing tuple and the insert data fails for the newly inserted tuple, however, `asc/desc` and unique index are allowed with **Extractvalue()**.

The following example shows how to create an index on an XML column using dmSQL. Please see the *SQL Command and Function Reference* for additional details on the syntax and usage of the SQL command `CREATE INDEX`.

## ➤ Example 1

To create an index use the **Extract** XML UDF:

```
CREATE TABLE tb(id int primary key, name char(20),c1 XMLTYPE);
```

```
Create index idx1 on tb (extract(c1, '/personnel/person/@id', NULL));
```

## ➤ Example 2

To create an index use the **ExtractValue** XML UDF:

```
Create index idx_extrV on tb_extract (extractValue(id, '/order/items/item/@product', NULL) asc);
```

The primary difference between **Extract()** and **Extractvalue()** are:

### Extract()

•————— EXTRACT () —————•

Figure 8-2 EXTRACT syntax

- allows a multi-value, a single value or zero value result.
- `asc / desc` are not allowed.
- unique index is not allowed.

### ExtractValue()

•————— EXTRACTVALUE () —————•

Figure 8-3 EXTRACTVALUE syntax

- allows a single value or a zero value of the UDF results (when the UDF result is a multi-value result then the create index fails for existing tuple and the insert data fails for newly inserted tuple).
- allows `asc / desc`.
- allows unique index.

## 9. XML Validate UDF

XML has become a key technology of today's Internet. Its self-description and extensibility have offered the flexibility for data exchange. DBMaster will support this new data type 'xmltype' and some XML related functions for user to store and manage XML data.

XML Type is a subset of media type. Large object columns may also be specified as media types to aid in media process functions.

We will provide some examples of how to create xml validate udf and how to plug-in udf for media type validation clause in the following chapter.

DBMaster will support a new data type XMLTYPE and provides the feature as following:

- Well-formed XML checking: inserted/updated xml content must be well-formed.
- XML validation: User might specify validation udf when create xmltype column. DBMaster will validate xml content with it.
- XML data will be stored in its original format.
- Query with XPath search: User might specify xpath and use extract functions to query/locate nodes in an XML data.
- Update XML content specified by XPath.

Build index on XPath extract: To speed up xpath query, user could build index on frequent query xpath expression.

### 9.1 Create DTD/XML Validate UDF

There are two ways to define the legal building blocks of an XML document. One is DTD and another is XML Schema. There are several parser generators or XML binding class generators on the market. For example, flexml is XML processor generator, which would process DTD and generate flex file. Then user can build XML processor base on the flex file. There are also a few commercial products such as Liquid XML 2006, which can build XML binding classes in different coding language (java).

#### 9.1.1 FLEXML

---

Kristoffer Rose's flexml distributed under the GNU General Public License, is an XML process generator. It takes a DTD file and generates a LEX file. Flexml is available at <http://flexml.sourceforge.net>.

#### Generating the LEXFile

```
$ flexml name.dtd
```

## Adding Customized YY\_INPUT

The original LEX input is a FILE input stream. The LEX file must be modified to use UDF Blob as an input source. The following example demonstrates this modification by adding customized YY\_INPUT.

### ➡ Example

Modify the definition section of the LEX file by adding YY\_INPUT as shown below. The definition section is located at the beginning of the file and between the “%{“ and ”}%” markers

```
#include "libudf.h"

typedef struct udf_file
{
    VAL *args;
    i31 handle;
    i31 rc;
    i31 left;
    i31 rlen;
} UDF_FILE;

#undef YY_INPUT
#define YY_INPUT(buf,result,max_size) {\
    UDF_FILE * uf =(UDF_FILE *)yyin; \
    errno=0; \
    if ( uf->left <= 0 )\
    {\
        result = (uf->rlen=0);\
    }\
    if ( (uf->rc = _UDFBbRead(uf->args, uf->handle, max_size, &(uf->rlen),\
buf))!=0 ) \
    { \
        errno = uf->rc; \
        result = 0;\
    }\
    else\
    {\
        uf->left -= uf->rlen;\
        result = uf->rlen;\
    }\
}\
}
```

Next, add the UDF function to the end of the LEX file as shown below:

```
#ifdef WIN32
__declspec(dllexport)
#endif
int XXX_VALIDATE(int nArg, VAL args[])
{
    BBObj bbin;
    UDF_FILE uf;
    int rc = 0;
    int rc2 = 0;
    memset(&uf, 0, sizeof(UDF_FILE));
    memcpy((char *)&bbin, args[0].u.xval, BBOBJ_SIZE);
}
```

```

uf.args = args;
rc = _UDFBbOpen( args, bbin, &(uf.handle));
if( rc != 0 )
    goto EXIT;
if (rc = _UDFBbSize(args, bbin, &(uf.left)) )
    {
        goto EXIT;
    }
yyin = (void *)&uf;
rc = validdtd00udf();
rc2 = _UDFBbClose(args, uf.handle);
EXIT:
if( args[0].type != NULL_TYP ) // null column data
    {
        args[0].type = INT_TYP;
        args[0].len = 4;
        args[0].u.ival = (rc == 0? 1:0);
    }
return _RetVal(args, args[0]);
}

