



arbortext®

Tutorial for Arbortext Import

Contents

| | |
|--|----|
| About This Guide | 5 |
| Prerequisite Knowledge | 5 |
| Technical Support | 6 |
| Documentation for PTC Products | 6 |
| Global Services | 6 |
| Comments | 7 |
| Documentation Conventions | 7 |
| Preface: About Arbortext Import Documentation | 9 |
| Guide to Documentation | 10 |
| Guide to This Document | 10 |
| Goals | 10 |
| A Quick Tutorial | 11 |
| Getting Started with Arbortext Import | 13 |
| Launch Arbortext Import Workbench | 14 |
| Creating a New Project Using Arbortext Import Workbench | 18 |
| Create a Transformation | 19 |
| Run a Transformation Object | 23 |
| Arbortext Import Workbench, Supported Transformations | 30 |
| Arbortext Import Workbench Interface | 31 |
| Overview | 32 |
| Transformations | 33 |
| Mapping Tab | 34 |
| Advanced Details Tab | 36 |
| File Menu | 39 |
| Transformation Menu | 40 |
| Tools Menu | 41 |
| Help Menu | 43 |
| Introduction to MapTemplates & MapObjects | 45 |
| Overview of MapTemplates | 46 |
| The Three Main Sections of MapTemplates | 46 |
| Setting up a New MapTemplate | 48 |
| Pre and Post-Processing Drivers | 52 |
| Pre and Post-Processing Drivers Interface | 53 |
| Overview of MapObjects | 56 |
| Main Functions of MapObjects | 56 |
| The Structure of a MapObject | 56 |
| Creating a new MapObject | 58 |
| MapObjects Details Pane | 67 |
| File Processing Scenarios for Arbortext Import Supported Transformations | 71 |
| Introduction | 72 |
| Importing a Microsoft Word 2003 (.doc) document into XML | 72 |
| Importing a Microsoft Word 2007 or 2010 (.docx) document into XML | 78 |
| Importing an HTML document into XML | 79 |

| | |
|--|------------|
| Importing a WordML document into XML | 79 |
| Transforming Word, FrameMaker, and HTML Documents into DocBook | 81 |
| Introduction | 82 |
| Overview of the DocBook MapTemplates..... | 82 |
| How the Templates Work..... | 83 |
| How to Customize the DocBook Template | 87 |
| Conclusion | 129 |
| Text Files | 131 |
| Parsing Text Files using Text Rules | 131 |
| How to Customize Configuration Files & Locations | 164 |



About This Guide

 **Note**

Although still a part of the Arbortext technology stack, this feature has been deprecated and is no longer supported by PTC.

The *Arbortext Import Tutorial* contains detailed, step-by-step tutorials for using Arbortext Import. The document contains several tutorials, which build in complexity. The tutorial guides you through the processes of building MapTemplates for parsing text files and building MapTemplates to parse Word and HTML documents. The processes particularly focus on using DocBook conversion templates. This document is available from the Arbortext Import Workbench **Help** menu and in the Arbortext Editor Help Center.

This tutorial is not meant to be a replacement for a conceptual overview of how Arbortext Import works. The *Arbortext Import Tutorial* is a companion to the *Arbortext Import Reference* available in the Arbortext Editor Help Center.

Prerequisite Knowledge

Creating MapTemplates requires experience working with document types generally, and with the specific document types that are the destinations of your MapTemplates.

Arbortext Editor and Arbortext Publishing Engine supporting documentation, along with the *Arbortext Import Tutorial* can be found in the Arbortext Editor Help Center, available as described in Documentation for PTC Products later in this section.

Technical Support

To contact PTC Technical Support, use the Contact Support and Customer Support Guide links on support.ptc.com.

The PTC Support pages also provide a search facility for you to browse for knowledge articles, best practices, and other information.

You must have a Service Contract Number (SCN) before you can receive technical support. If you do not have an SCN, contact PTC Technical Support or Customer Care Departments using the contact instructions found in your Customer Support Guide.

Documentation for PTC Products

You can access PTC product documentation using the following resources:

- Online Help

Click **Help** from the user interface for online help available for the product.

- Reference Documentation

PDFs of reference information are available from the Product Documentation area of support.ptc.com.

Select the Arbortext tab to access the Arbortext Reference Documentation link.

- Help Center

Help Centers for the most recent product releases are available from the Product Documentation area of support.ptc.com.

Select the Arbortext tab to access the Help Centers link.

You must have a Service Contract Number (SCN) before you can access the Arbortext Reference Documentation or Help Centers links. If you do not have an SCN, contact PTC Technical Support or Customer Care Departments using the contact instructions found in your Customer Support Guide.

Global Services

PTC Global Services delivers the highest quality, most efficient and most comprehensive deployments of the PTC Product Development System including Creo, Windchill, Arbortext, and PTC Mathcad. PTC's Implementation and Expansion solutions integrate the process consulting, technology implementation, education and value management activities customers need to be successful. Customers are led through Solution Design, Solution Development and Solution Deployment phases with the continuous driving objective of maximizing value from their investment.

Contact your PTC sales representative for more information on Global Services.

Comments

PTC welcomes your suggestions and comments on our documentation. You can submit your feedback to the following email address:

arbortext-documentation@ptc.com

Please include the following information in your email:

- Name
- Company
- Product
- Product Release
- Document or Online Help Topic Title
- Level of Expertise in the Product (Beginning, Intermediate, Advanced)
- Comments (including page numbers where applicable)

Documentation Conventions

This guide uses the following notational conventions:

- **Bold text** represents exact text that appears in the program's user interface. This includes items such as button text, menu selections, and dialog box elements. For example,
Click **OK** to begin the operation.
- A right arrow represents successive menu selections. For example,
Choose **File ▶ Print** to print the document.
- Monospaced text represents code, command names, file paths, or other text that you would type exactly as described. For example,
At the command line, type **version** to display version information.
- ***Italicized monospaced text*** represents variable text that you would type. For example,
installation-dir\custom\scripts
- *Italicized text* represents a reference to other published material. For example,
If you are new to the product, refer to the *Getting Started Guide* for basic interface information.

1

Preface: About Arbortext Import Documentation

| | |
|-----------------------------|----|
| Guide to Documentation..... | 10 |
| Guide to This Document..... | 10 |
| Goals | 10 |
| A Quick Tutorial | 11 |

Guide to Documentation

The following user manuals will teach you how Arbortext Import works and how best to customize it for your purposes:

- *Arbortext Import Tutorial* — This manual contains detailed, step-by-step tutorials for using Arbortext Import, creating MapTemplates to parse text files, parsing HTML and Word files, and also converting source documents into DocBook. This document is available from the Arbortext Import Workbench **Help** menu and also part of the Arbortext Editor Help Center.
- *Arbortext Import Reference* — This manual contains explanatory material about Arbortext Import - including all of the key concepts, such as MapTemplates, MapObjects, Text Parsing Rules, XML Parsing Rules, and ppXML. This document is available from the Arbortext Import Workbench **Help** menu and also part of the Arbortext Editor Help Center.

Guide to This Document

This document is meant to be a step-by-step tutorial on how to use Arbortext Import to convert files into meaningful XML. This document contains several tutorials that build in complexity.

This tutorial is not meant to be a replacement for a conceptual overview of how Arbortext Import works, nor is it meant to be a reference manual. Refer to the *Arbortext Import Reference* and Arbortext Import Workbench online help for conceptual and referential information.

Goals

The goals of this tutorial are twofold. Specifically we want to:

- Provide you with a quick tutorial so you can get Arbortext Import working on your system immediately
- Familiarize you with the different aspects of Arbortext Import, including:
 - Creating and running sample transformations and MapObjects
 - Creating and modifying MapTemplates
 - Gaining an in-depth understanding of ppXML
 - Becoming familiar with different file processing scenarios
 - Understanding the debugging environment
 - Understanding the terminology of Arbortext Import.

If you want to learn to use Arbortext Import, it is best to go through this entire tutorial step-by-step and recreate these templates so that you can have a better understanding of how to best use Arbortext Import.

A Quick Tutorial

The best way for you to learn about Arbortext Import is to start using it gradually. In that vein, this tutorial has three parts.

- First, it will guide you through the quick process of transforming a Microsoft Word document into a simple, stylistic-based XML that we call *pre-processed XML*, or ppXML. We refer to the process of conversion from one format to another as a *transformation*.
- Second, to illustrate different types of documents into ppXML, the tutorial will show you how to transform simple HTML and Framemaker MIF documents into ppXML, illustrating how similar it is to transform different types of documents into ppXML.
- Third, this tutorial will show you how to transform an ASCII text file into simple, yet semantic-based XML. This lesson will serve to expose you to the main portions of Arbortext Import, MapTemplates, and MapObjects.
- Finally, this tutorial will guide you through the process of creating and modifying MapTemplates and MapObjects.

 **Note**

*Many of the end results of the step-by-step tutorials are available in the examples project that is shipped with Arbortext Import. To see the example project, choose **File ▶ Open ▶ Project** and select **Examples .xyz** in the **\Examples** subdirectory. (The process of how to open the example project is further described in [Loading the Example Project on page 18.](#))*

2

Getting Started with Arbortext Import

| | |
|--|----|
| Launch Arbortext Import Workbench | 14 |
| Creating a New Project Using Arbortext Import Workbench..... | 18 |
| Create a Transformation..... | 19 |
| Run a Transformation Object | 23 |
| Arbortext Import Workbench, Supported Transformations | 30 |

Launch Arbortext Import Workbench

To Launch Arbortext Import Workbench perform the following steps:

Update Arbortext Import Workbench Configuration

1. Choose **File ► Import ► Import Workbench Configuration** in Arbortext Editor.

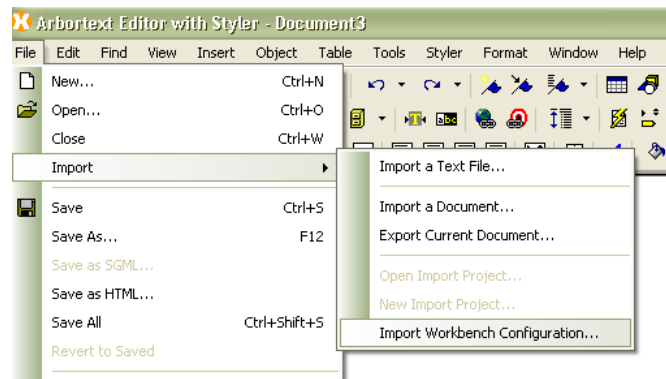


Figure - 2.1

2. The **Import Workbench Configuration** window will open. This window shows the path for home and the configuration directory. Here you will specify the path for the **Repository Directory**. It is the directory where import related data is updated, that is, the template directory (STDTemplates), TargetSchemas, and XSLT.

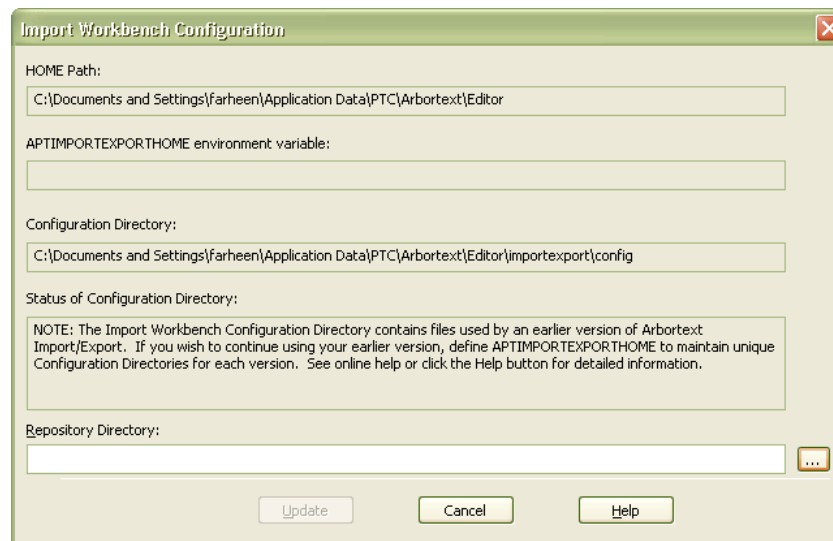


Figure - 2.2

- The repository path can be set either by manually giving the path in **Repository Directory:**, or by clicking browse (...), to specify the location.

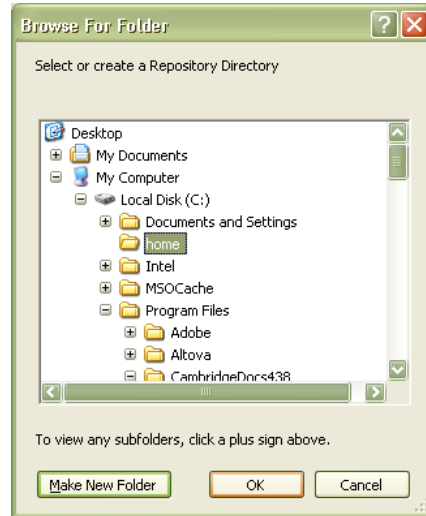


Figure - 2.3

- The generic repository used in this tutorial is **<home>**.

3. Once the path is set click **Update**.

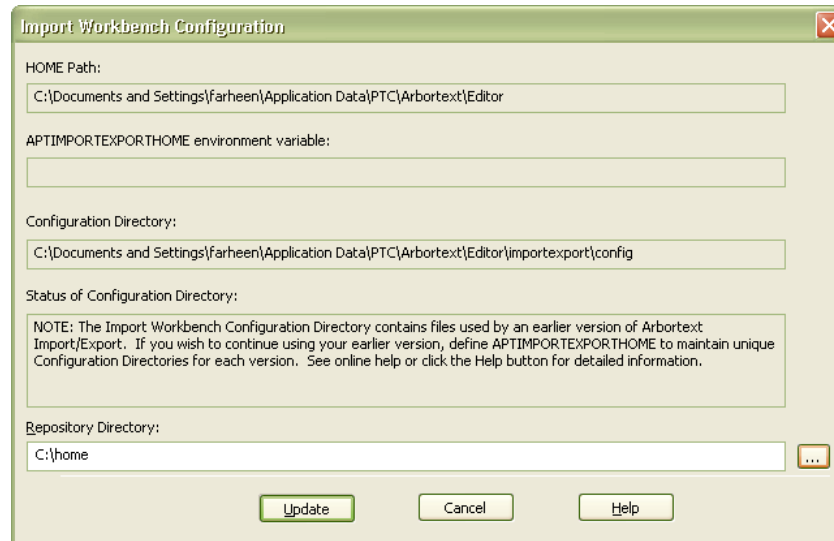


Figure - 2.4

4. When you click **Update**, the application will prompt you for confirmation updates. Click **Yes** to continue the configuration update procedure.

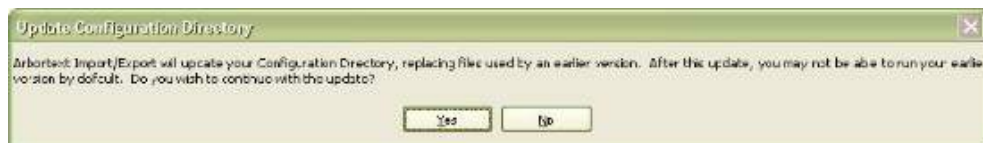


Figure - 2.5

5. The application will ask for further confirmation updates as in Figure 2.6. Click **Yes** again to continue.



Figure - 2.6

6. This will result in enabling the **New Import Project** and **Open Import Project** options in the Arbortext Editor **File** menu.

Creating New Import Project

1. Once the workbench configuration is successfully updated. In Arbortext Editor, choose **File ► Import ► New Import Project**.

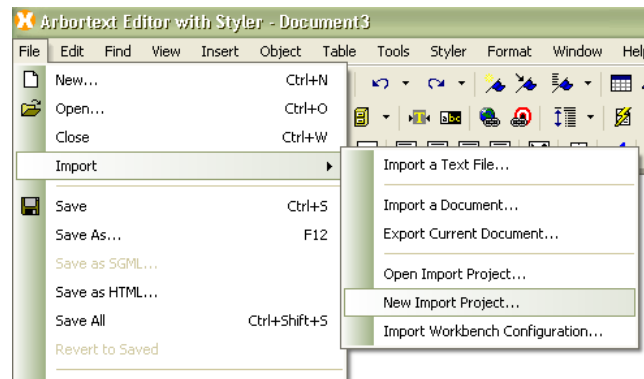


Figure - 2.7

2. The **Browse For Folder** window will open. To create a new import project, first specify the location where you want to save the new project. For this purpose:
 - a. Create a new folder.
 - b. Enter a name for the newly created folder.

c. Click **OK**.

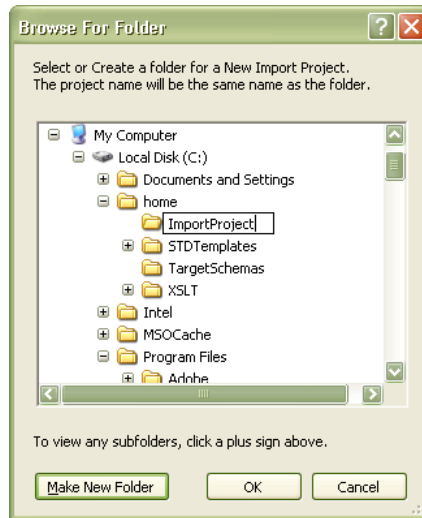


Figure - 2.8

3. As a result, an empty **ImportProject.xyz** opens in the **Arbortext Import Workbench**. You can also open existing projects using the **Arbortext Import Workbench** by selecting **File ► Import ► Open Import Project**.

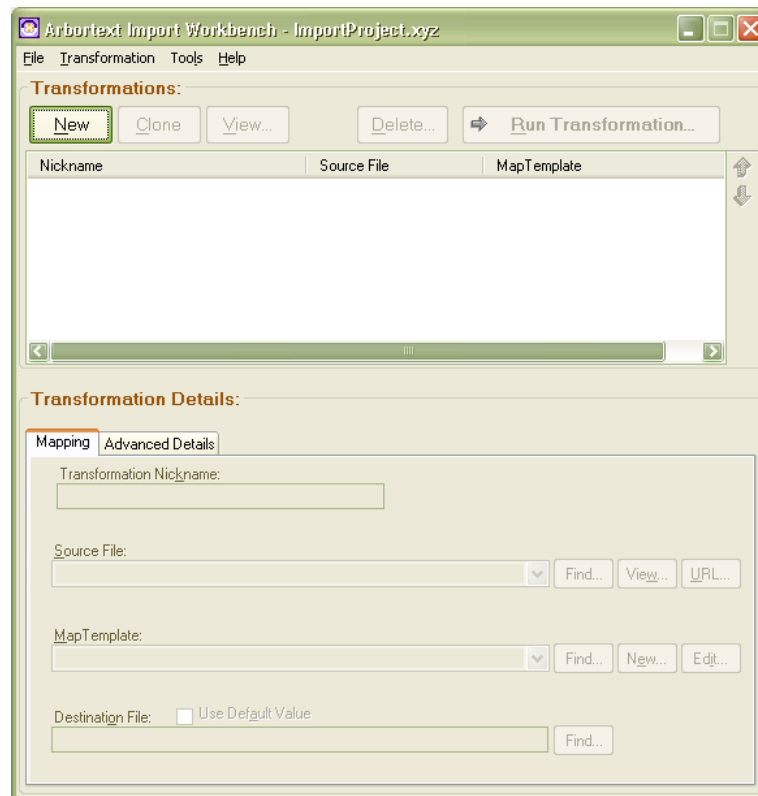


Figure - 2.9

Loading the Example Project

An example project that can be loaded from the **Arbortext Import Workbench** is shipped with Arbortext Import. To open the example project:

1. Select **File ► Open ► Project** from the **Arbortext Import Workbench**. The **Select Project** window opens.
2. Specify the location where the example project is located: **Arbortext-path \lib\cpix\data\MyProjects\Examples**.
3. Once **Examples.xyz** is specified click **Open**. The example project will replace the already-opened project in the **Arbortext Import Workbench**.

Creating a New Project Using Arbortext Import Workbench

You can also create a new project using Arbortext Import Workbench.

1. On the main window of Arbortext Import Workbench, click the **File ► New ► Project**. The **New Project** dialog box is displayed as shown in Figure 2.10.

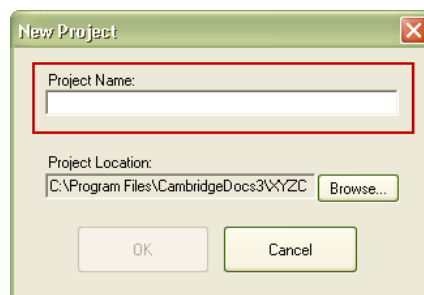


Figure - 2.10

2. In **Project Name**, highlighted in Figure 2.10, type the name of the project. You should keep the project name short. A subdirectory will be created with the name of the project, for example, “MyTestProject”.
3. The **Project Location** specifies the default location from where the new project can be accessed. To change this location, click **Browse** and navigate to another location.
4. Click **OK** to close the dialog box. An empty Arbortext Import Workbench is displayed.

The list of transformations (if any) appears under the **Transformations** area and the details of the transformations appear under **Transformation Details**. Each transformation is a specific command to Arbortext Import to transform a given source file (or set of source files indicated by wildcards, such as ***.doc**), using a specified MapTemplate and options.

MapTemplates, which this tutorial will cover extensively later, contain the rules Arbortext Import Workbench will use to transform existing content into XML.

Each time a file is referenced in a transformation, it is added to the list of the sample files included in the project. MapTemplates and sample files are available in **Source File:** and **MapTemplate:** under the **Transformation Details** pane of the Arbortext Import Workbench.

 **Note**

*In the Arbortext Import Workbench, you can always select **Manage Project Data** from the **File** menu to see a list of sample files and MapTemplates that are included in the current project.*

Create a Transformation

Now you are ready to create a transformation.

1. At the top of the Arbortext Import Workbench, you will see a **New** button just under **Transformations**. Click this button to display the **New Transformation Wizard** as shown in Figure 2.11.

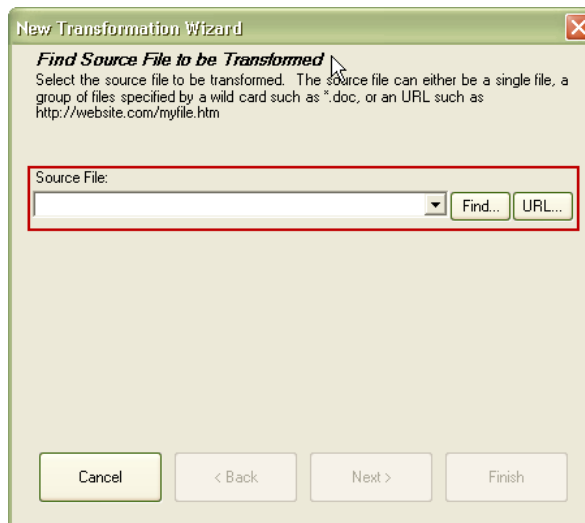



Figure - 2.11

 **Note**

*Whether this wizard runs each time you click **New** is determined by the setting **Use New Transformation Wizard** on the **Tools** menu in the Arbortext Import Workbench.*

2. The wizard will prompt you for a source file as highlighted in Figure 2.11. The **Source File** is the initial document that is to be transformed.

- The **Source File** list shows any sample files that have been added to the project to date.
- **Find** enables you to browse your hard drive for a source file. The sample files are located at: **Arbortext-path\samples\importexport**.

Click **Find**  to browse to the subdirectories and choose a Word sample file, named such as, **\word\articlemeaningful.xml.doc**.

- **URL** is used to open the **URL** window as shown in Figure 2.12. Here you can enter any URL of a source file to be transformed
- Click **OK** to continue.

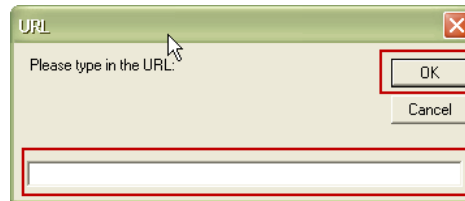


Figure - 2.12

3. The sample source file will be included in the **Source File**. Click **Next** to continue with the transformation process.
4. The next step is to choose a MapTemplate for your transformation. Select the **Custom / Other** option for now, as shown in Figure 2.13.
5. Click **Next**.

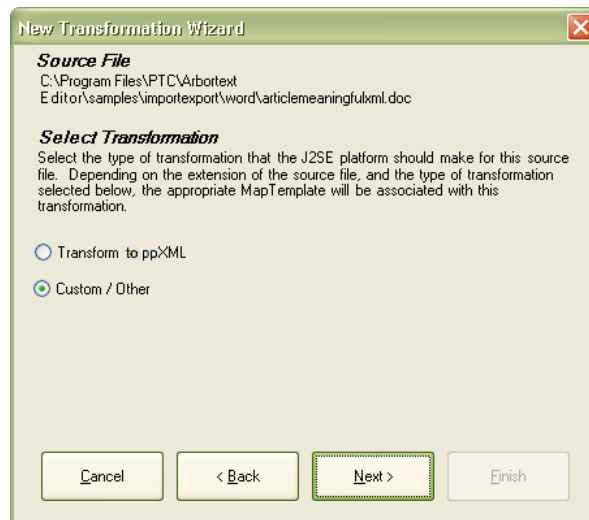


Figure - 2.13

6. The **Select to Direct Transformation** window is displayed as shown in Figure 2.14. Note that all the standard templates are stored in the `\Common` subdirectory of the `<home>\STDTemplates` directory.
 - **Find** will automatically take you to this directory.
 - Now choose your standard template for transforming a Word document into ppXML. Choose **Word2XMLJ.std**; which specifies to use the Java Word driver.
 - If you click **New...**, the **New MapTemplate Wizard** will open. This wizard can be used to create a new MapTemplate. Details of creating a new MapTemplate using the MapTemplate wizard are given in [Setting up a New MapTemplate on page 48](#).

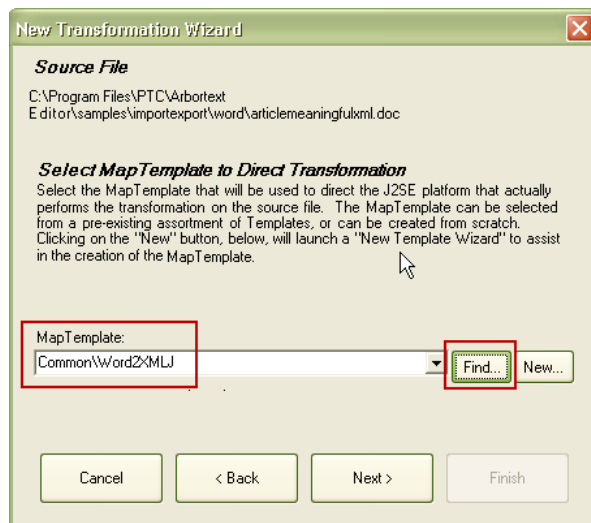


Figure - 2.14

Extensive use of the MapTemplate Editor will be further explained later in the tutorial.

 **Note**

The selection of the MapTemplate is directly dependent on the format of the source file. For example, if the source file is a Word file, then a FrameMaker transformation like MIF2XMLJ cannot be run on the file.

7. Click **Next** to display the **Provide Transformation Nickname** window.
8. The **Provide Transformation Nickname** window will prompt you to enter an identifying **Transformation Nickname**. For example,

SampleWordTransformation or something similar as shown in Figure 2.15. This will help remind you of the nature of this transformation.

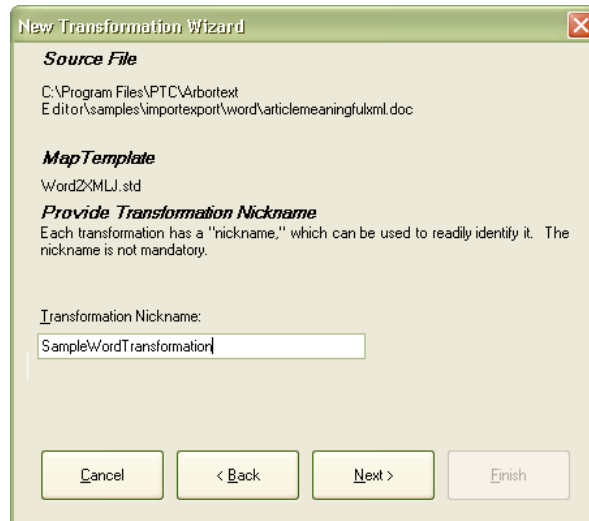


Figure - 2.15

9. Click **Next** to continue.

10. At this point, you are presented with three options as highlighted in Figure 2.16. You can select any of these options:

- **Open/Edit MapTemplate** — Opens the MapTemplate Editor to further edit the selected template.
- **Run Transformation When Complete**— Immediately runs the transformation using the preferences you just entered.
- **Do Not Run Transformation When Complete**— Does not run the transformation, but will put the transformation, along with all your settings into the transformation list on the main **Arbortext Import Workbench**.

For now select **Do NOT Run Transformation When Complete**.

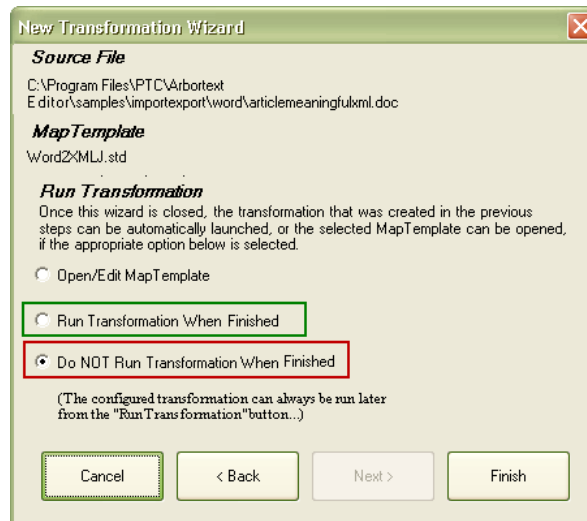


Figure - 2.16

11. Click **Finish** to finish the transformation wizard.
12. The new transformation created will be added and displayed on the transformation list of the project in the **Arbortext Import Workbench**. To save the created transformation and the current project, select **File ► Save Project** on the **Arbortext Import Workbench**. The details of all the menus are given in [3 Arbortext Import Workbench Interface on page 31](#).

Optionally, you can repeat this process to add more transformations for the following sample files using the specified built-in templates:

- **Arbortext-path\samples\importexport\html\articleWhyConvert.htm**, using the template **Common\HTML2XMLJ.std**.
- **Arbortext-path\samples\importexport\frame\sampldocument.mif**, using the template **Common\MIF2XML.std**; which is the Framemaker MIF driver.

Run a Transformation Object

Now, run the first transformation to transform a Word document into a simple XML document using ppXML. (ppXML is explained in the *Introduction to Pre Processing*

XML (ppXML) section of the *Arbortext Import Reference*). The Arbortext Import Workbench is shown in Figure 2.17.

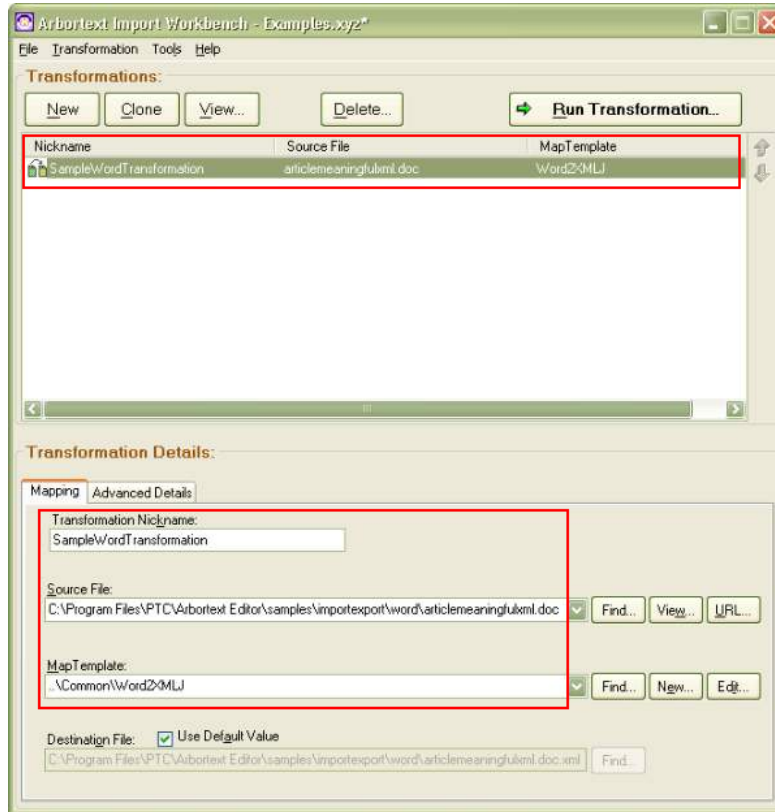


Figure - 2.17

To run the transformation process, click **Run Transformation** on the main Arbortext Import Workbench. This window contains important information that helps explain how a transformation took place, and what modifications may be needed to correct the transformation.

You will likely spend a lot of time using this **Transformation** window during the course of your document conversions, and the fine-grained ability to debug a transformation is a key part of the Arbortext Import methodology.

Console Output Display

The **Transformation** window shown in Figure 2.18 highlights the **Console Output Display** tab, which displays the output of the transformation process. This tab has a console that shows a series of output messages that give the status of the process. It also

highlights any errors and warnings that might come out during the transformation - including errors in the validation of the resulting XML.

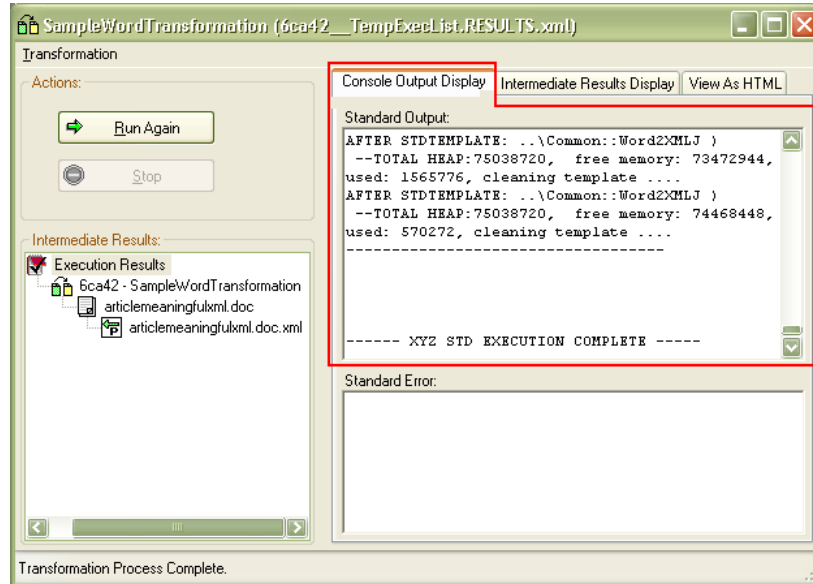


Figure - 2.18

Intermediate Results tree

The **Intermediate Results** tree on the left-hand side of the **Transformation** window as highlighted in Figure 2.19, shows a list of all the files that were generated by running the Transformation Object.

These files include the *original source file*, the *final destination file*, and *intermediate files* that were produced such as debug files. In this case, the Transformation Object has only

one source file, and there were no intermediate files generated. For more on intermediate files, see the subsequent chapters that describe MapTemplate pre & post-processing.

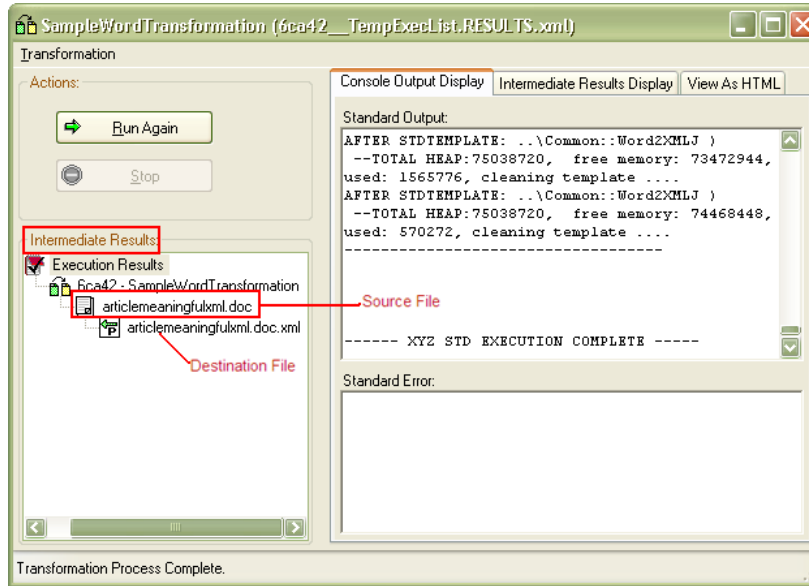


Figure - 2.19

Intermediate Results Display

For each file generated, the **Intermediate Results Display** tab as highlighted in the **Transformation** window in Figure 2.20, can be used to quickly look at the file and see the hierarchical nature of the relevant XML that was produced. This type of a view makes it easy to debug how the transformation was done. Notice that this view includes both attributes and text children of output XML.

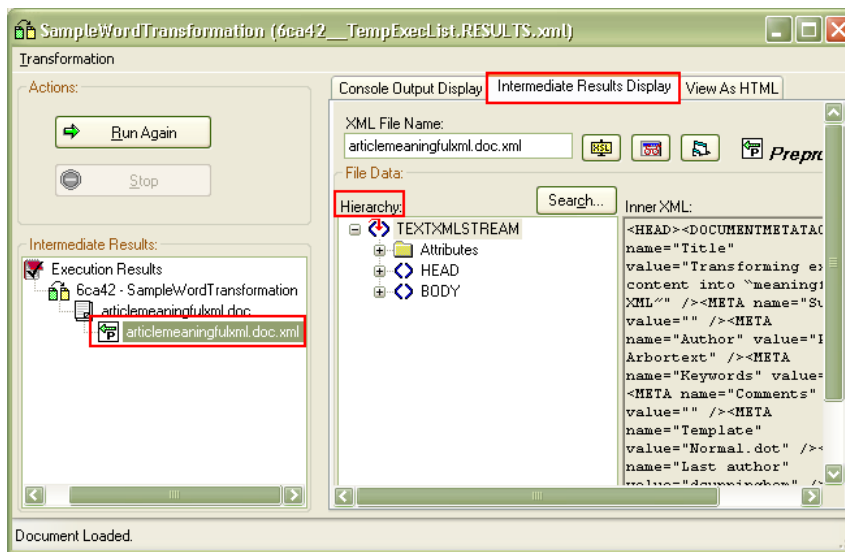





Figure - 2.20

Several buttons on the top of the **Intermediate Results Display** tab make it easy for you to further inspect the resulting XML.

- **View**  launches your default XML viewer, typically Arbortext Editor.
- **XSL**  allows you to view the XSLT applied to the XML. XSLTs are used for transforming an XML file into various formats for publishing, including HTML or another XML document. For more on applying XSLT as part of the transformation, see the *Pre and Post Processor Drivers* section of the *Arbortext Import Reference*.

For files that are in ppXML format, as this particular XML is, **ppRun.xsl** is a provided XSLT found in the `<home>\XYZConfig\XSLT\ppXMLtoHTML`. For more on ppXML and the XSL, see the *Introduction to Pre Processing XML (ppXML)* section of the *Arbortext Import Reference*.

- **Detach**  is used to view the ppXML file displayed within the **Intermediate Results Display** tab in a separate window. This feature can be useful when you have multiple intermediate files, and you want to make a visual, side-by-side comparison of the intermediate files.

Search

With a ppXML node selected in the **Intermediate Results** tree in the **Transformation** window, click **Search** in the **Intermediate Results Display** tab to open a separate **Search** window as shown in Figure 2.21. This window allows you to search you XML in different ways, including using an XPath expression.

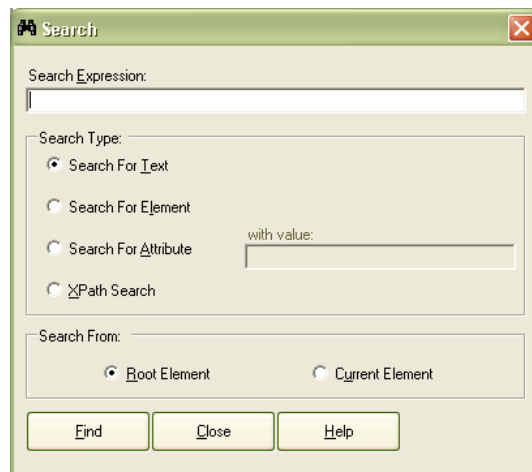


Figure - 2.21

The **Search** dialog box is displayed from a variety of windows in Arbortext Import whenever you click **Search**. The dialog box provides the means to search the document ppXML tree views shown in the MapTemplate Editor and the transformation results window.

In most cases, you search through ppXML markup and text, but in the transformation results window you can search through your final XML results.

When a match is found, it is selected and displayed in the corresponding document tree view. Following are options available on the **Search** window:


- **Search Expression** — Type the text, element name, attribute name or XPath expression to be used in the search (depending on **Search Type** option selected). All search expressions are case sensitive.
- **Search Type** — This group of options determines how the **Search Expression** will be interpreted:
 - **Search For Text** — Search for a text string that matches the **Search Expression** value. Only single text nodes are examined. Text that spans descendent nodes is not recognized as a match.
 - **Search For Element** — Search for an element that matches the **Search Expression** value.
 - **Search For Attribute** — Search for an attribute whose name matches the **Search Expression** value. To search for an attribute with a specific value, type the desired value in **with value** next to the option, otherwise any attribute will be selected with matching names.
 - **XPath Search** — Search using an XPath expression. With XPath expressions you can search for text, elements, and attributes in a wide variety of methods. One way to search for text using XPath is the expression:

```
//*[contains(text(),' search text')],
```

where *search text* is the search string.
- **Search from** — This group allows you to specify the starting point for the search process.
 - **Root Element** — Click to begin the search at the root element at the top of the document tree.
 - **Current Element** — Click to begin the search at the currently selected node in the document tree.
- **Find** — Click to begin your search. When **Current Element** is selected, click **Find** to find subsequent matches in the document tree. Click this button again to search forward from the last found node in the document tree.
- **Close** — Click to close this dialog box.
- **Help** — Click to display the help topic for this dialog box.

View As HTML

You can use the **View As HTML** tab to view the transformed file. You'll see that the **ppRun.xsl** file is already there in the **XSL File**, as highlighted in the **Transformation** window shown in Figure 2.22.

Using **Browse** you can select any other required XSL file. You can click **Reload**  **Reload** to view the ppXML transformed into HTML in the embedded browser window. This enables you to quickly review your transformed document to see the kind of XML you will get, and verify the transformed document to ensure it has all the information you'll need for further processing.