```

## Build dll/so

flex *name.l*

```
cc -c -DBUILD_DLL lex.name.c -Idbmaster-installed-dir/include
```

## Create UDF

```
CREATE FUNCTION dllname.udfname(BLOB) returns int;
```

## Create column with check constraint

```
CREATE TABLE table-name( c1 XMLTYPE CHECK udfname(value) = 1);
```

## 9.1.2 DBMASTER DTD VALIDATION UDF GENERATOR

A command line tool will generate a validation UDF for the specified DTD.

```
$ dmxmludfmk -dtd dtd-file-name [-o output-directory] [-p prefix]
```

- DTD file name is required input. If not specified, an error is generated.
- Output directory is optional. If not specified, the files would be created under current working directory.
- The prefix is optional. If specified, the generated file uses the prefix in the filename. If not specified, the DTD filename without a file extension is used.

Several files are generated as follows;

- Lex file: < user-specified-prefix.l>
- Yacc file: <user-specified-prefix.y>
- UDF function file: <user-specified-prefix> udf.c and <user-specified-prefix> udf.h
- The UDF function is named as <user-specified-prefix>\_VALIDATE
- hash.c and hash.h provide hash functions
- Makefile on UNIX platformsx or Makefile.msvc on Windows platforms

On UNIX, type 'make' to compile/link [user-specified-prefix].so. On windows, do 'nmake /f Makefile.msvc' to compile/link *user-specified-prefix.dll*.

The difference between dmxmludfmk and flexml are

- dmxmludfmk support not only ascii but also big5, gb, shiftJIS and utf8.
- flexml support content replacement for internal defined entity in DTD.

**NOTE** *dmxmludfmk supports only DTD but no XMLSchema in current version.*

### ➤ Example 1

```
Make <user-specified-prefix>.so ;for UNIX
```

### ➤ Example 2

```
Nmake /f Makefile.msvc ;for Windows
```

**NOTE** *Please note that dmxmludfmk supports ASCII, and BIG5, gb, shiftJIS, and utf8 while flexml supports content replacement for internally defined DTD only entities.*

## 9.1.3 DEFAULT VALIDATOR

I\_VALIDATE is provided as a default validator for checking the XML's syntax. I\_VALIDATE does not provide validation against the DTD or the XMLSchema. I\_VALIDATE is part of libmedia library.

## 9.2 Add XMLType column

Currently, user can add a new XMLType column, but can't alter an existing xmltype column from one schema to another, or from no schema specified to one. In this version, DBMaster not support alter xmltype column or alter other data type to xmltype.

```
column-definition ::=
    column-identifier data-type [NULL | NOT NULL]
    [default-clause]
    | [col-key-definition]
    | [check-column-constraint]
column-identifier xmltype [CHECK VALIDATE_FUNCTION_NAME (value)=1]
```

## 9.3 Query XMLType column

DBMaster provides several extract functions, which allow user to use XPath to retrieve/query xmltype column data.

The result of XPath might be one of the four types: nodeset, boolean, number, and string. But can't have various returned type for extract function. For easy access to the XPath query result, DBMaster support three XPath query functions: **extract**, **extractValue** and **existsNode**.

### 9.3.1 EXTRACT

Function **extract** is the most general one, which will serialize result into NCLOB.

```
create function libmedia.EXTRACT( long varbinary, varchar(512), varchar(512) ) return
NCLOB;
```

#### Input argument

- args[0]: xmldata is the xml content to be queried.

- args[1]: xpath-expression is the xpath user will use to query xmldata.
- args[2]: namespaces is used to specify the namespace(s) used in xpath-expression. It is optional.

The syntax in namespaces field;

```
prefix="namespace-URI" prefix="namespace-URI"...
```

XPath specification has no default namespace. All the namespaces must be associated with prefixes. Therefore, if there is a namespace even it's default namespace in XML, we need to have a pair of prefix and namespace in XPath expression in order to properly point out and retrieve xml content.

## Return Value

The result will be serialized into NCLOB. A nodeset might have nodes in various node types (element, attribute, text, comment...). To serialize such a nodeset into NCLOB will be an issue. Therefore, we won't support XPath which would generate nodes in various node types.

The rules to serialize nodeset into NCLOB

- element node → traverse all the element nodes in the nodeset, print these element nodes and their sub nodes into NCLOB in the format of <tag attr1=value ...> ...</tag>. Add new line when finish each element node in the nodeset.
- attribute node → loop through all attribute nodes in the nodeset, print each attribute in the format of att1=value1. Add new line when finish each attribute node.
- text node → loop through all text nodes in the nodeset, print each text in the format text\_value. Add new line after each text\_value.
- comment node → loop through all comment nodes in the nodeset, print comment in the format of <!--comment content -->.