To locate a particular piece of text in the ppXML file:

1. You can select any text from the HTML file loaded in the embedded browser
2. Click the **Locate Selected Text in XML**
3. Select the **Intermediate Result Display** tab and you will see that the selected text will be highlighted in the ppXML file in the **Hierarchy Tree**.

You can click **Clear** to clear any ppXML HTML view loaded in the embedded browser.

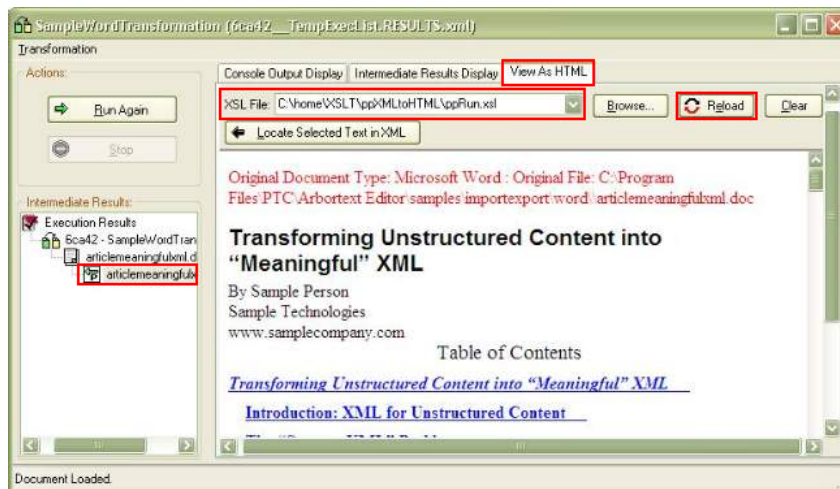


Figure - 2.22

Notice that in this first example, the Word document is transformed only into a simple XML format. You did not extract any meaningful information or structure from the ppXML. To see a more complete example, you might want to transform ppXML into a real, semantic-based XML format such as DocBook, XML, Legal XML, or any custom DTD or schema that you have. Arbortext Import comes with MapTemplates that you can customize for your source content to transform documents into DocBook.

MapTemplatesFor more on the DocBook , see [Overview of the DocBook MapTemplates on page 82](#).

For more on using Arbortext Import Workbench to transform into your own custom DTD, refer to the *MapTemplate Editor*, *Working with MapObjects*, and *Text Parsing Rules* sections of the *Arbortext Import Reference*.

Arbortext Import Workbench, Supported Transformations

Arbortext Import supports the following transformations:

- Transform to DocBook XML
- Word Document Transformations
- Docx (Word 2007 and 2010) Transformations
- Word ML Document Transformations
- HTML Document Transformations
- Text File Transformations
- Transform FrameMaker to XML



Note

For a complete list of supported formats, including post-release updates, refer to “Word Processing Compatibility for Import feature of Arbortext Import/Export” in support.ptc.com.

3

Arbortext Import Workbench Interface

| | |
|----------------------------|----|
| Overview | 32 |
| Transformations | 33 |
| Mapping Tab | 34 |
| Advanced Details Tab | 36 |
| File Menu | 39 |
| Transformation Menu | 40 |
| Tools Menu | 41 |
| Help Menu | 43 |

This section describes the Arbortext Import Workbench and its capabilities.

Overview

The first screen you see when you launch Arbortext Import, as explained in [Launch Arbortext Import Workbench on page 14](#), is the Arbortext Import Workbench main window. The Arbortext Import Workbench is the window where you can create and manage projects when a project consists of one or more individual transformations.

Each transformation in a project uses the following files:

- **Source File** – A sample source file that is being converted. Each source file in the project is considered a sample source file that can be used in the MapTemplate Editor.
- **MapTemplate** – The MapTemplate that will be used for converting the source content.
- **Destination File** – The output file created by the transformation. Often you will use the default value in this field, which is just the name of the source file with an additional extension. For example, the default destination file for **mydoc.doc** is **mydoc.doc.xml**.

Each transformation can be saved, debugged, and run again later.

A project is a way to group together one or more transformations. You should create your own project for your own transformations.

Transformations

Arbortext Import Workbench shows you the list of transformations that are in this particular project as shown in Figure 3.1.

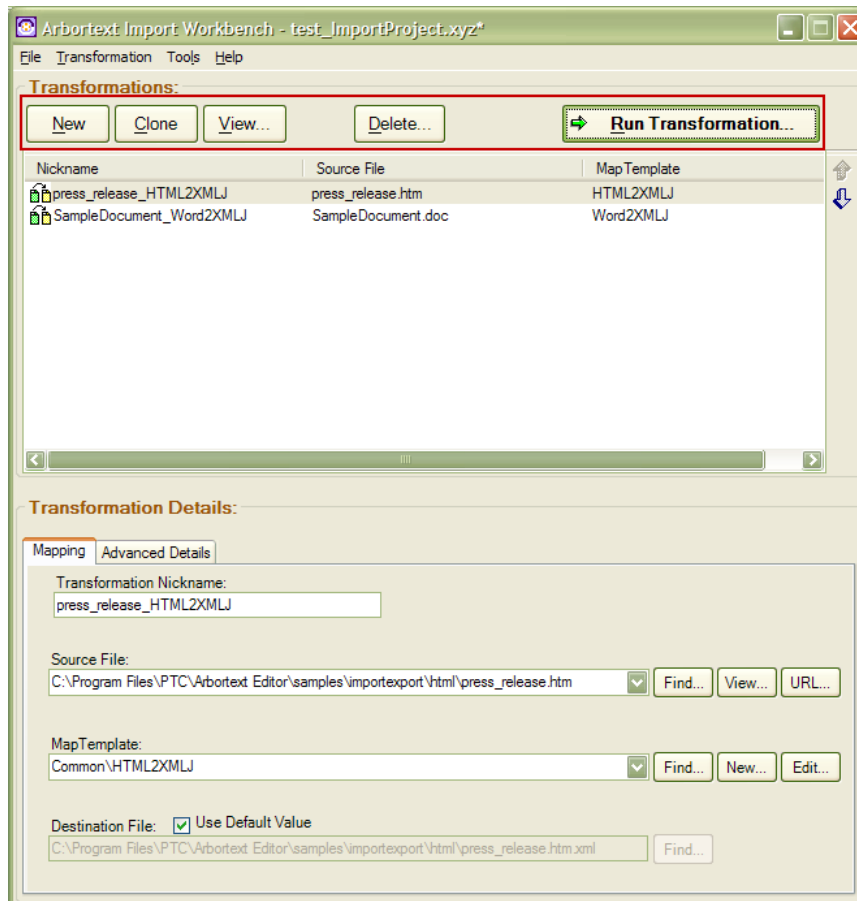


Figure - 3.1

You can manipulate the transformations with the following buttons, as highlighted in Figure 3.1.

- **New** – Creates a new transformation. If the menu item **Use New Transformation Wizard** under the **Tools** menu is checked, this button launches the **New Transformation Wizard**. The **New Transformation Wizard** will guide you through the selection of a source file, the selection or creation of a Map Templates, and the selection of a transformation nickname. If the menu item **Use New Transformation Wizard** under the **Tools** menu is unchecked, you will have to fill this information out directly in the **Mapping** tab for the newly created transformation.
- **Clone** – Clones the currently selected transformation. The cloned transformation is identical to the original, with the text **Clone of** being added to the start of the transformation.

Note that any overridden options are also cloned.

- **View** – Opens the last run of the selected transformation in the **Transformation** window. If the transformation has never been run, then you will see an error dialog box when Arbortext Import tries to locate the results manifest file.
- **Delete** – Deletes the selected transformation. A warning dialog box appears before you are allowed to make the deletion.
- **Run Transformation** – Launches the selected transformation in the **Transformation** window. Note that you can also launch the selected transformation by double-clicking it.

Mapping Tab

The **Mapping** tab under the **Transformation Details** pane, as highlighted in Figure 3.2, encloses the common options that must be specified for each and every transformation.

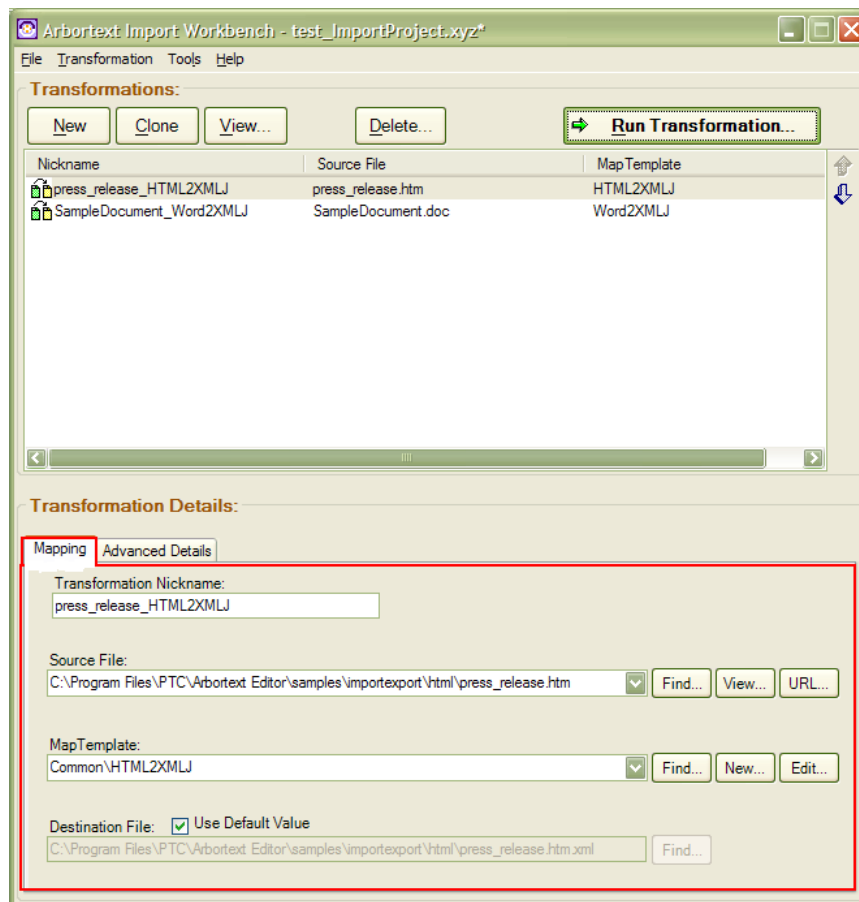


Figure - 3.2

The following options are available on the **Mapping** tab:

- **Transformation Nickname** — Edit the name of the selected transformation here.
- **Source File** — The source file that will be transformed. Note that multiple files in the same directory can be entered using wildcard syntax, such as ***.doc**, ***.xml**.
 - **Find** — Enables you to navigate to the source file on your local system.
 - **View** — Opens the source file using the application associated with its extension.
 - **URL** — Enables you to type in a URL to download a file if the file resides on the network server. Note that you must specify a destination file if you use a URL-based source file. You cannot simply use the default destination provided by the Arbortext Import Workbench main window.
- **MapTemplate** — The MapTemplate that will be used to direct the transformation.

The MapTemplate is specified by a package and a MapTemplate name, such as **Common\Word2XMLJ**. The package is the directory where the MapTemplate is stored, relative to the base MapTemplate directory, **<home>\STDTemplates**. The MapTemplate name is just the file name without the ***.std** extension.

 - **Find** — Enables you to navigate to and select another MapTemplate on your local system.
 - **New** — Creates a new MapTemplate. If the menu item **Use New MapTemplate Wizard** under the **Tools** option is selected, this button launches the **Use New MapTemplate Wizard**. The **Use New MapTemplate Wizard** will guide you through the creation of a MapTemplate, including its name, the type of source content it operates against, whether or not it should validate against a DTD or schema, and so on. If the menu item **Use New MapTemplate Wizard** under the **Tools** menu is unchecked, you will have to fill this information out directly in the MapTemplate Editor.
 - **Edit** — Launches the MapTemplate Editor for the particular MapTemplate enabling you to modify the MapTemplate.
- **Destination File** — The file name of the converted file after the transformation process. This has a **Use Default Value** checkbox next to it. If there are multiple files in the source document (for example, ***.doc**), you can use the default file name for outputs or you can specify a pattern for the output file name using the following special items:
 - "{Input}" — The full input file name and path.
 - "{Output}" — The full output file name and path (this will be a generated name for each step of the transformation).
 - "{InputDirectory}" — The directory of the source file.
 - "{OutputDirectory}" — The directory of the destination file.
 - "{InputFileName}" — The input file name only, with no path information (including extension).

-
- "{OutputFileName}" — The output file name only with no path information (including extension).
 - "{InputFileNameNoExt}" — The input file name only with no path and no extension.
 - "{OutputFileNameNoExt}" — The output file name only with no path and no extension.

Find enables you to navigate to a particular file that will act as the destination location and name. Note that the selected file will be replaced by the output of the selected transformation.

Advanced Details Tab

The **Advanced Details** tab enables you to specify additional options for each single transformation. The options on this tab are:

- **Run Preprocessing Driver** — (DEFAULT: True) This checkbox determines if the preprocessing drivers for the current MapTemplate should run. This is useful if you are working with a large document and are constantly re-running the transformation during development. Setting this checkbox to “unchecked” means that the Preprocessing Drivers will not run each time; they will only run when necessary. These options are shown in Figure 3.3.
- **override** — This button brings up the **Edit Override Options** dialog box. Use this dialog box to override the options that are by default defined for the Preprocessing Driver in the MapTemplate. The override options apply only for the selected transformation.

This feature enables you to modify different preprocessor options for a given MapTemplate without changing any of the options in the MapTemplate itself. Doing so can be very useful if you are trying out a previously developed MapTemplate on a different source file, and you want to experimentally change its preprocessor options. If the experiment proves successful, and you want to

change the options for every single run of this MapTemplate, then you can make the necessary changes inside the MapTemplate.

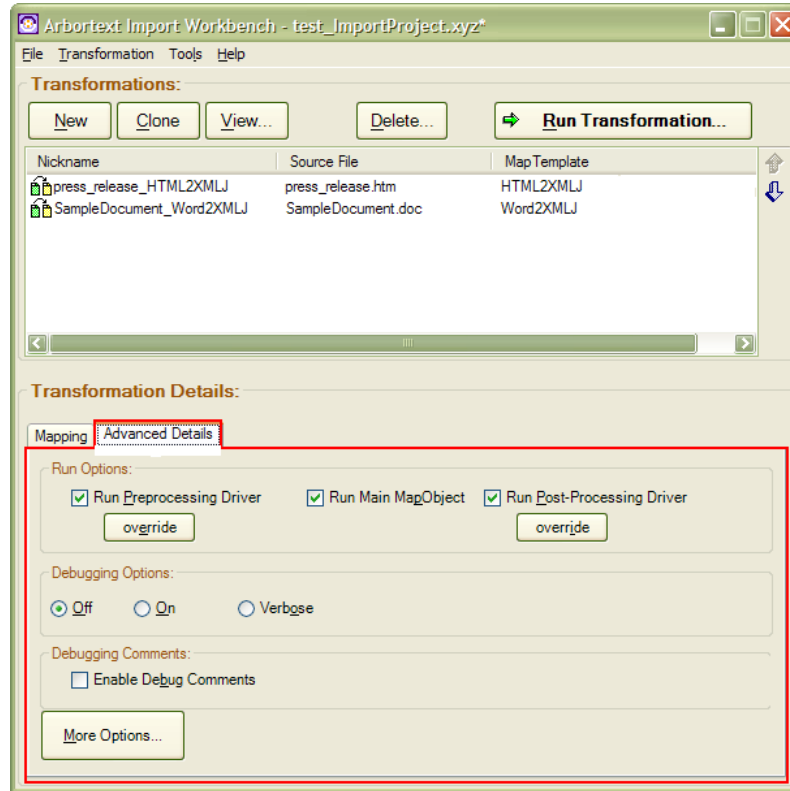


Figure - 3.3

- **Run Main MapObject** — Lets you turn off MapObject processing for a specific run of a MapTemplate.
- **Run Post-Processing Driver** — Lets you turn off Post-Processing Drivers for a specific transformation with a MapTemplate, similar to the way you can turn off Preprocessing Drivers as discussed above.
 - **override** — Lets you override specific Post-Processing options for the first Post-Processing Driver in a MapTemplate, again for a single run of the MapTemplate.
- **Debugging Options** — Turn on or off debugging logs for a given transformation. Debugging logs can be quite large and should only be used when developing and debugging a MapTemplate. Choosing **Verbose** option helps to record more information than the information recorded in the usual debugging mode.
- **Enable Debug Comments** — Enables and disables comments during debugging. These comments appear in MapTemplates.

- **More Options** — Displays the **Advanced Transformation Options** dialog box, as shown in Figure 3.4, which has several more options:

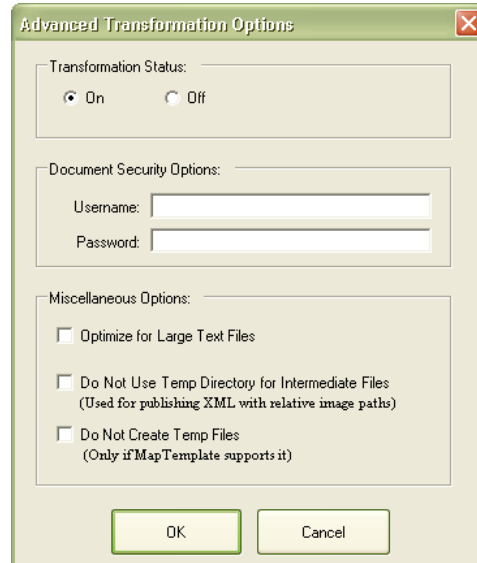


Figure - 3.4

- **Transformation Status** — Turns off processing of a single transformation. This option is typically useful only in projects where you are running multiple transformations (like when you select **Run All** from the **Transformation** menu on the Arbortext Import Workbench main window), or when you are using the Arbortext Import API.
- **Document Security Options** — If you have a document that requires a user name or password to open or access the document.
- **Miscellaneous Options** — These options are useful in certain, very specialized circumstances like:
 - **Optimize for Large Text Files** — Use only when parsing very large text files and not for processing documents (Word, FrameMaker, etc.). In this case Arbortext Import uses a more sophisticated string handling approach internally.
 - **Do Not Use Temp Directory for Intermediate Files** — Rather than having the intermediate XML files that are generated in a given transformation appear in a temporary file under the project directory, **Arbortext-pathlib\cpix\data\MyProjects\Examples**, you can specify the temporary files location which can be stored in the same location as that of the source file. This option is typically only used when publishing from XML that has relative image paths, and not when converting a document to XML.
 - **Do Not Create Temp Files** — Under normal operation, Arbortext Import creates temporary files during each step in a multi-step transformation. With

this option selected, however, the temporary files are not created, and instead are passed in-memory.

File Menu

Following are descriptions of the Arbortext Import Workbench **File** menu choices:

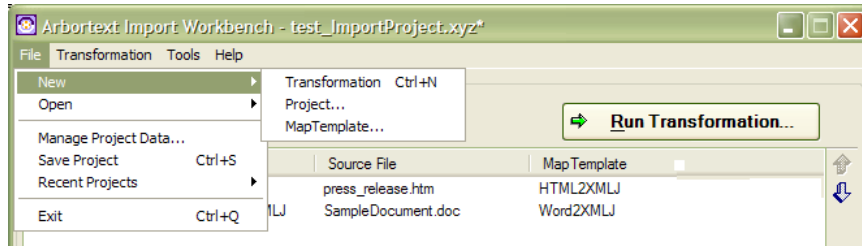


Figure - 3.5

- **New**
 - **Transformation** — Creates a new transformation. If **Use New Transformation Wizard** on the **Tools** menu is checked, this transformation launches the **New Transformation Wizard**. The **New Transformation Wizard** guides you through the selection of a source file, the selection or creation of a MapTemplate, and the selection of a transformation nickname. If **Use New Transformation Wizard** on the **Tools** menu is unchecked, you will have to fill this information out directly in the **Mapping** tab for the newly-created transformation.
 - **Project** — Displays the **New Project** dialog box, which prompts you for the name of the project. Each project is stored in a subdirectory of this directory, and each project has a manifest file called *projectname.xyz*.
 - **MapTemplate** — Creates a new MapTemplate. If **Use New MapTemplate Wizard** on the **Tools** menu is checked, this menu item launches the **New MapTemplate Wizard**. The **New MapTemplate Wizard** guides you through the creation of a MapTemplate, including creating the template name, identifying the type of source content the MapTemplate will process, the document type against which the resulting XML will be validated, and so on. If **Use New MapTemplate Wizard** on the **Tools** menu is unchecked, you will have to fill this information out directly in the MapTemplate Editor.

The new MapTemplate is automatically added to the project, and becomes available in various lists (such as the **Mapping** tab, in the **New Transformation Wizard**, and so on).

- **Open**
 - **Source File** — Opens the selected source file, using the application that is associated with the source file extension. The opened file is automatically added to the current project, making it available from various lists (such as the **Mapping** Tab, in the **New Transformation Wizard**, and so on).

- **MapTemplate** — Opens the selected MapTemplate, using the MapTemplate Editor. The opened MapTemplate is automatically added to the current project, making it available from various lists (such as the Mapping Tab, in the **New Transformation Wizard**, and so on).
- **Project** — Opens an Arbortext Import project.
- **Manage Project Data** — Opens the **Manage Project Data** dialog box. This dialog box enables you to add MapTemplates to and delete MapTemplates from the current project. These MapTemplates and source files are then available in the various lists (such as the **Mapping** tab, in the **New Transformation Wizard**, and so on). This dialog box also enables you to add more than one source file and more than one MapTemplate at a time.
- **Save Project** — Saves the currently opened project.
- **Recent Projects** — Lists recently saved projects. A given project can be opened by selecting it.
- **Exit** — Closes Arbortext Import Workbench.

Transformation Menu

The following options are available on the **Transformation** menu:

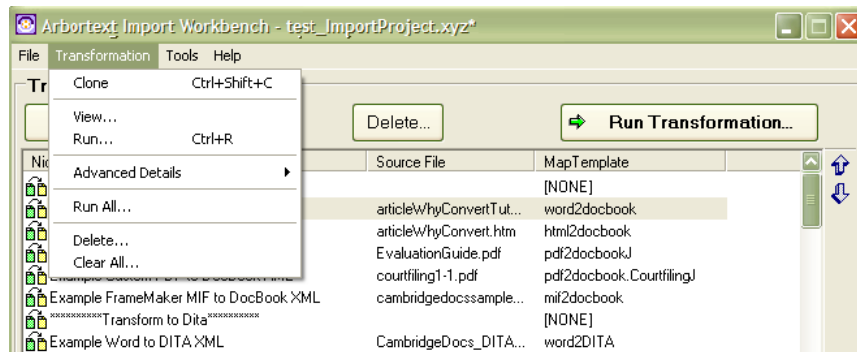


Figure - 3.6

- **Clone** — Clones the currently selected transformation. The cloned transformation is identical to the original, except that its transformation has the text **Clone of** in the start of the transformation nickname.

Note that any overridden options are also cloned.

- **View** — Opens the last executed transformation in the transformation window. If the transformation has never been run, then you will see an error dialog box as Arbortext Import tries to locate the results manifest file.
- **Run** — Launches the selected transformation in the **Transformation** window.
- **Advanced Details:**

- **Override Preprocessing Driver Settings** — Opens the **Edit Override Options** dialog box to override the options that are defined for the first **Preprocessing Driver** in the MapTemplate. The override options only apply to the selected transformation.
- **Override Post-Processing Driver Settings** — Opens the **Edit Override Options** dialog box to override the options that are defined for the first **Post-Processing Driver** in the MapTemplate. The override options only apply to the selected transformation.
- **More Options** — Opens the **Advanced Transformation Options** dialog box to let you modify various options for the selected transformation. For more information see [Advanced Details Tab on page 36](#).
- **Run All** — Runs all the transformations in the currently open project.
- **Delete** — Deletes the currently selected transformation. A warning dialog box appears before allowing you to make the deletion.
- **Clear All** — Deletes all the transformations in the project. A warning dialog box appears before you are allowed to make all the deletions.

Tools Menu

The **Tools** menu is similar to Figure 3.7.

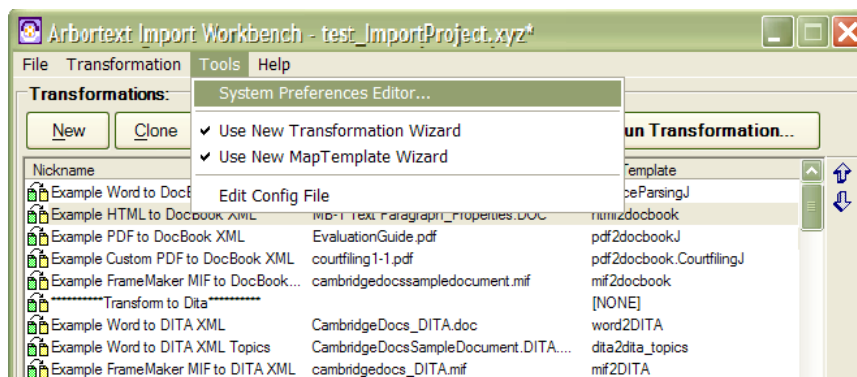


Figure - 3.7

The following options are available on **Tools** menu:

- **System Preferences Editor** — Opens the **System Preferences** dialog box allowing you to specify (and modify) how Arbortext Import calls its back-end Java platform, which default project to open, and so on. The values in this dialog box should only be rarely (if ever) changed.
 - **Home Directory** — Base directory where Arbortext Import is located.
 - **Repository Directory** — Base directory where projects, MapTemplates, and XSLTs are stored.

- **Bin Directory** — Base directory of the Arbortext Import back-end Java platform.
- **Java Virtual Machine Directory** — Location of Java Run-time Environment (JRE)
- **Default Project** – Project that is opened by Arbortext Import when it is first run, or when it cannot find the last opened project. Normally, Arbortext Import just opens the last project you were working with.

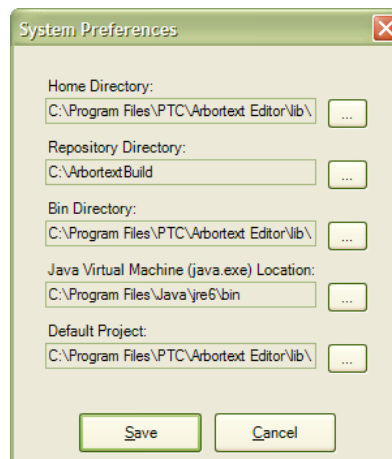


Figure - 3.8

- **Use New Transformation Wizard** — When checked, specifies whether the **New Transformation Wizard** will be launched when you click **New** and when you click the **Transformation** option in the **New** option of the **File** menu.
- **Use New MapTemplate Wizard** — When checked, specifies whether the **New MapTemplate Wizard** will be launched when you click **New** on the **MapTemplate** dialog box and when you click the **MapTemplate** option in the **New** option of the **File** menu.
- **Edit Config File** — Define custom variables and paths that can be used in various transformations and projects. Details of this feature are described in [How to Customize Configuration Files & Locations on page 164](#).

Help Menu

The following options are available on the **Help** menu:

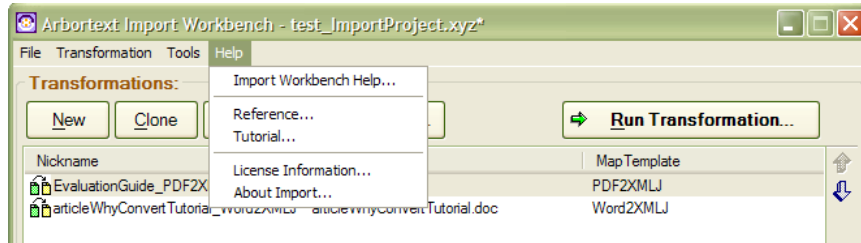


Figure - 3.9

- **Arbortext Import Workbench Help** — Opens the help file Arbortext Import Workbench online help.
- **Reference** — Opens the PDF version of the *Arbortext Import Reference*.
- **Tutorial** — Opens a PDF version of this *Arbortext Import Tutorial*.
- **License Information** — Opens a dialog box that shows your license information for Arbortext Import, or a dialog box that enables you to navigate to the license file itself.
- **About Import** — Opens the **About Import** dialog box, which shows the version of the product you are using, the version of the .NET Framework it is running on, and the open source and third-party software that is being used.

4

Introduction to MapTemplates & MapObjects

| | |
|---|----|
| Overview of MapTemplates | 46 |
| The Three Main Sections of MapTemplates..... | 46 |
| Setting up a New MapTemplate | 48 |
| Pre and Post-Processing Drivers | 52 |
| Pre and Post-Processing Drivers Interface | 53 |
| Overview of MapObjects | 56 |
| Main Functions of MapObjects..... | 56 |
| The Structure of a MapObject | 56 |
| Creating a new MapObject..... | 58 |
| MapObjects Details Pane..... | 67 |

Overview of MapTemplates

MapTemplates define the rules through which the transformations take place producing the destination file. They are a vital part of the transformation process.

The main functions of MapTemplate are:

- They direct the extraction of values from your source file
- They specify the creation of XML elements in the final document
- They define rules against sample files

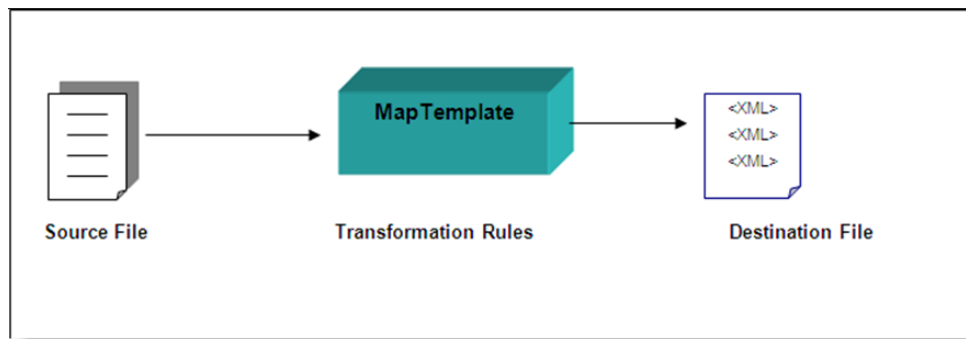


Figure - 4.1

The Three Main Sections of MapTemplates

MapTemplates contain three sections:

- Preprocessing — Prepares the input file for the main processing. Preprocessing is often used to normalize non-structured content, for example, Microsoft Word, HTML and FrameMaker to ppXML.
- Main processing — Contains the rules for a transformation. Main processing maps ppXML to a schema or DTD. This section contains MapObjects that map elements to elements.
- Post Processing — Includes XSLT or other external programs for further

transformation to the final output form. For example, this processing is necessary for HTML output, SCORM content, or alternative XML formats (WAP or WML).

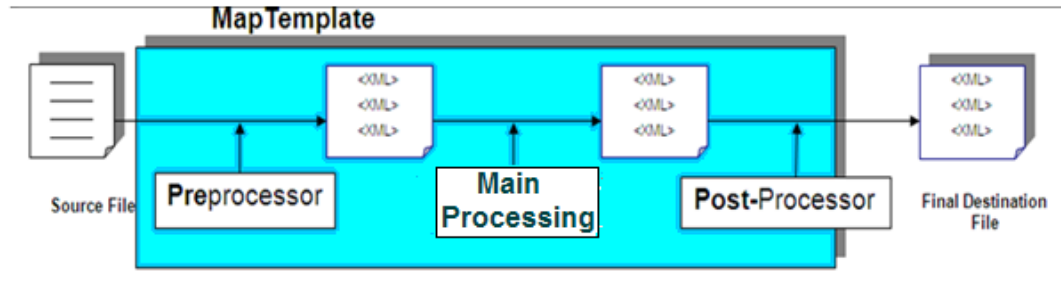


Figure - 4.2

MapTemplates can also create intermediate files that are needed for complex transformations. Occasionally, more than one intermediate XML file is needed to get the required details in the final output document.

The **MapTemplate Editor** enables you to edit and modify the MapTemplate rules. The MapTemplate Editor is shown in Figure 4.3.

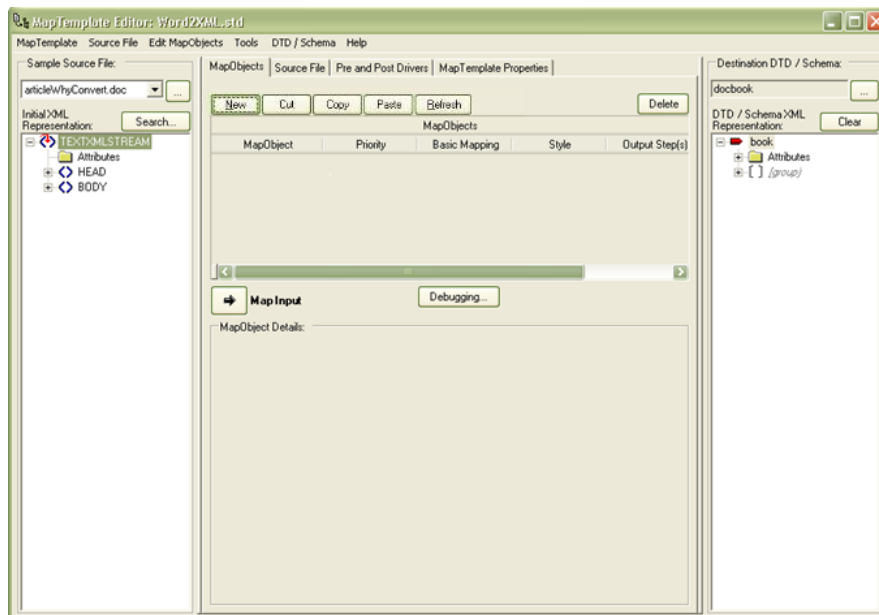


Figure - 4.3

The MapTemplate Editor is divided into three distinct panes:

- **Sample Source File** (left) pane — Displays the ppXML rendering of a particular source file. It displays a hierarchical tree view of the source file in the **Initial XML Representation** area as shown in Figure 4.3. The display consists of ppXML markup, as converted by the corresponding preprocessing driver.

- **MapObjects** (middle) pane — Contains tabs for editing **MapObjects**, **Source File**, **Pre and Post drivers** and **MapTemplate Properties**.
- **Destination DTD/Schema** (right) pane — Allows you to select, view, and clear the target DTD and Schema.

Details of the MapTemplate Editor are given in the *MapTemplate Editor* section of the *Arbortext Import Reference*.

Setting up a New MapTemplate

The tutorial will guide you through creating a new MapTemplate using Arbortext Import Workbench.

Set the Environment to Create a New MapTemplate

1. Ensure that Arbortext Import Workbench is launched as explained in [Launch Arbortext Import Workbench on page 14](#).
2. Load a separate project as explained in [Creating a New Project Using Arbortext Import Workbench on page 18](#).
3. Open the sample Word document in Microsoft Word. Examine the document and notice all the different formatting used in this document.

Create the New MapTemplate

1. Select **File ► NewMapTemplate** on the main Arbortext Import Workbench as shown in Figure 4.4.

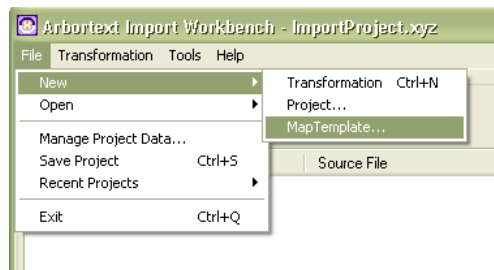


Figure - 4.4

2. A **New MapTemplate Wizard** is opened. Specify the **MapTemplate Name** for MapTemplate as in Figure 4.5. The new MapTemplate is created by default at `<home>\STDTemplates`.

You can also save this MapTemplate at some other location using **Browse**.

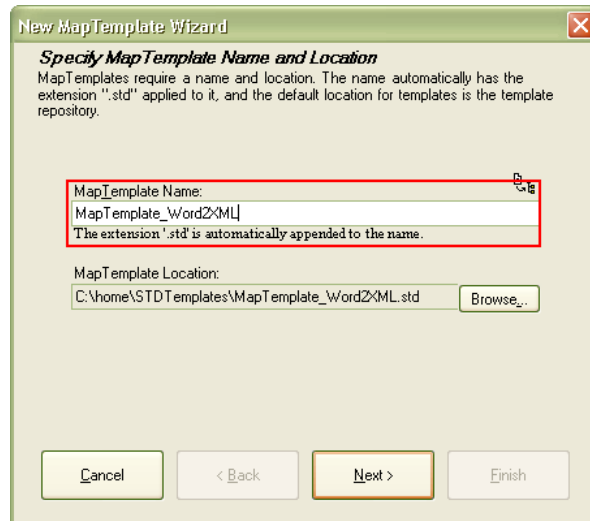


Figure - 4.5

3. Specify name for the MapTemplate.
4. Click **Next** to continue.
5. In the next **Specify New or Cloned MapTemplate** window, you can either replicate an existing MapTemplate by selecting **Clone Pre-existing Template** or you can create new MapTemplate. Select **Create New MapTemplate**.

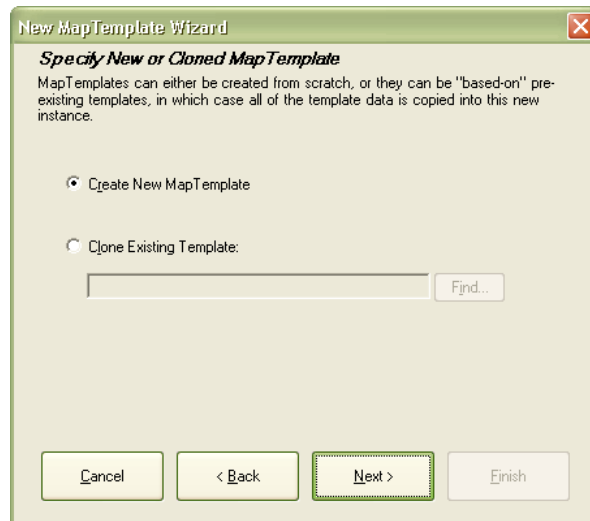


Figure - 4.6

6. Click **Next** to create a new MapTemplate.

7. In the **Specify MapTemplate Driver** window, select Word to XML Driver (Java) from the **Driver** list as shown in Figure 4.7.

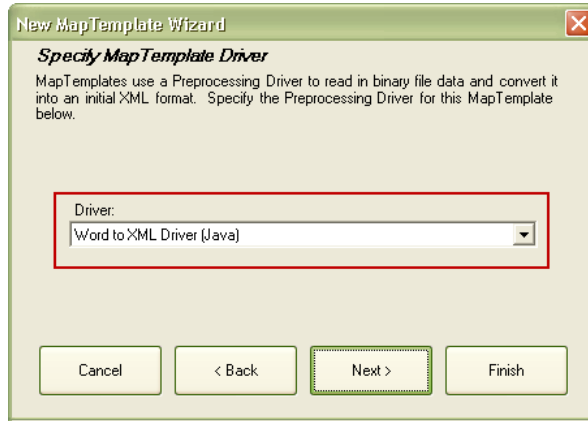


Figure - 4.7

8. Click **Next**.
9. Now the **Specify MapTemplate Driver Options** window is displayed. Here driver options and the custom name of the driver can be modified as shown in Figure 4.8. Details of these driver options are given in the *Pre and Post Processor Drivers* section of the *Arbortext Import Reference*. Leave the default driver options as they are.

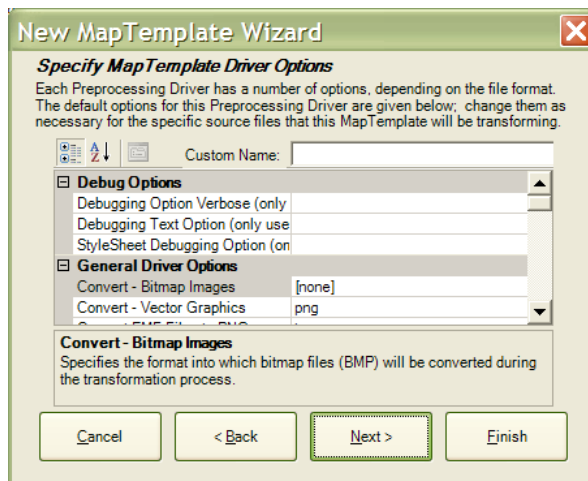


Figure - 4.8

10. Click **Next** to continue.
11. In the next **Specify MapTemplate Schema** window you can select a particular schema to identify layout rules of the output or destination file.

To specify a schema:

- a. Select **Use selected schema**.

- b. Click **Select Schema** as shown in Figure 4.9.

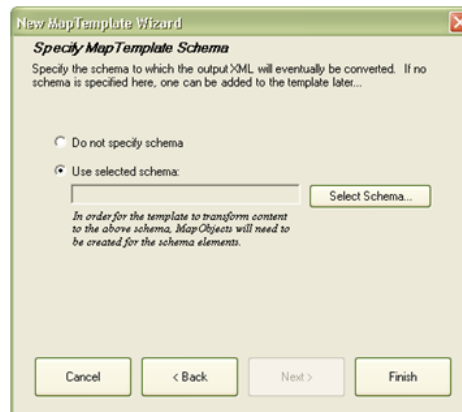


Figure - 4.9

- c. The **Select DTD / Schema** window is opened as shown in Figure 4.10.

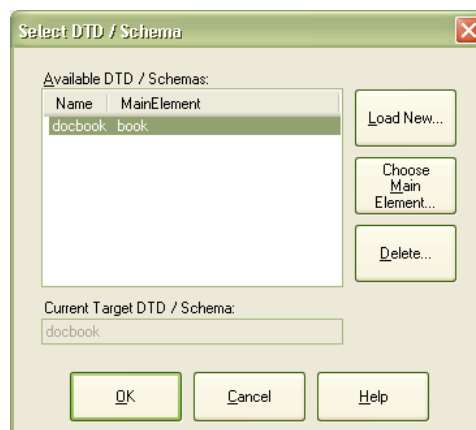


Figure - 4.10

- d. You can either select a schema from the given list in the **Available DTD / Schemas:** pane, or click **Load New** to navigate to your required schema.

In case of a new schema:

- i. Select a main element by clicking **Choose Main Element**.
- ii. Click **OK**.

- e. You can also delete any selected schema using **Delete**.

- f. Select DocBook from the available list.

- g. Click **OK**.

12. Click **Finish** on the **Specify MapTemplate Schema** window. As a result, the **MapTemplate Editor** window is opened.

13. In the left pane of the window, select the source document by clicking **Browse**.
Doing so will load the ppXML tree, representing the XML structure of the source document in the left pane as shown in Figure 4.11.
14. Open the **DTD / Schema** menu and enable **Turn Off All Groups and Qualifiers**.
This will make the schema view clear in the right pane of the window as shown in Figure 4.11.