## Index on extract function

In order to speed up query, user can create index on frequent used xpath-expression.

```
CREATE [UNIQUE] INDEX index-name ON ( extract( xmldata, xpath-expression, namespaces) )
```

To build acceptable useful index, there are some rules for XPath used in extract:

- The XPath shouldn't have any predicate.
- The XPath shouldn't have any function in it.
- The XPath should have absolute location path.
- Only allow 'child' axis.
- All the nodes in the result nodeset should be the leaf. It can be simple type element node or attribute node.
- If it's element node, the name of each node must be the same.
- If it's attribute node, the name of each attribute must be the same.

### 9.3.2 EXTRACTVALUE

Function **extractValue** is used if the result has only one value.

```
create function libmedia.EXTRACTVALUE( long varbinary, varchar(512), varchar(512) )
return nstring;
```

Function `extractValue` is similar with `extract`. But it will return just the value of a node in the nodeset. Here are some rules for acceptable xpath for `extractValue()`:

- If the result of XPath is a nodeset, it must have only one node.
- This node can be element, text, attribute or comment.
- If it's element node, it must have no sub element. The returned value is its text.
- If it's text node, the returned value is the text.
- If it's attribute node, the returned value is the attribute value.
- If it's comment node, the returned value is the comment text.
- If the result of XPath is boolean value, it would either return "TRUE" or "FALSE".
- If the result of XPath is string value, it would simply return the string value.
- If the result of XPath is numeric value, it would return in string form.
- The returned type is NCHAR\_TYP.

### 9.3.3 EXISTSNODE

---

Function `existsNode` is used to check if specified node is found.

```
create function libmedia.EXISTSNODE( long varbinary, varchar(512), varchar(512) )
return int;
```

The function's return value is true or false.

- If the result of XPath is boolean, number or string, return true.
- If the result of XPath is a nodeset, then return true if it has at least one node. Otherwise, return false.

**NOTE** *We always process xml data in space-normalized mode. So any xml fragment retrieved might not be exact same as the original xml data. Those exceeding spaces will be ignored.*

## 9.4 Update XMLType column

Use XPath to locate the part of xml data to be updated.

```
create function libmedia.XMLUPDATE( long varbinary, varchar(20), varchar(512),
varchar(512), string ) return long varbinary;
```

Input argument

- `args[0]`: xmldata is the xml content to be updated.
- `args[1]`: modification-type
- `args[2]`: xpath-expression is used to locate the part in xmldata to be updated.
- `args[3]`: namespaces is used to specify the namespace(s) used in xpath-expression. It is optional.
- `args[4]`: value is the value to replace the content located by xpath.

Return Value

The xpath evaluated result must be a nodeset. `XMLUpdate()` will return the whole xml document after updated.

### 9.4.1 INSERT-BEFORE

- All the nodes in the result nodeset can be any node type but attribute.
- The input parameter 'value' should be an element fragment.
- Insert the element fragment before each node in the result nodeset.

#### Example :

```
dmSQL> create table xmltb(c1 int,c2 xmltype);
dmSQL> insert into xmltb values(?,?);
dmSQL/Val> 1,&'card.xml';
1 rows inserted
dmSQL/Val> 2,&'card.xml';
1 rows inserted
dmSQL/Val> END;
```

The following is card.xml template:

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE DBSAMPLE5 SYSTEM "DBSAMPLE5.dtd">
<!--
    Generated by DBMaster2XML V1.0
    Database: DBSAMPLE5
-->
<DBSAMPLE5>
<!-- Query string : select ID, FNAME, LNAME, TITLE, PHOTO from "SAMPLE"."CARD"
-->
<CARD>
  <ID>1</ID>
  <FNAME>Eddie</FNAME>
  <LNAME>Chang</LNAME>
  <TITLE>Manager</TITLE>
  <PHOTO>&BLBTMP_TMP0;</PHOTO>
</CARD>
<CARD>
  <ID>2</ID>
  <FNAME>Hook</FNAME>
  <LNAME>Hu</LNAME>
  <TITLE>Software Engineer</TITLE>
  <PHOTO>&BLBTMP_TMP1;</PHOTO>
</CARD>
<CARD>
  <ID>3</ID>
  <FNAME>Jackie</FNAME>
  <LNAME>Yu</LNAME>
  <TITLE>Software Engineer</TITLE>
  <PHOTO>&BLBTMP_TMP2;</PHOTO>
</CARD>
<CARD>
  <ID>4</ID>
  <FNAME>Ray</FNAME>
  <LNAME>Sung</LNAME>
  <TITLE>Software Engineer</TITLE>
  <PHOTO>&BLBTMP_TMP3;</PHOTO>
```