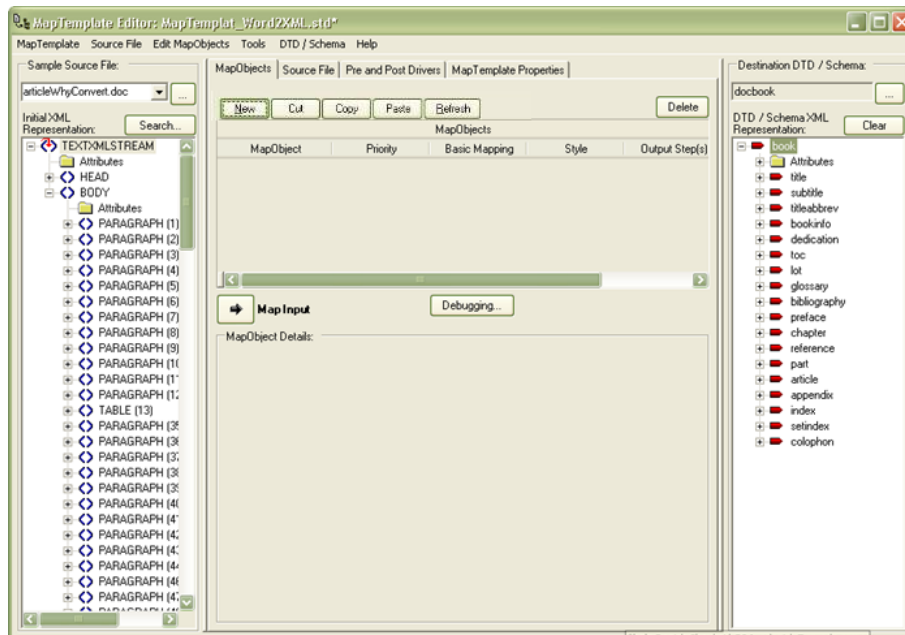


Figure - 4.11

15. To save your newly created MapTemplate, select **MapTemplate ► Save MapTemplate** on the **MapTemplate Editor**.
16. You can now specify preprocessing rules, MapObjects, and post processing rules on the **MapTemplate Editor**.
17. Choose the **MapTemplate** menu on the **MapTemplate Editor** and select **Save and Run** to save any changes made in the MapTemplate.
18. Execute the transformation.

Pre and Post-Processing Drivers

Preprocessing and Post-Processing Drivers are auxiliary data transformations, applied respectively before and after the MapObject processing for a given MapTemplate.

Both Preprocessing and Post-Processing drivers are listed in the **Pre and Post Drivers** tab of the main window of the **MapTemplate Editor**. No restrictions exist on the number of either type of drivers that are present in a MapTemplate, although in many cases a MapTemplate contains only one Preprocessing Driver and no Post-Processing Drivers.

The MapTemplates which require no drivers are those which process XML files using MapObjects only.

Preprocessing Drivers

A common use of a Preprocessing Driver is the conversion of a binary-based format (for example, Microsoft Word *.doc files) into ppXML that can then either be published or converted further using MapObjects.


Preprocessing options enable you to normalize content into ppXML for easier parsing. Preprocessing calls Arbortext Import drivers to automatically convert files into ppXML. There are preprocessing options for many different file types, including Word, RTF, and HTML files.

Post-Processing Drivers

A Post-Processing Driver is often used to invoke a subsequent MapTemplate or XSL transformation in a convenient and modular way. Examples include transforming to HTML view at the end of a transformation. (For more details related to view as HTML see [View As HTML on page 29.](#))

Pre and Post-Processing Drivers Interface

The **Pre and Post Drivers** tab located on the MapObject pane enables you to manage the pre- and post-processing drivers used by your MapTemplate.

New pre-and post processing drivers can also be added using **New Preprocessing Driver** or **New Post-Processing Driver** .

- When you click **New Preprocessing Driver** within the **Preprocessing Drivers** area, a new **Processing Driver Options** dialog box is opened as shown in Figure

4.12. You can select any driver or stylesheet as a preprocessing driver from the **Driver** list.

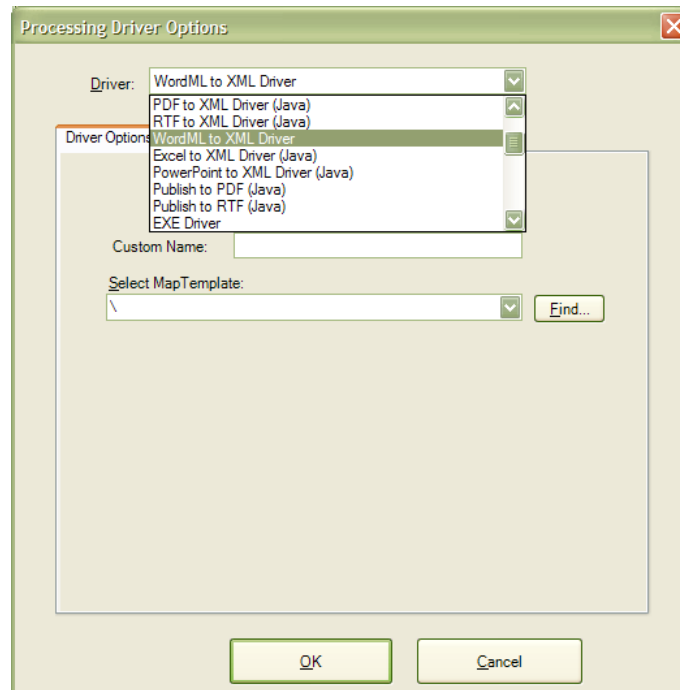


Figure - 4.12

- When you click **New Post-Processing Driver** within the **Post-Processing Drivers** area, a new **Processing Driver Options** window is opened as shown in Figure 4.12. You can select any driver or stylesheet as a postprocessing driver from the **Driver** list.

The driver options can be customized according to your requirements using the **MapTemplate Editor**.

There are no material differences between preprocessing and post-processing drivers,

other than the fact that preprocessing is done before the MapObject processing and post-processing is done after MapObject processing.

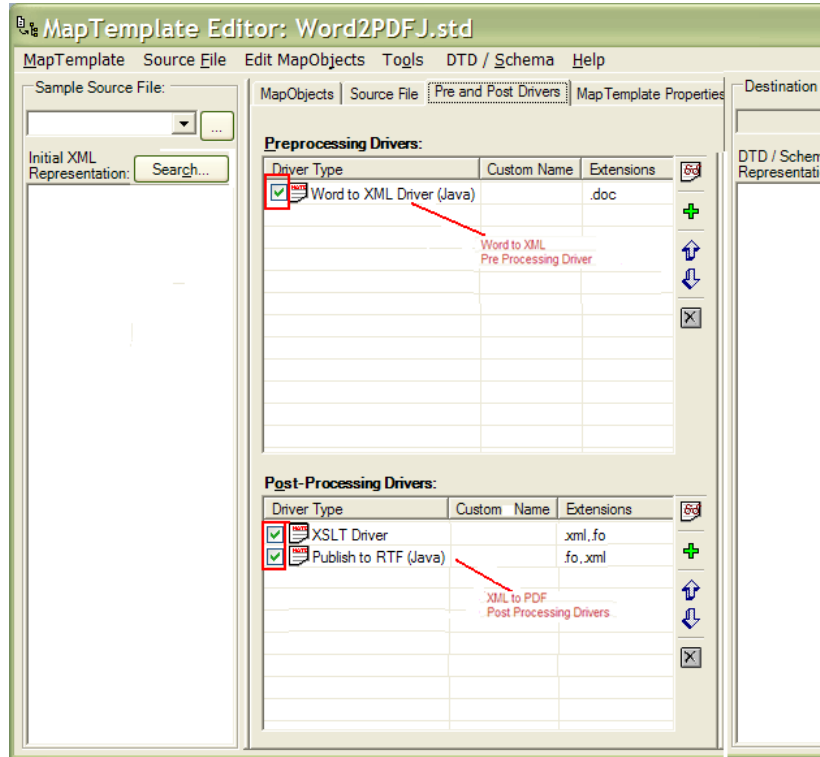



Figure - 4.13

To display the driver options, select a driver and click **Edit Preprocessing Driver/Edit Post-processing Driver**  or double-click on any driver.

When you select the driver options as stated before, a **Processing Driver Options** dialog box is opened. Here, driver options can be added and excluded using different options. Details of these driver options are given in the *Pre and Post Processor Drivers* section of the *Arbortext Import Reference*.

Turning on and off single or multiple drivers

The **MapTemplate Editor** enables you to turn on and off single or multiple drivers by selecting and deselecting them as shown in Figure 4.12. Using these options, you can select or deselect any pre-post processing drivers.

Overview of MapObjects

MapObjects are the building blocks of MapTemplates, and serve as the cornerstone of Arbortext Import's ability to transform a free-form unstructured or semi-structured document into XML.

MapObjects allow you to extract meaningful values out of unstructured content. That in turn allows you to produce meaningful XML. The form of the final output depends entirely on the function of your transformations and the nature of the source documents that you want to transform.

Main Functions of MapObjects

MapObjects perform the following main functions:

- Programmatically select content (text, paragraphs, and so on) in the source document
- Output one or more XML elements
- MapObjects are mapped one-to-one with the source elements (the XML elements generated in the previous item).
- Optionally, a MapObject can also create a hierarchy above or below the source element
- MapObjects that are reusable can be referenced from other MapTemplates

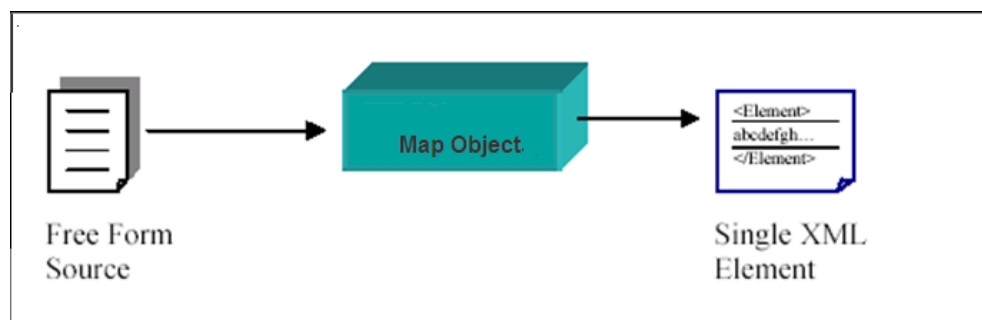


Figure - 4.14

The Structure of a MapObject

MapObjects contain three parts: input rules, output rules, and child MapObjects and rules.

Input Rules

Input rules are used to find specific content or markup within a source document. The input selection is the content found by the input rules.

Input rules are the crux of MapObjects. They specify how the MapObject parses the source files to get a certain value.

- The returned value is called the current selection.
- If the input is a text stream or document, then the current selection is called a text selection.
- If the input is an XML document or stream, then the current selection is called an element selection.

XML input rules can be applied to virtually any XML schema. The user interface of the MapTemplate Editor is designed to work best with the ppXML, and generic enough that it can be applied to any XML document structure. XML elements can be selected based on attributes, and element text can be parsed.

Input rules have their own tab window in the **MapObject Details** area of the **MapObjects** tab. On the **Input Rules** tab, there is a basic mapping type field (containing the values based on predefined rules for ppXML fields), and an English description of the input rules. In all cases, the input rules can be edited after their initial creation. That editing can be done using the **Map Input Wizard** or by clicking **Edit Input Rules** on this tab.

Details regarding the input rules can be found in the *XML Input Rules* section of the *Arbortext Import Reference*.

Output Rules

Once a selection has been done by selecting text or an XML element, a MapObject usually creates at least one output XML element (the single XML element that is produced is called the core output element) that appears in the destination XML file. Output rules control the appearance and content of the output element.

A destination XML element might look something like this:

```
<PressRelease type="corporatenews" date="5/12/03">  
  text and child elements  
</PressRelease>
```

Output rules specify how Arbortext Import constructs the output XML markup to include elements, attributes, and text.

Output rules have their own tab in the **MapObject Details** area of the **MapObjects** tab.

Details regarding the Output rules can be viewed in the *Output Rules* section of the *Arbortext Import Reference*.

Child MapObjects and Rules

After a MapObject processes input data to find and process content, it can call child MapObjects that can further process the content inside the original selection. For example, if PARAGRAPH has been selected by a MapObject, child MapObjects and rules can then process the text inside the PARAGRAPH, as well as any inline elements, as shown in Figure 4.15.

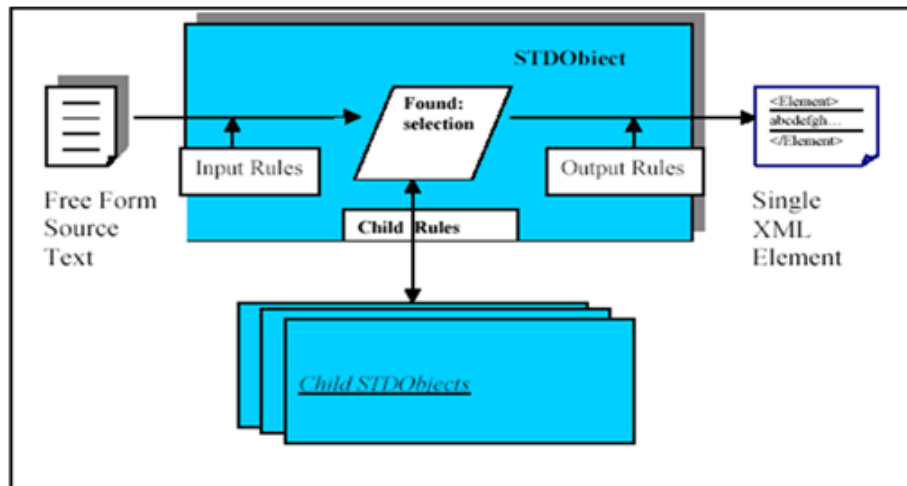


Figure - 4.15

Creating a new MapObject

The best way to understand how to create MapObjects is to create one. The tutorial will now guide you through the steps to create new MapObjects and to apply input rules and output rules to these MapObjects.

To set environment to create MapObject:

1. Ensure that Arbortext Import Workbench is started, a project and **Word2XMLJ.std** MapTemplate is open in the MapTemplate Editor with the source document **articleWhyConvert.doc** loaded in the left pane, and the schema is loaded in the right pane of the window.
2. To load the schema:
 - Click **Browse** on the **Destination DTD/Schema:** pane that will show you the list of target schemas and DTDs in the **Select DTD/ Schema** window.
 - Select “DocBook” from the **Available DTD/Schema** list.
 - Click **OK** to load the schema in the MapTemplate.

articleWhyConvert.doc is located in the following directory:

Arbortext-path\samples\importexport\word

This is the correct environment to create a new MapObject. At this point the **MapTemplate Editor** will appear as shown in Figure 4.16.

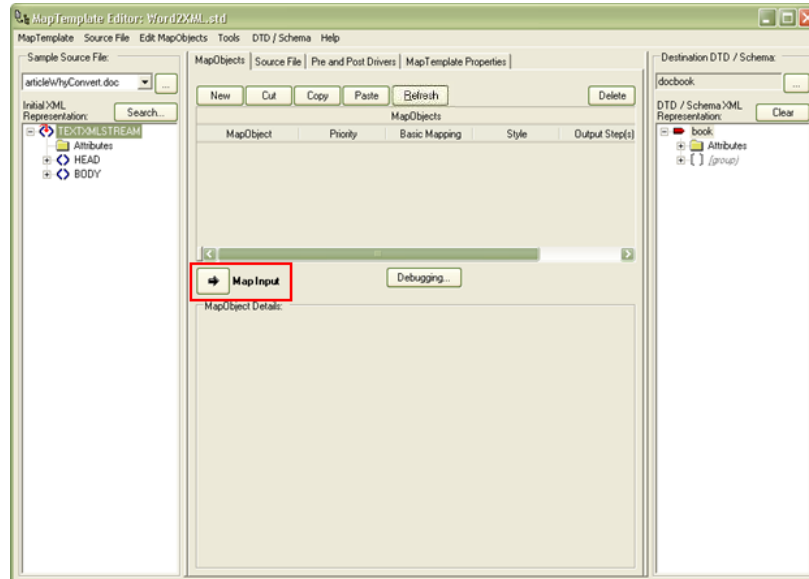


Figure - 4.16

To create top level object:

1. Select the TEXTXMLSTREAM element in the left pane of the window from the ppXML tree. This is the top element in the XML hierarchy.
2. Click **Map Input** to start the **Map Input Wizard**. The **Map Input Wizard** will guide you through how to create a MapObject, and will guide you in defining the rules for the MapObject.

3. On the first window of the wizard, appears as shown in Figure 4.17, ensure that the **Create New MapObject** option is selected.

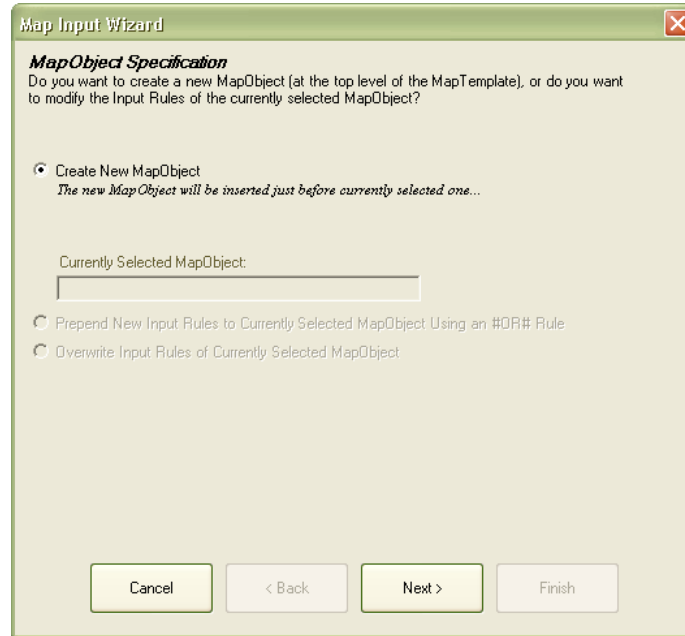


Figure - 4.17

4. Click **Next**.
5. On the next **MapObject Properties** window, specify the name of the MapObject. You can also select the name of the MapObject similar to the name of the current selected element in the source file, such as TEXTXMLSTREAM in this case.

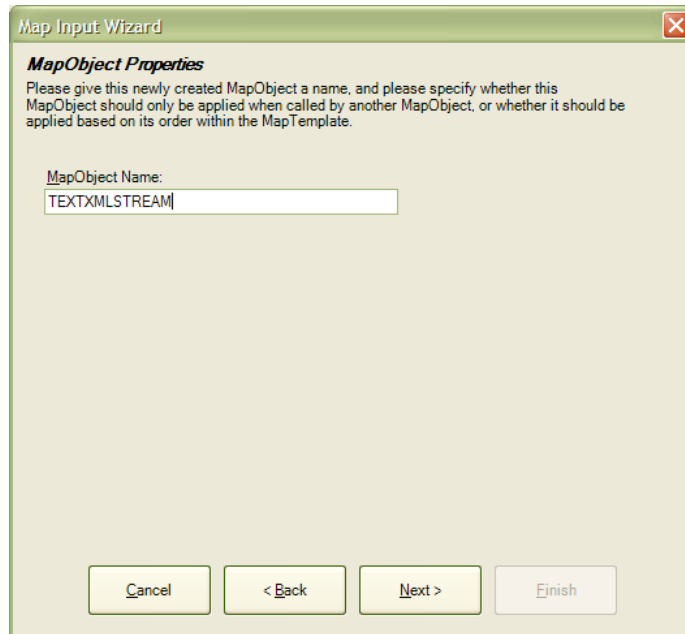


Figure - 4.18

6. Click **Next** to move to the next window.
7. On the **MapObject Basic Input Mapping** window, ensure that the **Basic Input Mapping** option is set to TEXTXMLSTREAM from the list as shown in Figure 4.19.

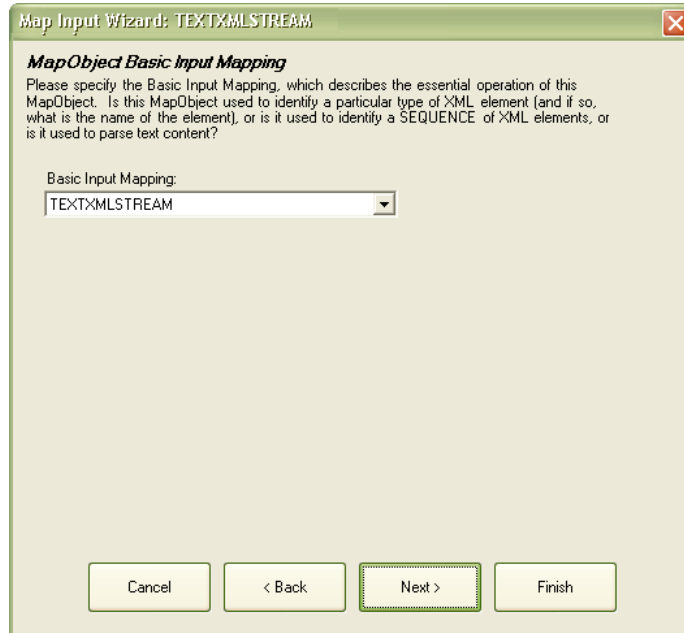


Figure - 4.19

8. Click **Next**.

9. On the **MapObject Text Verification Editor** window set the text input rules to ensure that the MapObject only selects the source element whose text contains or starts with a defined text string.



Figure - 4.20

10. Click **Next**.

11. On the **MapObject Input Hierarchy Refinement** window as shown in Figure 4.21, you can specify the ancestor, descendant, and output relationship that a source element must have in order for the MapObject to select the element.



Figure - 4.21

12. Click **Next**.

13. On the **MapObject Advanced Rule Editor** window you will see the input rules that

you have set so far. You can add and edit any XML rules here, but for now just click **Finish**.

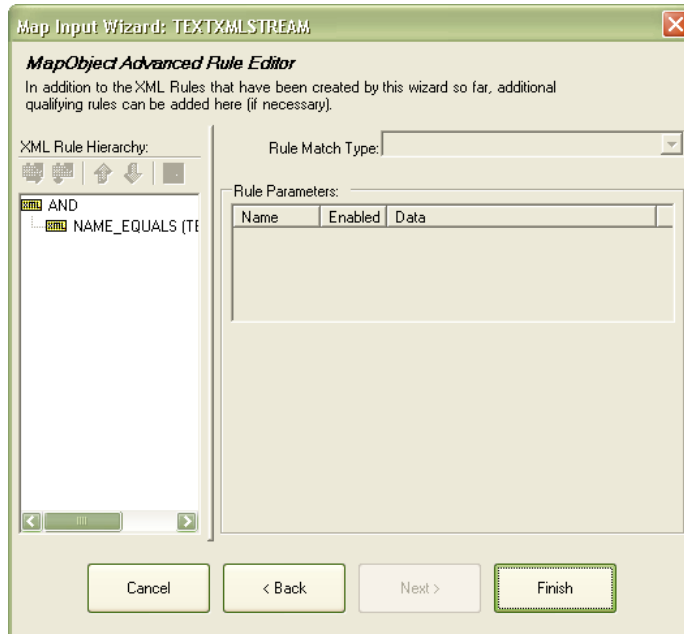


Figure - 4.22

The MapTemplate Editor will add the TEXTXMLSTREAM MapObject to the list, as shown in Figure 4.23.

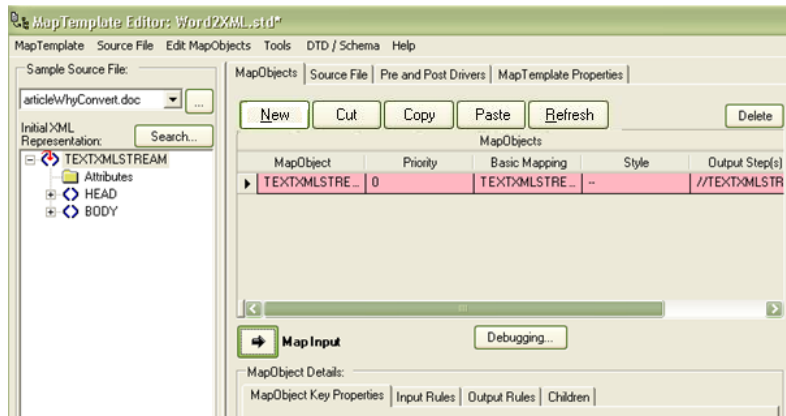


Figure - 4.23

You've created your first MapObject. Next, you will create five more MapObjects, using the steps you just learned.

Create the HEAD MapObject

1. Select the HEAD element in the left pane.

2. Create a new MapObject using the following settings:
 - **MapObject Name:** HEAD
 - **Basic Input Mapping:** HEAD
3. Click **Finish**.

Create the BODY MapObject

1. Select the BODY element in the left pane.
2. Create a new MapObject using the following settings:
 - **MapObject Name:** BODY
 - **Basic Input Mapping:** BODY
3. Click **Finish**.

Create the PARAGRAPH MapObject

1. Expand the BODY element.
2. Select the third PARAGRAPH element in the left pane.

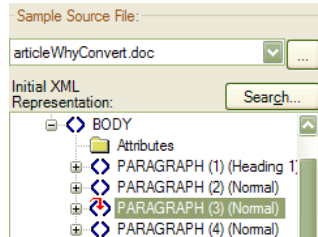


Figure - 4.24

3. Create a new MapObject using the following settings:
 - **MapObject Name:** PARAGRAPH
 - **Basic Input Mapping:** PARAGRAPH
4. On the **MapObject Style Mapping** window use following:
 - **Only Map Source Content with the Following Style:** Normal
5. Click **Next**.
6. The **MapObject Attribute Mappings** window appears as shown in Figure 4.25. On this screen, you can further refine attribute mappings by keeping track of the current value of the attribute. (You will learn more about the MapObject Attribute

Mappings in Customizing the Top Level MapObjects on page 94.) Leave default values as they are and click **Finish**.

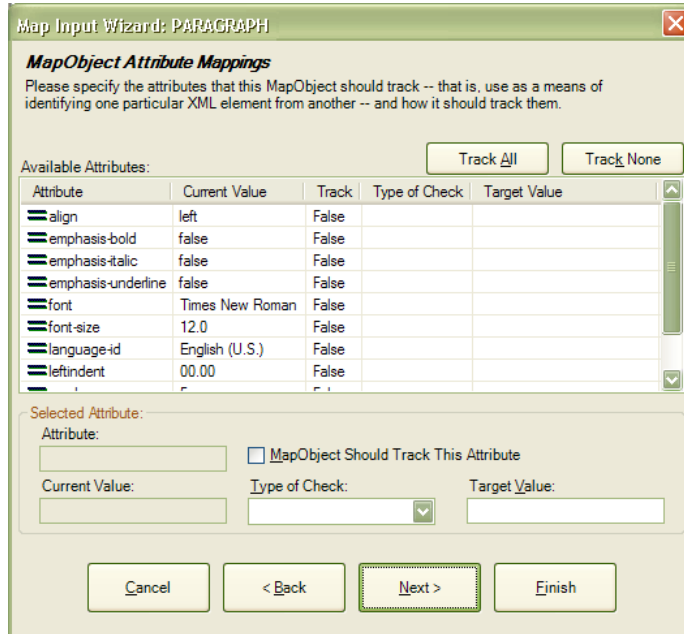


Figure - 4.25

Create the SUBHEADING MapObject

1. Expand the BODY element.
2. Select the second PARAGRAPH element in the left pane.
3. Create a new MapObject using the following settings:
 - **MapObject Name:** SUBHEADING
 - **Basic Input Mapping:** PARAGRAPH
4. On the **MapObject Style Mapping** window select the following:
 - **Only Map Source Content with the Following Style:** Heading 2
5. On the **MapObject Attribute Mappings** window leave default values as they are and click **Finish**.

Create the Heading MapObject

1. Expand the BODY element.
2. Select the first PARAGRAPH element in the left pane.
3. Create a new MapObject using the following settings:

- **MapObject Name:** Heading
 - **Basic Input Mapping:** PARAGRAPH
4. On the **MapObject Style Mapping** window, select the following:
 - **Only Map Source Content with the Following Style:** Heading 1
 5. On the **MapObject Attribute Mappings** window, leave default values as they are and click **Finish**.

Now you have six MapObjects present in the MapObjects list (priority 0-5) as shown in Figure 4.26.

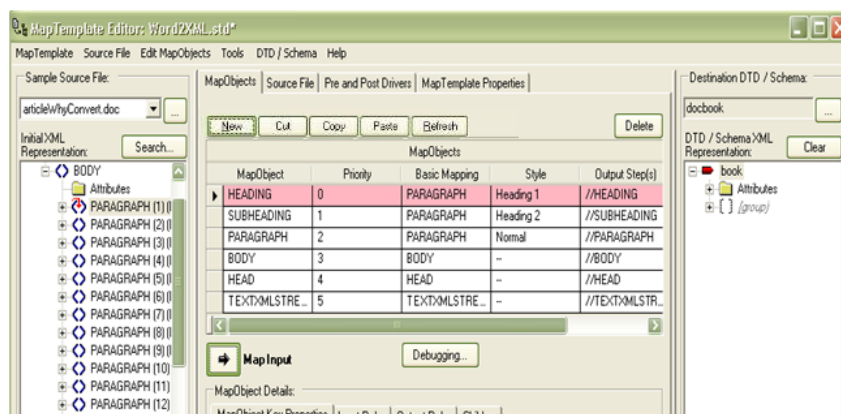


Figure - 4.26

Click the **MapTemplate** menu and select **Save MapTemplate** to save your new MapTemplate.

MapObjects Details Pane

This pane is on the lower half of the **MapTemplate Editor**. There are several tabs on this pane that are used to edit and modify input, output rules, and MapObject properties.

MapObject Key Properties Tab

This tab enables you to specify the MapObject name (names should be unique), and set

the priority of the MapObjects in the list. The priority determines the order in which they are called. 0 is the highest priority.

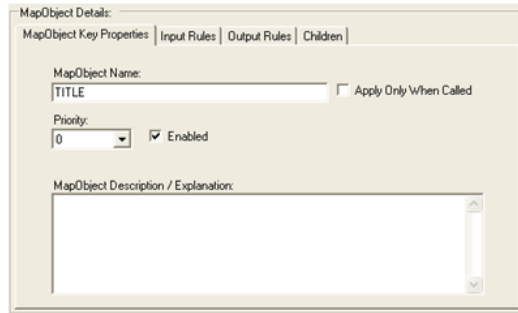


Figure - 4.27

This tab also enables you to define how you want to call MapObjects. You can check **Enable** to specify that the MapObject is called when you are applying all MapObjects. You can also deselect the **Enable** option for a MapObject, and good way to disable a MapObject during MapTemplate development. Also, if you deselect the **Apply Only When Called** option, you will have to explicitly call that particular MapObject either in input rules or as a child MapObject.

Additionally, you can use **MapObject Description/Explanations** to annotate and comment each MapObject.

Input Rules Tab

On the **Input Rules** tab, input rules are defined for the selected MapObjects. The input rules of a MapObject are used to find specific content or markup within a source document.

On the **Input Rules** tab, there is a list for input rules basic mapping and for an English description of the input rules. In all cases, the input rules can be edited after their initial creation by clicking **Edit Input Rules** as shown in Figure 4.28.

Details regarding the **Input rules** tab can be found in the *MapTemplate Editor* section of the *Arbortext Import Reference*.

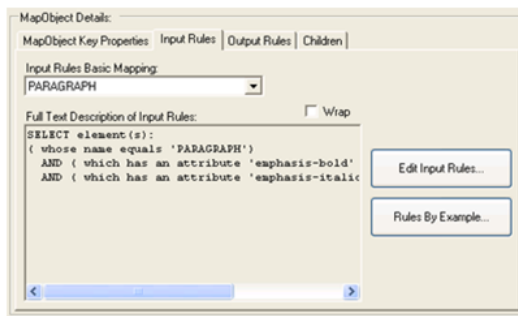


Figure - 4.28

Output Rules Tab

On the **Output Rules** tab, output rules are defined for the selected MapObjects. Once a selection is found, the MapObject usually creates an output element. Output rules control the appearance and content of the output element. Details regarding the **Output Rules** tab can be found in the *MapTemplate Editor* section of the *Arbortext Import Reference*.

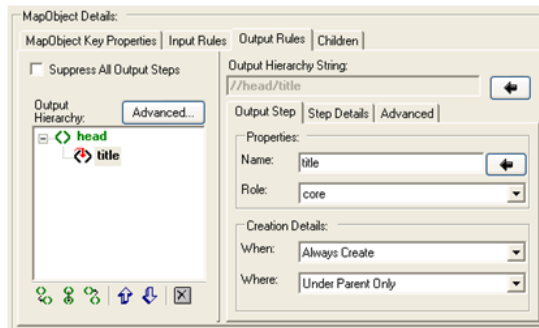


Figure - 4.29

Children Tab

This tab enables you to define rules for the child elements. The child elements section enables you to control options related to the processing of child elements.

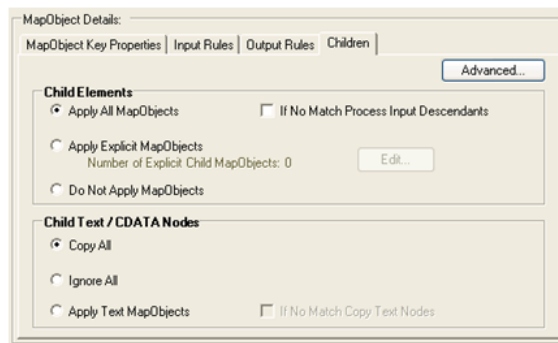


Figure - 4.30

The **Child Text/CDATA Nodes** section enables you to control options related to the processing of child text and CDATA nodes.

Details regarding the **Children** tab can be found in the *MapTemplate Editor* section of the *Arbortext Import Reference*.

5

File Processing Scenarios for Arbortext Import Supported Transformations

| | |
|---|----|
| Introduction..... | 72 |
| Importing a Microsoft Word 2003 (.doc) document into XML..... | 72 |
| Importing a Microsoft Word 2007 or 2010 (.docx) document into XML | 78 |
| Importing an HTML document into XML | 79 |
| Importing a WordML document into XML..... | 79 |

Introduction

This chapter explains scenarios to transform Microsoft Word 2003 (.doc), Microsoft Word 2007 and 2010 (.docx), HTML, and WordML documents into XML. This chapter explains the step by step procedure for the following transformations:

- Importing a Microsoft Word 2003 (.doc) document into XML
- Importing a Microsoft Word 2007 or 2010 (.docx) document into XML
- Importing an HTML document into XML
- Importing a WordML document into XML

Importing a Microsoft Word 2003 (.doc) document into XML

With the help of Arbortext Import Workbench, you can import Microsoft Word 2003 documents (.doc files) into XML.

You will perform the following steps using the Arbortext Import Workbench window:

1. Create a new transformation.
2. Select the source Word (.doc) document.
3. Select the Word2XMLJ MapTemplate.
4. Execute the transformation.

Creating a new transformation in Arbortext Import Workbench

To create a new transformation:

Select **File ► NewTransformation** from the Arbortext Import Workbench.

 **Note**

The new transformation can also be created using **New** on the main window, as shown in Figure 5.1.

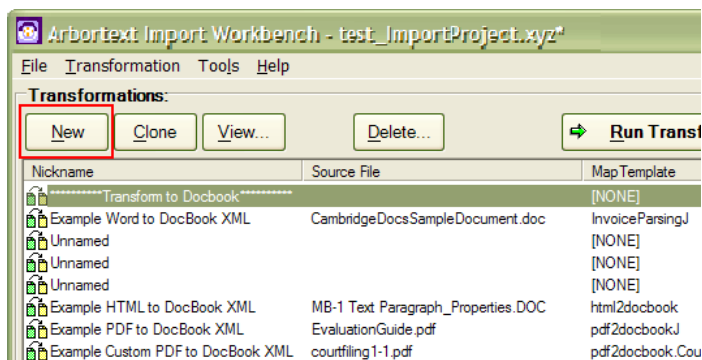


Figure - 5.1

Selecting the Source Document

Once you have selected **Transformation**, a **New Transformation Wizard** is opened in a separate window.

To select a source file:

1. Click **Find** to browse for the source Word document (.doc). Use **URL** on this window to give the exact path of a source file that is located on a network.

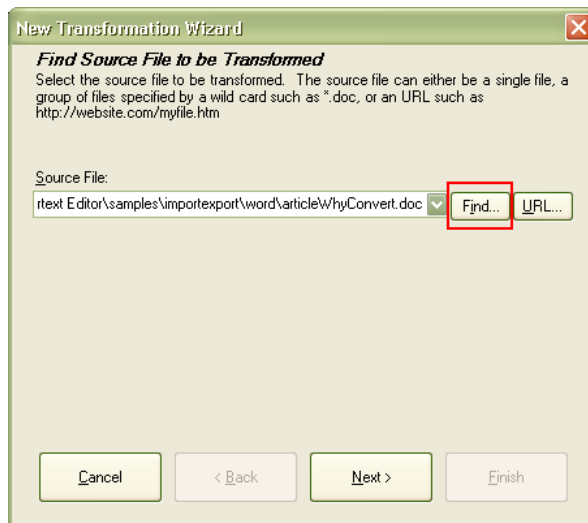


Figure - 5.2

2. Sample word documents are provided with the installation of Arbortext Import. These are located in **Arbortext-path\samples\importexport\word**.
For this Word transformation example, select **articleWhyConvert.doc** from that directory.
3. After selecting the Word source file, click **Next** to continue.

Selecting the Word2XMLJ Template

After selecting the source file, select the appropriate MapTemplate. In this procedure, select **Word2XMLJ.std**.

1. Select **Custom/Other** on the **Select Transformation** window.
2. Click **Next**. The next MapTemplate selection window will appear, as shown in Figure 5.3.



Figure - 5.3

3. Click **Find** to locate the MapTemplate. This will open the default location of MapTemplates. That is, **<home>\STDTemplates\Common**.

4. Select the **Word2XMLJ .std** MapTemplate.

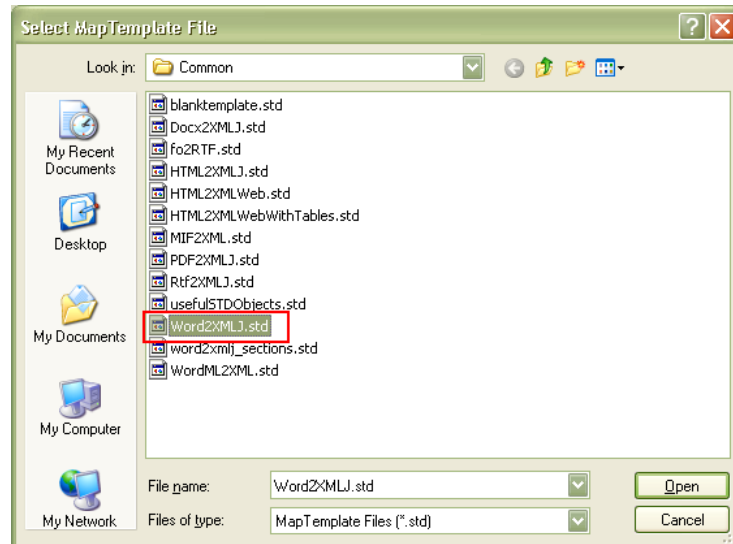


Figure - 5.4

5. After selecting the Word2XMLJ MapTemplate, click **Next**.
6. The **Provide Transformation Nickname** window for defining the nickname of the transformation is displayed. Here, you will see the default value for the transformation nickname, created from the selected source file and the MapTemplate names. You can change the nickname as you wish, but for now, leave it as default.

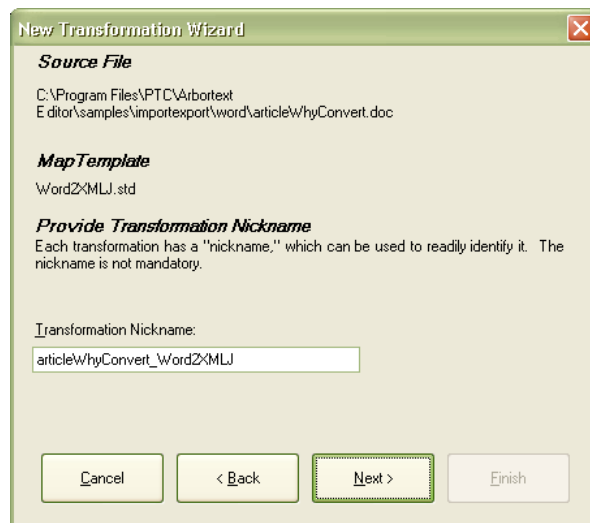


Figure - 5.5

7. Click **Next**. The **Run Transformation** window is displayed.

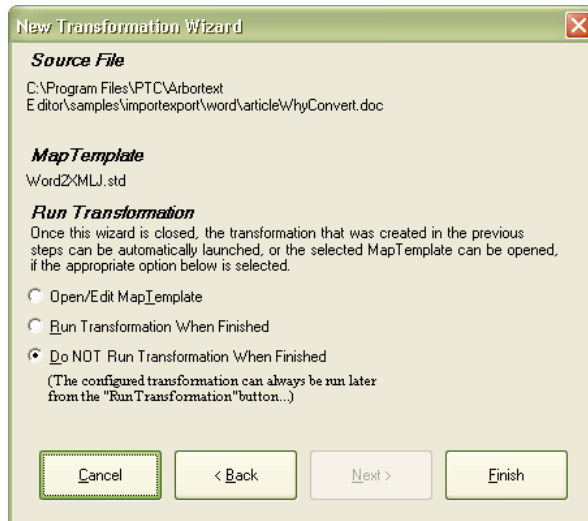


Figure - 5.6

8. With the **Run Transformation** window, you can select any one of the options:
 - **Open/Edit MapTemplate** — Opens the MapTemplate Editor to further edit the selected template.
 - **Run Transformation When Complete** — Immediately runs the transformation using the preferences you just entered.
 - **Do Not Run Transformation When Complete** — Does not run the transformation, but will put the transformation, along with all your settings, into the transformation list on the main window.

On the **Run Transformation**, select **Do Not Run Transformation When Complete**.

9. Click **Finish** The transformation wizard will return you to the Arbortext Import Workbench main window with your created transformation.

Executing the Transformation

At this point, the Arbortext Import Workbench shows all the basic details related to the newly created transformation as shown in Figure 5.7.

The transformation nickname, source file, selected MapTemplate, and path of the

destination file are specified in the text boxes on the **Mapping** tab under the **Transformation Details** section on the Arbortext Import Workbench main window.