```

</CARD>
<CARD>
  <ID>5</ID>
  <FNAME>Louis</FNAME>
  <LNAME>Liu</LNAME>
  <TITLE>Software Engineer</TITLE>
  <PHOTO>&BLBTMP_TMP4;</PHOTO>
</CARD>
<CARD>
  <ID>6</ID>
  <FNAME>Trent</FNAME>
  <LNAME>Clowater</LNAME>
  <TITLE>Software Engineer</TITLE>
  <PHOTO>&BLBTMP_TMP5;</PHOTO>
</CARD>
<CARD>
  <ID>7</ID>
  <FNAME>Oscar</FNAME>
  <LNAME>Tseng</LNAME>
  <TITLE>Software Engineer</TITLE>
  <PHOTO>&BLBTMP_TMP6;</PHOTO>
</CARD>
<CARD>
  <ID>8</ID>
  <FNAME>Jerry</FNAME>
  <LNAME>Liu</LNAME>
  <TITLE>Manager</TITLE>
  <PHOTO>&BLBTMP_TMP7;</PHOTO>
</CARD>
</DBSAMPLE5>

```

```

update XMLTB set c2 = xmlupdate(c2, 'insert-before', '/DBSAMPLE5/CARD[1]/ID', NULL,
'<NEWID>11</NEWID>') where c1 = 2;

```

## 9.4.2 INSERT-AFTER

- All the nodes in the result nodeset can be any node type but attribute.
- The input parameter 'value' should be an element fragment.
- Insert the element fragment after each node in the result nodeset.

➔ **Example :**

```

update XMLTB set c2 = xmlupdate(c2, 'insert-after', '/DBSAMPLE5/CARD[2]/TITLE', NULL,
'<NEWNODE>123-4567</NEWNODE>') where c1 = 2;

```

## 9.4.3 INSERT-ATTRIBUTE

- All the nodes in the result nodeset must be element nodes.
- The input parameter 'value' should contain a pair of name and value in the syntax; attribute-name=attribute-value.
- Insert attribute for each of the element nodes in the result nodeset. Attribute name should be unique for an element. And there shouldn't be any sequence issue.

➔ **Example :**

```
update XMLTB set c2 = xmlupdate(c2, 'insert-attribute', '/DBSAMPLE5/CARD/NULLID',
NULL, '<NULLID>') where c1 = 2;
```

#### 9.4.4 INSERT-TEXT-BEFORE

---

- All the nodes in the result nodeset can be any node type but attribute.
- The input parameter 'value' should be text node value.
- Insert text node before each node in the result nodeset.

➔ **Example :**

```
update XMLTB set c2 = xmlupdate(c2, 'insert-text-before',
'/DBSAMPLE5/CARD[3]/FNAME/text()', NULL, '<insert-text-before>') where c1 = 2;
```

#### 9.4.5 INSERT-TEXT-AFTER

---

- All the nodes in the result nodeset can be any node type but attribute.
- The input parameter 'value' should be text node value.
- Insert text node after each node in the result nodeset.

➔ **Example :**

```
update XMLTB set c2 = xmlupdate(c2, 'insert-text-after',
'/DBSAMPLE5/CARD[4]/FNAME/text()', NULL, 'insert-text-after') where c1 = 2;
```

#### 9.4.6 APPEND-TEXT

---

- All the nodes in the result nodeset must be element nodes.
- The input parameter 'value' should be the text node value.
- Append the text node as sub node of each of the element nodes in the result nodeset.

➔ **Example :**

```
update XMLTB set c2 = xmlupdate(c2, 'append-text', '/DBSAMPLE5/CARD[2]/FNAME', NULL,
'<append-text>') where c1 = 2;
```

#### 9.4.7 APPEND

---

- All the nodes in the result nodeset must be element nodes.
- The input parameter 'value' should be an element fragment.
- Append the element fragment as sub element of each of the element nodes in the result nodeset.

➔ **Example :**

```
update XMLTB set c2 = xmlupdate(c2, 'append', '/DBSAMPLE5/CARD[1]', NULL, '<append/>')
where c1 = 2;
```

#### 9.4.8 UPDATE

---

- All the nodes in the result nodeset must be in the same node type. The acceptable node types are element, text and attribute node.
- If node type is element, the input parameter 'value' should have an element fragment <tag>...</tag>. The value can't be multiple nodes. Value must be either

NULL or a valid XML fragment. NULL value will make the element become an empty one.

- If node type is text, the input parameter 'value' should have text value only. Any '<' and '>' will be replaced as '&lt;' and '&gt;'. NULL value will remove the text value from the element.
- If node type is attribute, the input parameter 'value' should have the attribute value only. Any '<' and '>' will be replaced as '&lt;' and '&gt;'. NULL value will set the attribute value to empty string.

➤ **Example :**

```
update XMLTB set c2 = xmlupdate(c2, 'update', '/DBSAMPLE5/CARD[5]/FNAME/text()', NULL, 'update') where c1 = 2;
```

## 9.4.9 REMOVE

---

- Remove all the nodes in the result nodeset.

➤ **Example :**

```
update XMLTB set c2 = xmlupdate(c2, 'remove', '/DBSAMPLE5/CARD/FNAME', NULL, NULL) where c1 = 2;
```

## 9.4.10 RENAME

---

- All the nodes in the result nodeset must be in the same node type. The possible node types are element and attribute node.
- The input parameter 'value' should be XML qualified name.
- Rename nodes in the result nodeset.

➤ **Example :**

```
update XMLTB set c2 = xmlupdate(c2, 'rename', '/DBSAMPLE5/CARD/ID', NULL, 'RENAME-ID') where c1 = 2;
```

**NOTE** *To simplify interface, DBMaster only support insert/append element, text, and attribute node. All other node types like comment, processing-instruction, are not supported in current design.*

## 10. JData Transfer Tool

The Data Transfer Tool is a separate application, you can view it as extension to import/export xml text; it may be opened from windows start>programs>DBMaster 5.0>JData Transfer, or opened from within JDBA Tool. It consists of a main console and a menu bar, as illustrated in Figure. The main console provides five options: import from text, import from XML, and import from ODBC, export to text, and export to XML. The Menu Bar consists of three menus: the Transfer menu, the Option menu, and the Help menu. The Transfer menu provides the same transfer functions as the main console, with the addition of the batch transfer function. The option menu can be used to change the language that the UI is displayed in; currently English, Japanese, and Chinese (traditional) are the supported languages. The help menu provides access to the help system for JDBA Tool.



### ☞ To open the Data Transfer Tool:

1. Start JDBA Tool and connect to the database that data is to be transferred to or from.
2. Select **Data Transfer** from the Tool menu. The Data Transfer tool window will open

### 10.1 Importing data from XML

XML files may also be imported into the database. XML tags may first be defined in a *Document Type Definition* (DTD) file before being imported into the database. Furthermore, the DTD may define the schema in a way that is acceptable to the database.

It is important to consider the structure of the XML file you wish to import. To ensure that the structure of the XML file and associated DTD have compatible structure, examine the structure of XML files produced by the Data Transfer Tool: Export to XML File wizard. Files produced using the Export to XML wizard always can be imported; however, the extent to which a table's schema is reproduced varies. The setting that influences table schema the most is the Column as Element / Attribute setting. Make sure the destination database is started.

- Column as Element: Stores data items in elements. If table schema information exists as element attributes (data type, column name, length, etc.) in the DTD, then

columns will be created with names and of the appropriate data type and length. Columns are child elements, and the table is represented as the parent element. File objects must be referenced as entities in the DTD file if Column as Element is chosen.

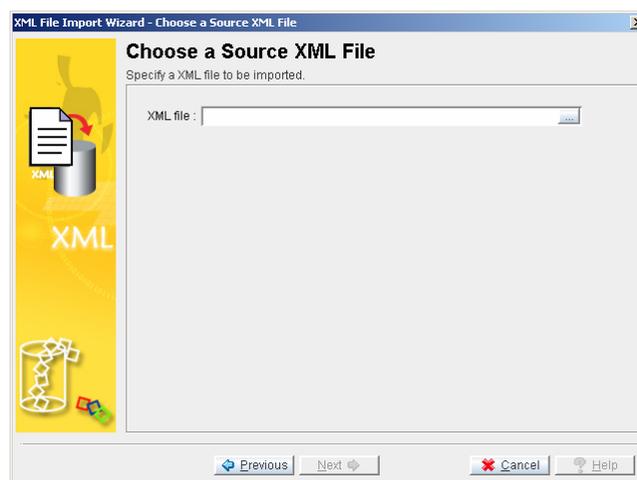
- **Column as Attribute:** Stores data in an attribute of an element. Each element is a record. If column names are represented as attributes of the root element (the table) in the DTD, and each tag in the XML file represents one record, then Column as Attribute should be chosen.

## ☞ To import data from an XML file:

1. Open the Data Transfer Tool.
2. Select **Import XML File** from the main console or the **Transfer** menu. The **Welcome to Import from XML File Wizard** window will open, displaying a summary of the steps to be taken in the wizard.



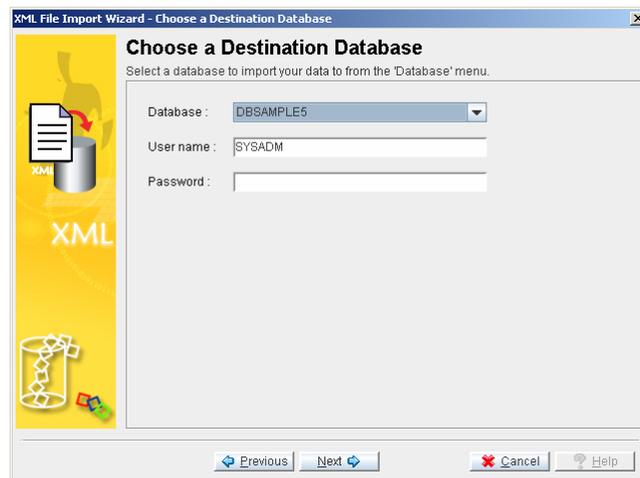
3. Click **Next**. The **Choose a Source XML File** window will open.



4. Enter the full path of a text file to import or click the browse button to search for a text file.
5. Click **Next**. If the XML file has a structure acceptable to DBMaster's parser, the **Specify Base Element** window will open.