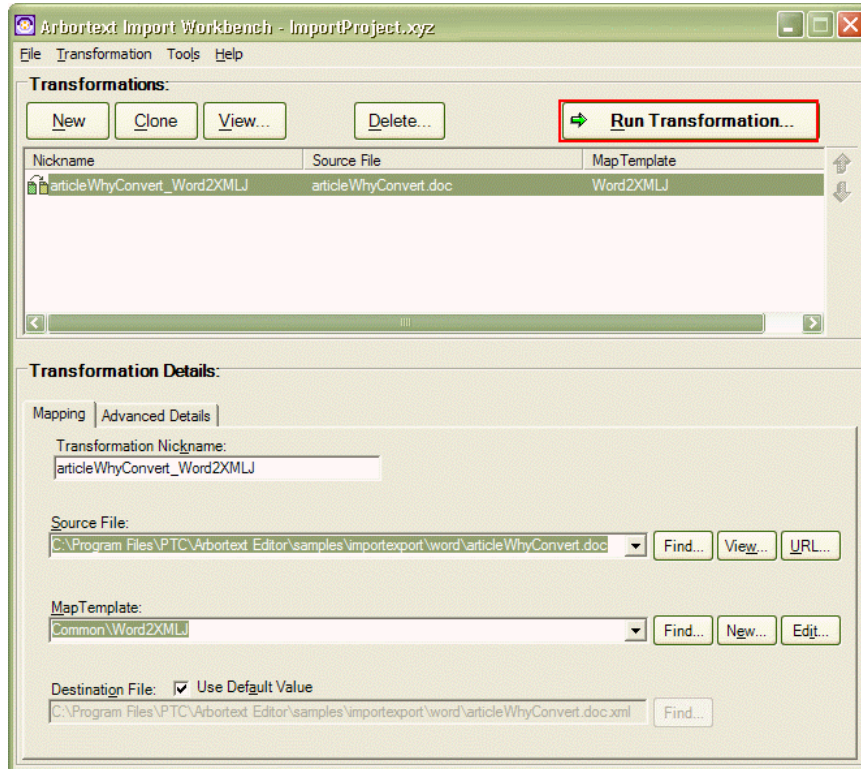


Figure - 5.7

You can also select your desired location and name for the destination file.

To specify your desired location for the destination file:

1. Deselect the **Use Default Value** option.
2. Click **Find** adjacent to the **Destination File**.
3. On the **Save As** window, browse for the location where you want to save the destination file.
4. Enter the desired name for the destination file in the **File name** field.
5. Click **Save**. The Arbortext Import Workbench main window is displayed.

To run the transformation:

1. On the Arbortext Import Workbench main window, select the transformation you have created.

2. Click **Run Transformation** as shown in Figure 5.7. When you click **Run transformation**, the **Transformation** window will open showing all the processes involved in the transformation as shown in Figure 5.8.

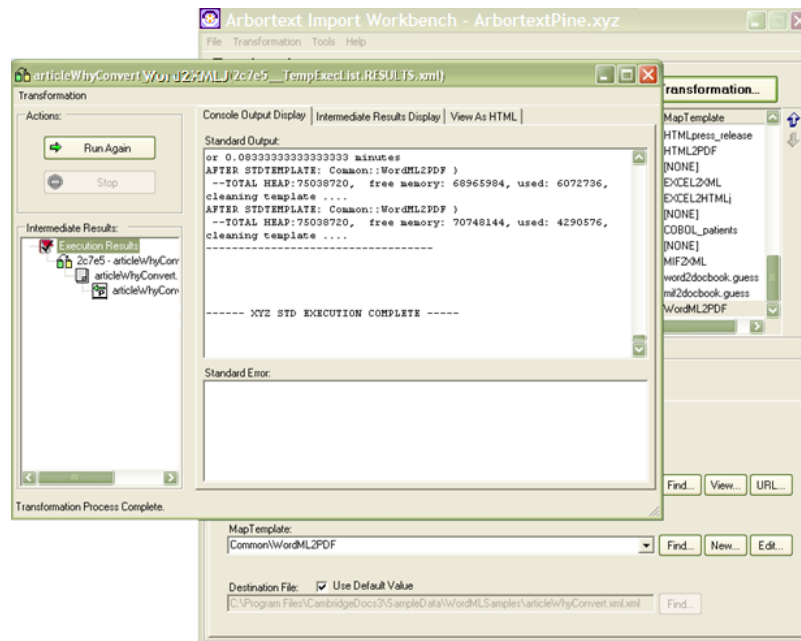


Figure - 5.8

Once the transformation is completed, an XML document will be created at the path given in the **Destination File**. The Word document has been successfully transformed to XML.

In the **Transformation** window three tabs are shown:

- **Console Output Display**
- **Intermediate Results Display**
- **View As HTML**

Each of these tabs is explained previously in Chapter 2.

Importing a Microsoft Word 2007 or 2010 (.docx) document into XML

Using Arbortext Import, you can import Microsoft Word 2007 and 2010 (.docx) documents into XML. Perform the following steps for this transformation:

1. Create a new transformation as described in [Creating a new transformation in Arbortext Import Workbench](#) on page 72.

-
2. Select the source **.docx** file as described in [Selecting the Source Document on page 73](#). In this case, select a **.docx** document from the following location:
`Arbortext-path\lib\samples\importexport\docx`.
 3. Select the Docx2XMLJ template as described in [Selecting the Word2XMLJ Template on page 74](#). In this case, instead of selecting **Word2XMLJ.std**, select the **Docx2XMLJ.std** MapTemplate.
 4. Execute the transformation as described in [Executing the Transformation on page 76](#).

Importing an HTML document into XML

Using Arbortext Import, you can import HTML documents into XML. Perform the following steps for this transformation:

1. Create a new transformation as described in [Creating a new transformation in Arbortext Import Workbench on page 72](#).
2. Select the source HTML file as described in [Selecting the Source Document on page 73](#). In this case, select an HTML document from the following location:
`Arbortext-path\lib\samples\importexport\html`.
3. Select the HTML2XMLJ MapTemplate as described in [Selecting the Word2XMLJ Template on page 74](#). In this case, instead of selecting **Word2XMLJ.std**, select the **HTML2XMLJ.std** MapTemplate.
4. Execute the transformation as described in [Executing the Transformation on page 76](#).

Importing a WordML document into XML

Using Arbortext Import, you can import WordML documents into XML. Perform the following steps for this transformation:

1. Create a new transformation as described in [Creating a new transformation in Arbortext Import Workbench on page 72](#).
2. Select the source WordML file as described in [Selecting the Source Document on page 73](#). In this case, select a WordML document from the following location:
`Arbortext-path\lib\samples\importexport\wordml`.
3. Select the WordML2XML MapTemplate as described in [Selecting the Word2XMLJ Template on page 74](#). In this case, instead of selecting **Word2XMLJ.std**, select the **WordML2XML.std** MapTemplate.
4. Execute the transformation as described in [Executing the Transformation on page 76](#).

6

Transforming Word, FrameMaker, and HTML Documents into DocBook

| | |
|--|-----|
| Introduction..... | 82 |
| Overview of the DocBook MapTemplates | 82 |
| How the Templates Work..... | 83 |
| How to Customize the DocBook Template..... | 87 |
| Conclusion..... | 129 |

Introduction

Arbortext Import can transform existing content such as Word, PDF, FrameMaker, and HTML documents into any target XML format. Arbortext Import is shipped along with sample templates for DocBook documents. DocBook is a popular XML standard that is used for technical documentation that has been adapted to be used in many other industries. For more information on DocBook, refer to <http://www.docbook.org>.

The examples included with Arbortext Import are configured to transform sample documents into DocBook. They are meant to be customized and used to transform your own custom source files into DocBook. Because the DocBook templates touch many aspects of doing conversions to XML, they can also serve as examples of how to build templates for other XML targets.

Overview of the DocBook MapTemplates

This chapter and the DocBook MapTemplates serve as reference examples for the transformation of contents into meaningful XML. Even if you are transforming Word, FrameMaker, or HTML files into your custom DTD, examining the DocBook MapTemplates will give you ideas on how to build your own MapObjects for common constructs such as tables, images, and inline elements.

The DocBook MapTemplates are stored in the following directory:

`<home>\STDTemplates`

The templates included in this directory are:

- **word2docbook.std** — This MapTemplate (`<home>\STDTemplates\DocBook\word2docbook.std`) is used to transform Microsoft Word 2003 (.doc) content into DocBook. The MapObjects in this template are configured to transform **SampleDocument.doc** (`Arbortext-path\samples\importexport\word\SampleDocument.doc`) into DocBook.
- **html2docbook.std** — This MapTemplate (`<home>\STDTemplates\DocBook\html2docbook.std`) is used to transform HTML (.htm) content into DocBook. The MapObjects in this template are configured to transform **articleWhyConvert.htm** (`Arbortext-path\samples\importexport\html\articleWhyConvert.htm`) into DocBook.
- **mif2docbook.std** — This MapTemplate (`<home>\STDTemplates\DocBook\mif2docbook.std`) is used to transform Adobe FrameMaker (.mif) content into DocBook. The MapObjects in this template are configured to transform **sampledocument.mif** (`Arbortext-path\samples\importexport\frame\sampledocument.mif`) into DocBook.
- **docx-to-axdocbook.std** — This MapTemplate (`<home>\STDTemplates\ArbortextSamples\docx-to-axdocbook.std`) is used to transform Microsoft Word .docx content into axdocbook.

- **pdf-to-asdocbook.std** — This MapTemplate (`<home>\STDTemplates\ArbortextSamples\pdf-to-asdocbook.std`) is used to transform Adobe PDF (.pdf) content into asdocbook.
- **wordml-to-axdocbook.std** — This MapTemplate (`<home>\STDTemplates\ArbortextSamples\wordml-to-axdocbook.std`) is used to transform WordML (.xml) content into axdocbook.
- **import2axdocbook.std** — This MapTemplate (`<home>\STDTemplates\ArbortextSamples\import2axdocbook.std`) is used to convert sample Microsoft Word (.doc) files into ppXML and then convert the ppXML elements to axdocbook.
- **importmif2axdocbook.std** — This MapTemplate (`<home>\STDTemplates\ArbortextSamples\importmif2axdocbook.std`) is used to convert a sample FrameMaker (.mif) file to ppXML and then convert the ppXML elements to axdocbook.

Each of these templates is configured to transform a given set of files into DocBook. To transform your own files, you will have to customize the templates.

This tutorial will guide you step-by-step through customizing the `word2docbook.std` template for the sample file `articleWhyConvertTutorial.doc` (`Arbortext-path\samples\importexport\word`).

How the Templates Work

The DocBook MapTemplate contains a preprocessing rule that transforms its source format (for example, .doc for Word files) into ppXML. Then, each DocBook MapTemplate has a set of MapObjects that transforms specific elements in the ppXML (PARAGRAPH, TABLE, etc.) into chapter titles, section titles, and other DocBook related elements for the output document.

To examine the Word-to-DocBook MapTemplate:

1. Open the Arbortext Import Workbench main window.
2. Select **File ►Open ►MapTemplate**.

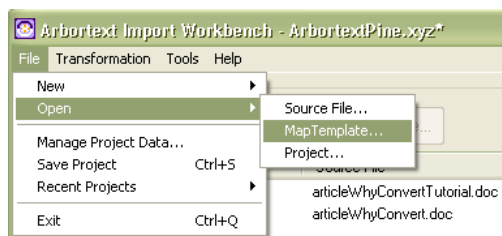


Figure - 6.1

3. Browse to and open the `word2docbook.std` MapTemplate. The **MapTemplate Editor** opens.

Figure - 6.3

 **Note**

Other similar templates, such as `html2docbook` or `mif2docbook`, have preprocessing drivers that transform the source file format into ppXML.

6. In the MapTemplate Editor, you can click the **MapTemplate Properties** tab, where you will see that the operation is set to **Use MapObject Priority** as shown in Figure 6.4.

To transform the ppXML into DocBook, select the **Use MapObject Priority** option. This option means that after the preprocessing driver produces ppXML, the MapObjects will run in order, starting from the top most element in the ppXML to the last one, for each that matches this source element. Then, for each child of this source element, the process is repeated. This process is continued recursively in the source document.

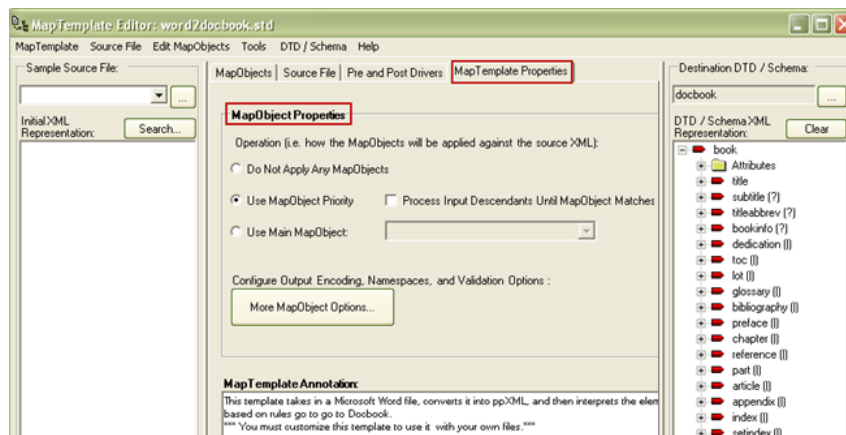


Figure - 6.4

7. The lower half of the **MapObjects** tab shows the **MapObject Details**. The **Input Rules** sub-tab can be selected to show the input rules. Whenever a MapObject is

selected in the MapObjects grid at the top of the **MapObjects** tab, the **Input Rules** tab shows the details of the input rules, as highlighted in Figure 6.5.

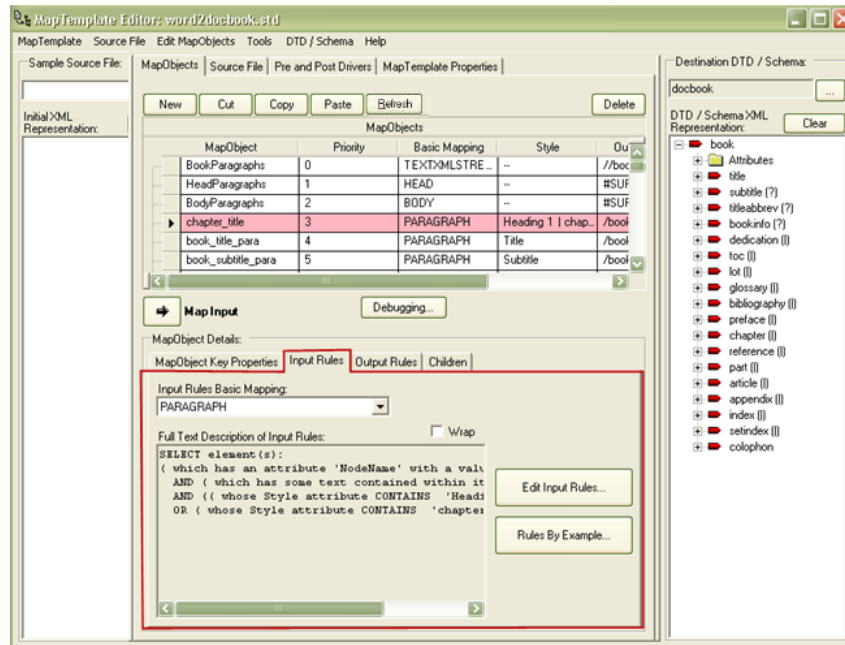


Figure - 6.5

Each MapObject has input rules, the rules that are to be followed to match ppXML elements of the source file to a particular MapObject. For example, the chapter_title MapObject has been set up with rules to find the chapter titles in the ppXML, as highlighted in Figure 6.5. Details regarding the input rules can be viewed in the *XML Input Rules* section of the *Arbortext Import Reference*.

For example, selecting the chapter_title MapObject in the word2docbook MapTemplate shows:

```
SELECT element(s) :
( which has an attribute 'NodeName' with a value of 'PARAGRAPH ' )
AND ( which has some text contained within it.)
AND (( whose Style attribute CONTAINS 'Heading 1')
OR ( whose Style attribute CONTAINS 'chapter title'))
```

This means that the only PARAGRAPH whose Style is “Heading 1” or “chapter title” in the ppXML(or in the ppXML representation of the source document) will match this MapObject.

- Each MapObject has output rules that specify the elements to be emitted into the output XML. In the case of chapter_title, a <chapter> element and a <title> element would be emitted as valid DocBook, as highlighted in Figure 6.6. To see the output

rules, select the **Output Rules** sub-tab in the **MapObject Details** area of the **MapObjects** tab.

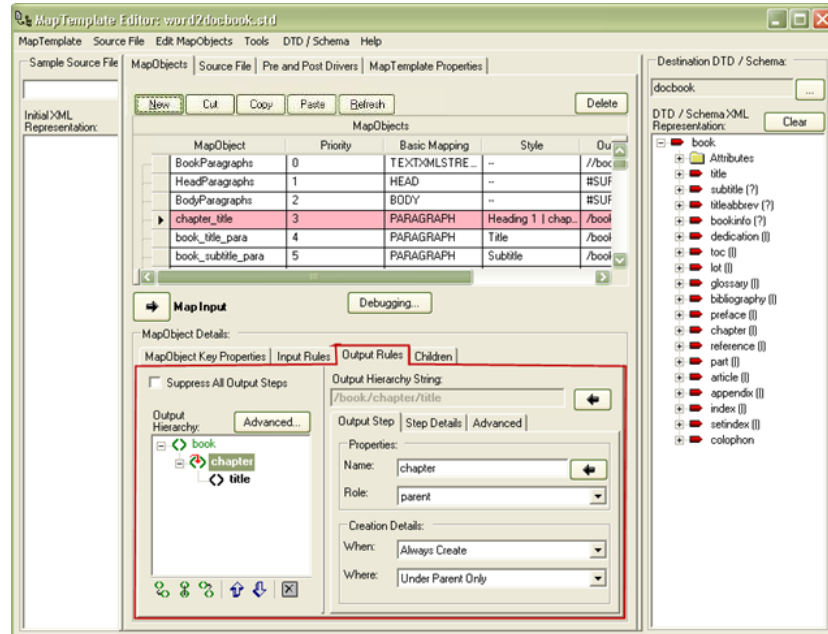


Figure - 6.6

The output rules for this MapObject create a hierarchy shown in the **Output Hierarchy** window of the **Output Rules** tab.

- If a paragraph matches the input rules, the MapObject outputs a <title> element based on the current PARAGRAPH.
- The <title> element is placed under the <chapter> element that is created before the title element (if one does not already exist).
- If Always Create item from the **When** list is selected, a new chapter is always created.
- The chapter element is always created underneath a <book> element.

Details regarding the **Output Rules** tab can be viewed in the *MapTemplate Editor* section of the *Arbortext Import Reference*.

How to Customize the DocBook Template

To customize the templates to work with other source files, you should save the template with a new name and change the input rules to match the appropriate elements in your source document. If the template is working fine, you probably won't need to modify the output rules. The steps to customize DocBook templates are:

1. Create a new template based on the existing template.

-
2. Customize the template according to your requirements.
 3. Run the template against the document.
 4. Customize the Top Level MapObjects for your source document including:
 - Bookinfo — book title, subtitle, and publisher name. This means customizing the `book_title` and `book_subtitle` MapObjects.
 5. Customize the Chapter and Section MapObjects
 - Chapters and parts — the `chapter_title` MapObject. (And if your book contains parts, then the `part_title` MapObject.)
 - Sections and subsections — the `sect1_title`, `sect2_title`, `sect3_title` MapObjects.
 6. Customize the List Objects.
 7. Customize the Table and Image MapObjects.
 8. Add or Customize other remaining MapObjects.
 9. Validate by rerunning the transform. Examine the output, refining the template if necessary.

The goal of this chapter is to create a template that will handle many documents. If writers have used similar constructs in all of the documents, then the goal is to create one template that handles all of these, meaning minimal work is needed once the output is created. Arbortext Import may not be able to get 100% styled DocBook content from a Word document if the creators of the Word documents did not use consistent practices when creating the Word documents.

Creating a New Template Based on Existing DocBook Template

To create a new template based on one of the DocBook templates, you should start with a new transformation in the Arbortext Import Workbench. Identify a sample source document you want to work with and customize the MapTemplate according to that document. In this tutorial you will use the following data:

- Source Document — `articleWhyConvertTutorial.doc` located at:
`Arbortext-path\samples\importexport\word.`
- Transformation template to match the source — Because this is a Word document, you will be modifying the `word2docbook.std` located at `:<home>\STDTemplates\DocBook.`

Use the following procedure to create a new MapTemplate based on an existing template:

1. From the main Arbortext Import Workbench window, click **New**. The **New Transformation wizard** will appear, and you will be prompted for the name of the source file.

-
2. Choose **articleWhyConvertTutorial.doc** from the source file available list. (Select **Find** if the file isn't in the list of source files that have already been added to the project.) Click **Next**.
 3. In the next window of the wizard, you will be asked for the MapTemplate you want to use. Choose **Custom / Other** and click **Next**.
 4. This will take you to the next **Select MapTemplate to Direct Transformation** window. This window contains a **Find** button that lets you find an existing MapTemplate and a **New** button that lets you create a new MapTemplate as explained in [Setting up a New MapTemplate on page 48](#).

For this example, select **New**.

5. The new template wizard is opened, and you will be prompted to specify MapTemplate name and location. The default location is: **<home>\STDTemplates**. For this example, create a subdirectory of STDTemplates called "tutorial". To do this:
 - a. Click **Browse** next to **MapTemplate Location**. Once you have created the new directory, its name will be shown under **MapTemplate Location**.
 - b. Provide the name of this new template in **MapTemplate Name**. Only use the first part of the file name, such as **article2docbook**, as the **.std** extension will automatically be appended to the name under **MapTemplate Location**.
6. Click **Next**.
7. The next window in the wizard is **Specify New or Cloned MapTemplate**. In this window, you can specify that the template should be based on (be a "clone of") an existing template.
 - Select the option **Clone Existing Template**.
 - Click **Find** to locate the existing template.
 - Select **word2docbook.std** located at: **<home>\STDTemplates\DocBook**.
8. Clicking **Finish** to end the **New MapTemplate wizard**. The **New Transformation Wizard** will be displayed with the new template name selected.
9. Click **Next**. You will be prompted to specify a nickname for this transformation. You can specify `CustomExample` or something descriptive.
10. Click **Next**. A wizard prompting you to run this new transformation will be shown. Do not run the transformation, as we want to customize this cloned template. Select **Do NOT Run Transformation When Complete**.