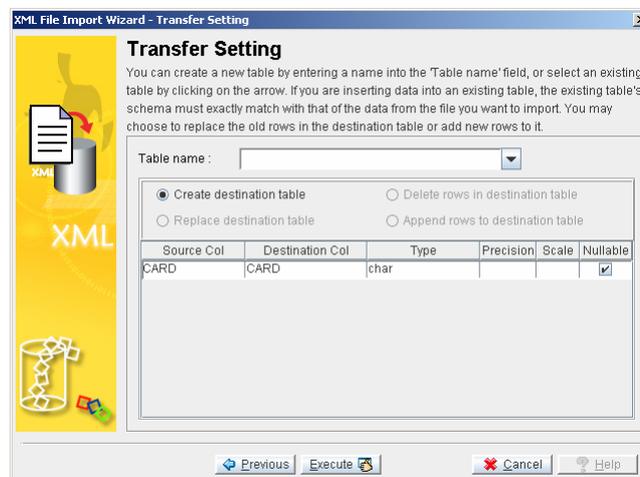
6. The nodes of the tree structure represent the elements in the XML file. Click the nodes on the tree until they are fully expanded. Select a parent element to be the table name. The child elements will become the columns of the table. Check Column as attribute if appropriate.
7. Click **Next**. The **Choose a Destination Data Source** window will open.



8. Select the database to import data to from the **Database** menu.
9. Enter a user name and password into the appropriate fields.

**NOTE** DBA authority or higher is required to import a text file.

10. Click **Next**. The **Transfer Setting** window will open.



11. Enter a new **table name** into the Table Name field, or select a table from the menu. Selecting a table from the menu will allow you to choose to replace the

destination table, delete rows in the destination table, or append new rows to the destination table. Click **Execute** to import the XML file. A confirmation dialog box will appear.

12. Click **OK**

## 10.2 Exporting data to XML

DBMaster supports the export of data from a table to an XML file. Columns may be stored as individual elements, or as attributes of the table element. When an XML file is created, an associated DTD file is created. The DTD contains information necessary for defining the elements and attributes of the XML file. The structure of both the DTD and XML file will vary depending on whether the columns are stored as attributes or elements.

Consider how the following settings affect the XML file produced by the Export to XML wizard.

- **Column as Element:** If columns are represented as elements in the resultant files, then schema information will be retained as element attributes (data type, column name, length, etc.) in the DTD. Columns are child elements, and the table is represented as the parent element. If the XML file is later imported back into the database, then the table's structure will be exactly replicated. File objects are referenced as entities in the DTD file if Column as Element is chosen.
- **Column as Attribute:** Columns are represented as attributes of the table element in the DTD. There is no record of the table's schema. An element in the XML file represents each record.
- **Export file link name for FILE type data:** The original full path will reference system and user file objects if this option is selected. If this option is not selected, file type data will be treated as Long Varbinary.
- **Translate all tag names to uppercase:** All tag names are converted to uppercase characters.
- **Build temp file to store LONGVARCHAR and LONGVARBINARY data type column constant:** If this option is chosen, BLOB data will be stored in a temporary directory under the directory the XML file resides in. If this option is not selected, BLOB data is stored directly in the XML file.
- **XML file cannot include DTD file reference:** if this option is selected, no DTD is created. No information about the elements will be preserved in the DTD if this option is selected.

### ➔ Example 1

Assume the table 'supportqueries' with columns 'LOGINID' CHAR(10); 'REQUEST' SQL\_LONGVARCHAR; 'REQUESTTIME' SQL\_TIMESTAMP; 'ATTACHMENT' 'SQL\_FILE'; 'BINARY\_C' SQL\_BINARY(10); 'DECIMAL\_C' SQL\_DECIMAL(10, 3). The table has two records. The entire table is exported to an XML file with columns as elements. File link names are exported, temp files are built to store BLOB data, and the DTD is included. The resulting XML file follows:

```
<?xml version="1.0" encoding="BIG5"?>
<!DOCTYPE WEBDB SYSTEM "Support.dtd">
<WEBDB>
  <SUPPORTQUERIES>
    <LOGINID>A_HOWARD          </LOGINID>
    <REQUEST>&BLBTMP_TXT0;</REQUEST>
    <REQUESTTIME>2001-09-09 12:47:05.000</REQUESTTIME>
```

```

<ATTACHMENT>&DBMASTER_FO_0;</ATTACHMENT>
<BINARY_C>10000000000000000000</BINARY_C>
<DECIMAL_C>10.250</DECIMAL_C>
</SUPPORTQUERIES>
<SUPPORTQUERIES>
  <LOGINID>A_HOWARD      </LOGINID>
  <REQUEST>&BLBTMP_TXT1;</REQUEST>
  <REQUESTTIME>2001-09-22 10:14:21.000</REQUESTTIME>
  <ATTACHMENT>&DBMASTER_FO_1;</ATTACHMENT>
  <BINARY_C>20000000000000000000</BINARY_C>
  <DECIMAL_C>13.550</DECIMAL_C>
</SUPPORTQUERIES>
</WEBDB>

```

The associated DTD follows:

```

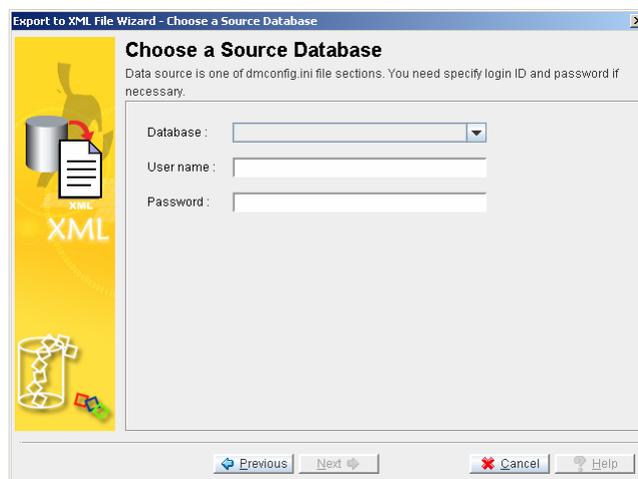
<!ELEMENT SUPPORTQUERIES (LOGINID, REQUEST, REQUESTTIME, ATTACHMENT, BINARY_C,
DECIMAL_C)>
<!ELEMENT LOGINID (#PCDATA)>
  <!ATTLIST LOGINID
    TYPE CDATA #FIXED "SQL_CHAR"
    NAME CDATA #FIXED "LOGINID"
    LENGTH CDATA #FIXED "20"
    ISNULL (true|false) 'true'
    xml:space (default|preserve) 'preserve'
  >
<!ELEMENT REQUEST (#PCDATA)>
  <!ATTLIST REQUEST
    TYPE CDATA #FIXED "SQL_LONGVARCHAR"
    NAME CDATA #FIXED "REQUEST"
    ISNULL (true|false) 'true'
    xml:space (default|preserve) 'preserve'
  >
<!ELEMENT REQUESTTIME (#PCDATA)>
  <!ATTLIST REQUESTTIME
    TYPE CDATA #FIXED "SQL_TIMESTAMP"
    NAME CDATA #FIXED "REQUESTTIME"
    STORAGE CDATA #FIXED "29"
    ISNULL (true|false) 'true'
    xml:space (default|preserve) 'preserve'
  >
<!ELEMENT ATTACHMENT (#PCDATA)>
  <!ATTLIST ATTACHMENT
    TYPE CDATA #FIXED "SQL_FILE"
    NAME CDATA #FIXED "ATTACHMENT"
    ISNULL (true|false) 'true'
    xml:space (default|preserve) 'preserve'
  >
<!ELEMENT BINARY_C (#PCDATA)>
  <!ATTLIST BINARY_C
    TYPE CDATA #FIXED "SQL_BINARY"
    NAME CDATA #FIXED "BINARY_C"
    LENGTH CDATA #FIXED "10"
    ISNULL (true|false) 'true'

```

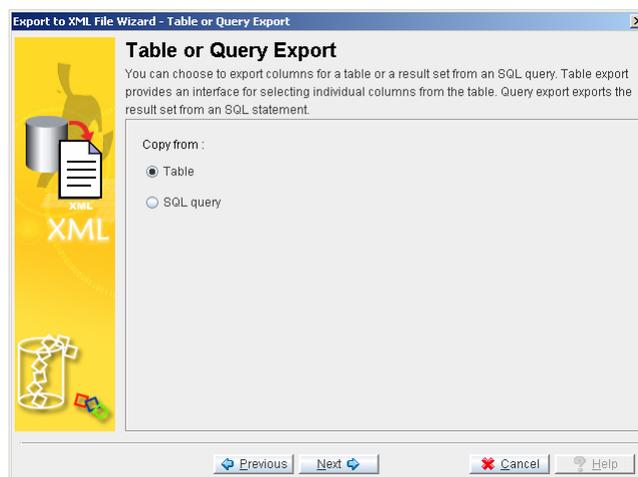




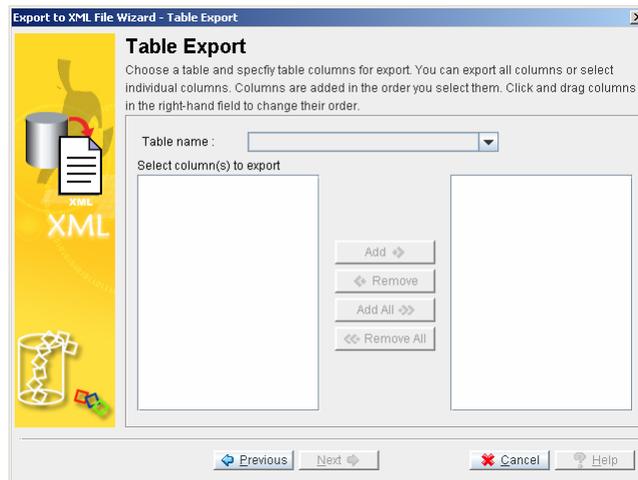
3. Click **Next**. The **Choose a Data Source** window will open.



4. Select a database from the **Database** menu. Enter a user name and password into the appropriate fields.
5. Click **Next**. The **Table or Query Export** window will appear.



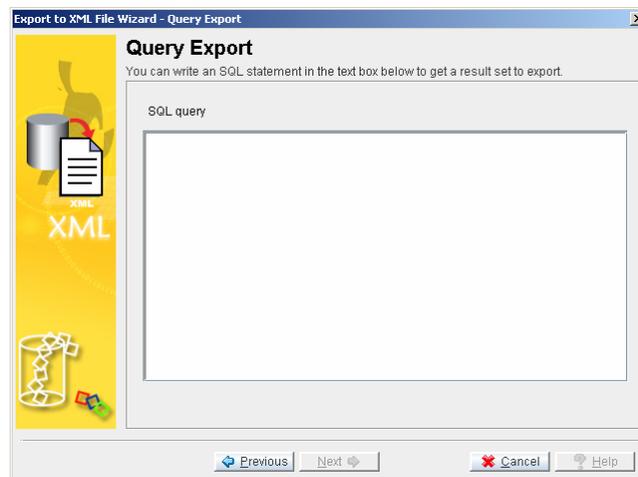
6. If you selected **Table** from the **Table or Query Export** window, the **Table Export** window will open. If you selected **SQL query**, then proceed to step 13.
7. Click **Next**, the **Table Export** window will open.



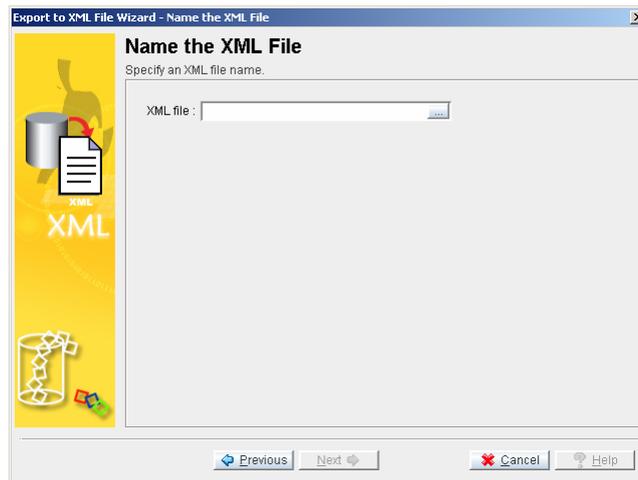
8. Select a table to export from the **Table name** menu. A list of columns in the table will appear in the **Select columns to export** field.
9. Select columns by clicking on the column name and clicking **Add**, or select all columns by clicking **Add All**. Selected column names will appear in the right hand field.



10. Click **Next**. The **Name The XML File** window will appear (proceed to step 16).
11. If you selected **SQL query** from the **Table or Query Export** window, the **Query Export** window will open.
12. Enter a valid SQL select statement into the **SQL query** field.



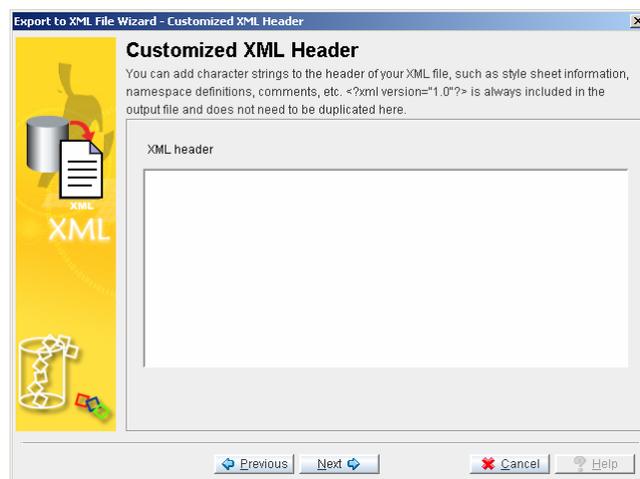
13. Click **Next**. The **Name The XML File** window will appear.



14. Enter the full path of an XML file to export to, or select one by using the browse button.
15. Click **Next**. The **XML File Format Setting** window will open.



16. Select the appropriate settings for the format of the XML file you will create.
17. Click **Next**. The **Customized XML Header** window will open.



18. Enter appropriate information, such as namespace and style sheet definitions, if relevant.
19. Click **Next**. The **Customized Element Name** window will appear.



20. It is possible to modify the tag definitions. Enter new tag definitions into the **Replace Tag Name** column. The name of the corresponding column will be changed in the resulting XML file.
21. Click **Execute** to export the table to the XML file. A confirmation window will appear.
22. Click **OK**.