11. When you click **Finish**, the new transformation will be shown in the project main window, along with the source file: **articleWhyConvertTutorial.doc**, nickname: CustomExample, and the MapTemplate: **article2docbook**.

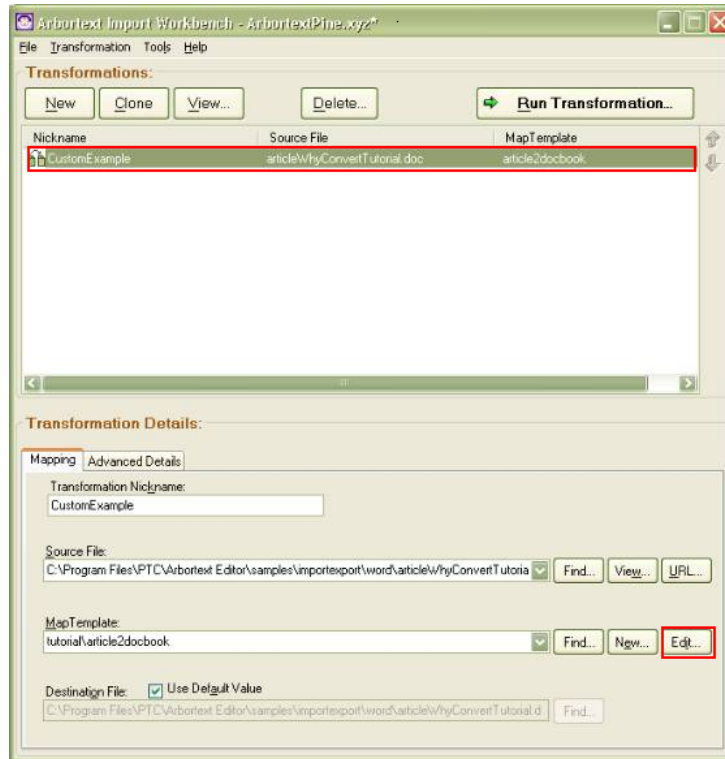


Figure - 6.7

12. Click **Edit** next to **MapTemplate**. The MapTemplate Editor is opened with a new template that is currently a complete clone of the word2docbook template, including its preprocessing drivers and MapObjects.

Customizing the Template

On the left side of the MapTemplate Editor there is a **Sample Source File** pane. There will not be any **Initial XML Representation** tree.

To load the ppXML initial representation, choose **articleWhyConvertTutorial.doc** from the file list. You can also browse to the source file location using browse (...). You will see the ppXML generated from this file. (This generation occurs only once. Subsequently it is available in the MapTemplate Editor).

On the right side of the MapTemplate Editor, you will see a **Destination DTD/Schema** pane. The DocBook DTD is selected, and a tree-like view of the DocBook DTD is shown. If this is not what you see, click DTD browse (...) and find the DocBook DTD on your local machine. It should be available in **Arbortext-path\doctype\docbook**. Arbortext Import will take a few seconds to transform it into an internal compiled format, and you will be prompted to choose main element for this DTD.

For this example, choose **book**. The MapTemplate window will appear as shown in Figure 6.8.

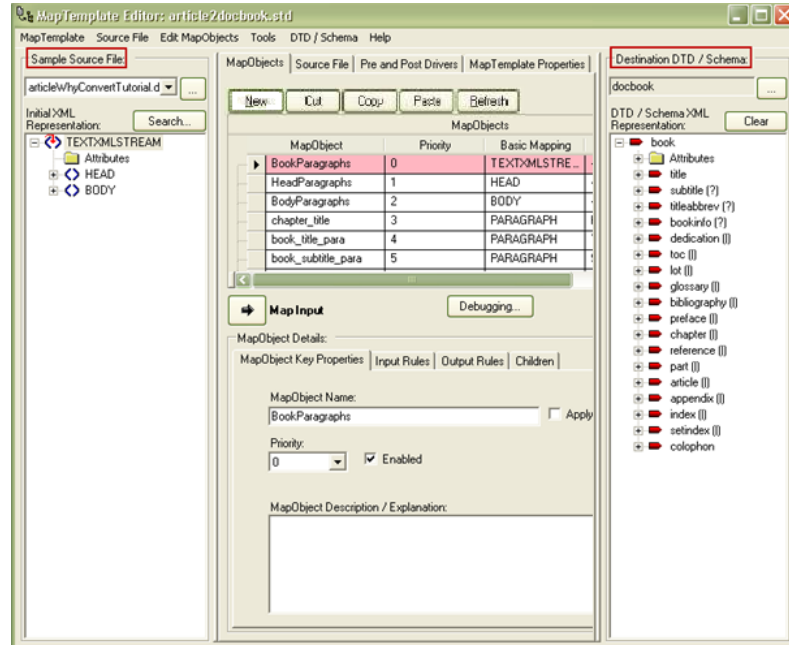


Figure - 6.8

In the middle main pane of the MapTemplate Editor, you will see multiple tabs. The **Source File** tab shows an HTML view of the ppXML for this document that is very similar to the original Word document.

To locate any text in the ppXML tree:

1. In the source file HTML view, select a part of any PARAGRAPH.
2. Click **Locate Selected Text** and the desired PARAGRAPH will be found in the source file tree in the left hand pane, as shown in Figure 6.9.

The tree can show you the style used, font formatting, and so on, which are all in the attributes of the PARAGRAPH. This will be important during the mapping process.

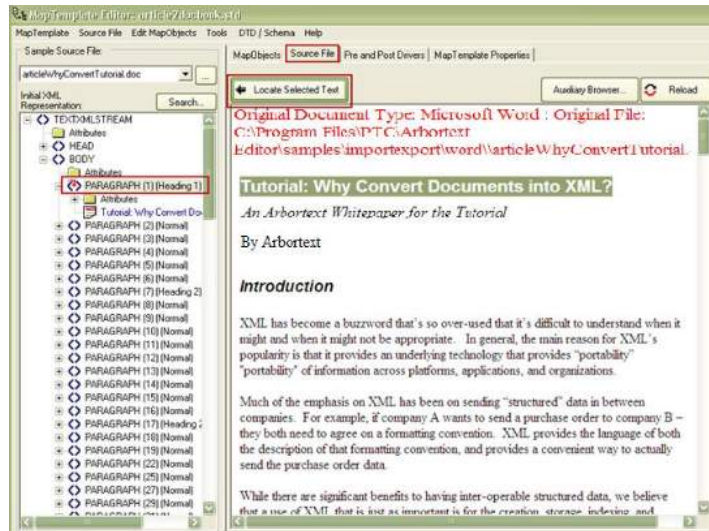


Figure - 6.9

The **Pre and Post drivers** tab contains the preprocessing rule that transforms the binary Word file (.doc file) into ppXML using various options.

To see preprocessing driver options, double-click the preprocessing driver. There are several important options that the DocBook templates use, and other templates may or may not use. For example:

- **Include Lists** — This option is set to false. This is because the DocBook template relies on the style and formatting properties of PARAGRAPHS, rather than relying on actual elements in the ppXML. A list in DocBook might include more than a simple Word list paragraph.
- **Use CALS Table Format** — This option is set to true. This causes the TABLE element in the ppXML to follow the CALS table, that makes it easy to map to DocBook (CALS table model: <http://www.oasis-open.org/specs/a502.htm>). This means that the element will have the descendants TGROUP, COLUMN, THEAD, and TBODY in addition to the regular descendants of ROW, CELL, and para. It means that *rowspan* and *colspan* attributes on CELL will follow the CALS model that DocBook uses.

Details of the driver options are given in the *Pre and Post Processor Drivers* section of the *Arbortext Import Reference*.

Run the Template Against the Document

Before you customize the rules to examine what this template does when run against **articleWhyConvertTutorial.doc**, save the template against the document. To do this, select **Save and Run** from the **MapTemplate** menu of the MapTemplate Editor. The **Save**

and Run command saves the current template and runs it against the currently selected source file. (In this case, **articleWhyConvertTutorial.doc**), bringing up the **Transformation** window, as shown in Figure 6.10. The **Transformation** window is explained in [Run a Transformation Object on page 23](#)

You will see that this window has an **Intermediate Results** tree on the left that contains a source file (**articleWhyConvertTutorial.doc**) with two XML files underneath it.

- The first of these is the ppXML file. If you click on this file, the ppXML id displayed in the **Intermediate Results Display** tab.
- The second file is the output XML after applying the MapObjects (**articleWhyConvertTutorial.doc.xml** in the destination directory).

If you examine the XML file created where the source file is placed, you will see:

- The top level element is <book>.
- There is no <title> and <subtitle> for the book.
- There is only one chapter, with a title of “Tutorial: Why Convert Documents into XML”.
- There are a number of <para> elements and <sect1> element in the chapter.

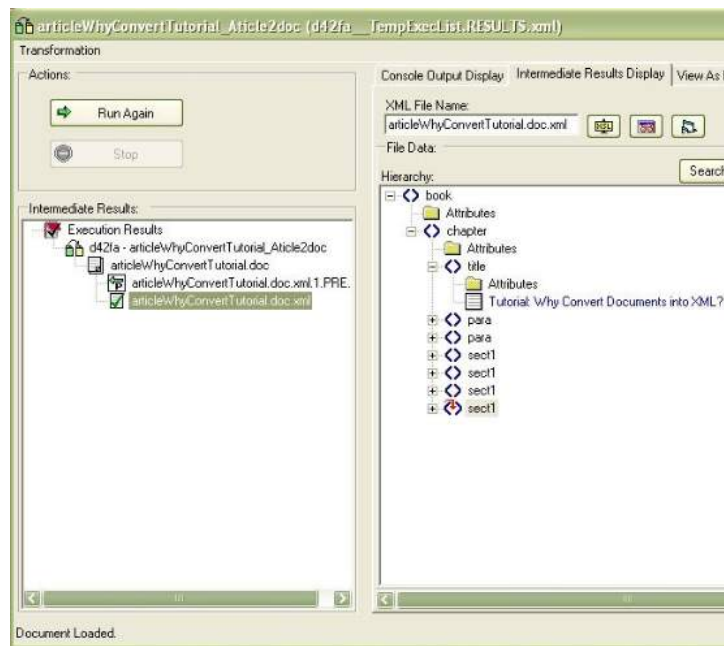


Figure - 6.10

Let us assume that we want a book element that contains a “title”, “subtitle” and multiple chapters; each section of this chapter can become its own chapter. This means that you will have to change the input rules for each of the relevant MapObjects. In general, you

do not have to change many of the output rules as they are already set up to produce valid DocBook.

Customizing the Top Level MapObjects

Now it is time to start customizing the MapObjects. To customize the top level MapObjects:

1. Open the MapTemplate Editor.
2. Select the first MapObject BookParagraphs from the **MapObjects** middle pane that has basic mapping type of TEXTXMLSTREAM.
3. Select the top-most element TEXTXMLSTREAM in a ppXML document from the left pane.

When you select the BookParagraphs MapObject, Arbortext Import will fill in the details. Examine the **Input Rules** sub-tab in the **MapObjects** tab. The **Input Rules** tab should say:

```
SELECT element(s) :  
whose name equals 'TEXTXMLSTREAM'
```

This means that this MapObject will always select TEXTXMLSTREAM. The output rules shown in the **Output Rules** tab under the **MapObjects** tab for this particular MapObject show that it only outputs a “core” element name of “book”. This is because; it is the top-level element in DocBook.

There is no need to change this element. However, this source file (**articleWhyConvertTutorial.doc**) contains a title; which is currently being treated as a chapter title, rather than a book title. This means that the PARAGRAPH with the title “Tutorial: Why Convert Documents into XML?” has a style attribute of “Heading 1”, as shown in the following Figure 6.11. If you look at Figure 6.10 then you will see

that there is no tag for book title as the style attributes are matching the chapter_title MapObject rather than the book_title_para MapObject.

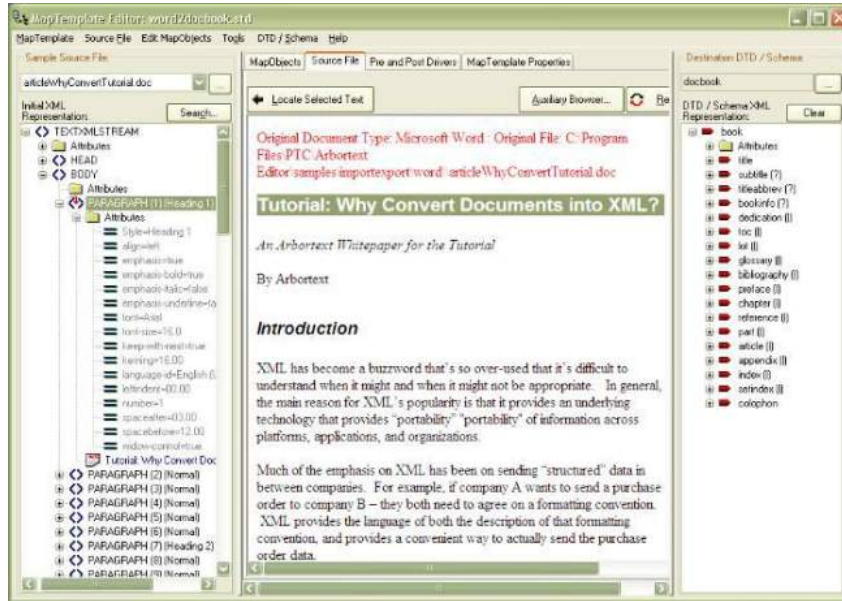


Figure - 6.11

To locate the chapter title PARAGRAPH element:

1. Select a portion of the title in the HTML view of the **Source File** tab.
2. Click **Locate Selected Text**.
3. Select the **MapObject** tab.
4. Choose the chapter_title MapObject.

If you examine the input rules for chapter_title MapObject, you will see the following:

```
SELECT element(s) :
( which has an attribute 'NodeName' with a value of 'PARAGRAPH ' )
AND ( which has some text contained within it.)
AND (( whose Style attribute CONTAINS 'Heading 1')
OR ( whose Style attribute CONTAINS 'chapter title'))
```

If you examine the input rule for the book_title_para MapObject, you will see the following:

```
SELECT element(s) :
( whose name equals 'PARAGRAPH ' )
AND ( which has some text contained within it.)
AND (( whose Style attribute CONTAINS 'Title')
AND ( NOT ( whose Style attribute CONTAINS 'Subtitle'))))
```

Customizing book_title MapObject:

To output the first title as a book title, the input rules for the book_title_para must be changed. Also, the order of the MapObjects should be reversed so the book_title MapObject runs against the PARAGRAPH first, and the chapter_title MapObject only runs on a PARAGRAPH if it does not match book_title.

You can easily find your required MapObject by arranging MapObjects in alphabetical order. To do this, select **MapObject** from the MapObjects grid's header on the **MapObjects** tab.

To customize the book_title MapObject:

1. Select the first PARAGRAPH in the source tree in the sample **Source File** pane.
2. Select the book_title_para MapObject in the MapObjects grid.
3. Click **Map Input** under the MapObject grid. This will open the **Map Input Wizard** to guide you through the process of updating rules for the currently selected MapObject. The **Map Input Wizard** is explained in [Creating a new MapObject on page 58](#).

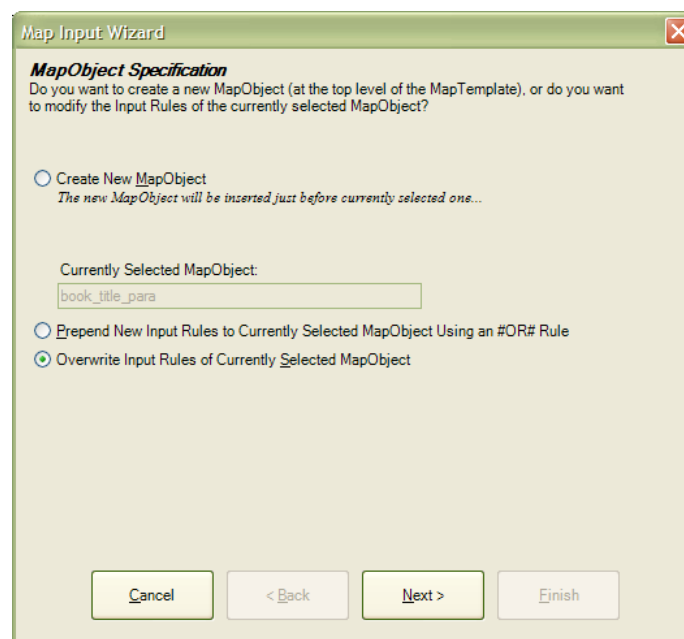


Figure - 6.12

4. For the input wizard, you can select the following for each window:
 - **MapObject Specification** — Rather than creating a new MapObject, you want to update the input rules of the currently selected MapObject (book_title_para).
 - a. Select **Overwrite Input Rules of Currently Selected MapObject** option.
 - b. Click **Next**.

- **Basic Input Mapping**
 - a. Ensure that PARAGRAPH is set here.
 - b. Click **Next**.
- **MapObject Style Mapping** — This will ask you if you want to set particular style.
 - a. In this case, the style selected will be “Heading 1”. Ensure that `Heading` is selected.
 - b. Click **Next**.
- **MapObject Attribute Mappings** — In this case, the style is the main distinguishing characteristic.

As no other PARAGRAPH in this document uses that style, you don’t need any other input rule. Click **Finish** on this window.

The following input rules should be displayed for the `book_title` MapObject:

```
SELECT element(s) :
(whose name equals 'PARAGRAPH ')
AND (whose Style attribute EQUALS 'Heading 1')
```

Before running the transformation, change the order of `book_title_para` so it comes before “chapter_title”. To change the order:

1. Select the **MapObject Key Properties** tab in the **MapObject Details** area of the **MapObjects** middle pane.
2. Using **Priority**, change the value from 4 to 3. Notice that the rest of the numbers are automatically resorted.

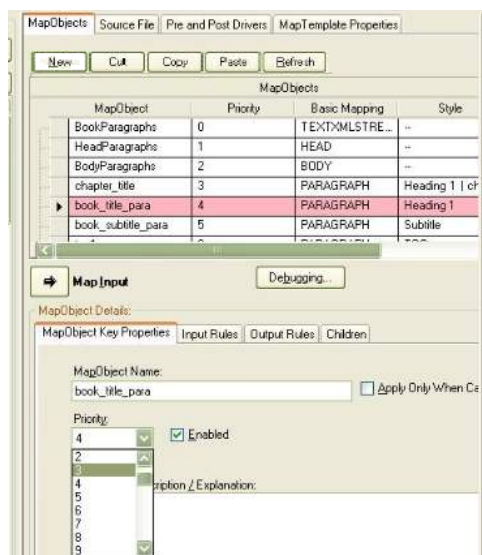


Figure - 6.13

After changing the order, test your changes by running the transformation again. For this, click **MapTemplate ► Save and Run**. In the DocBook XML, you'll see a <book> with a <bookinfo> element, a <title> element underneath the <bookinfo>, and some <chapter> elements after the <bookinfo> element. You will also see some additional <para> elements in the <bookinfo> tag under the <title>.

Customizing book_subtitle_para MapObject:

Now you will work the subtitle PARAGRAPH text “An Arbortext Whitepaper for the Tutorial”. To customize the subtitle:

1. Find the text “An Arbortext Whitepaper for the Tutorial” in the HTML view of the source file.
2. Click **Locate Selected Text**.
3. Go to the book_subtitle_para MapObject on the MapObjects grid as shown in Figure 6.14.
4. Run the **Map Input Wizard** by clicking **Map Input**.

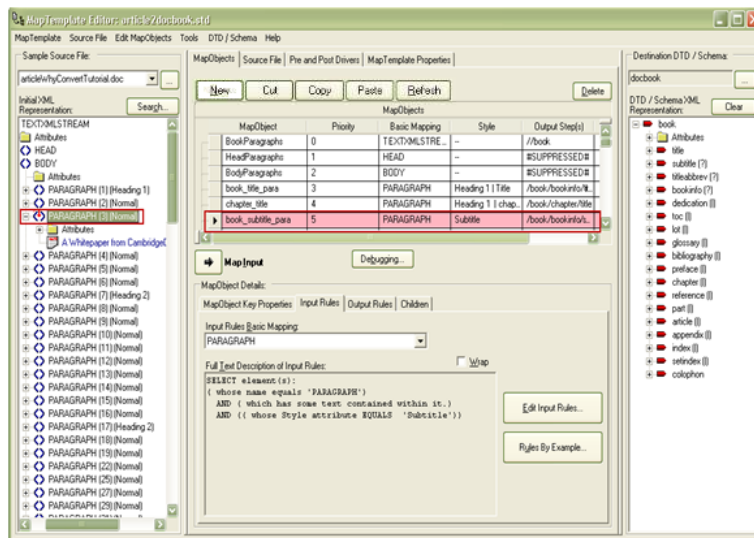


Figure - 6.14

5. Perform the following steps on each window of the **Map Input Wizard**:
 - **MapObject Specification** — Once again, rather than creating a new MapObject, update the input rules of the currently selected MapObject (book_subtitle_para).
 - a. Select **Overwrite Input Rules of Currently Selected MapObject** option.
 - b. Click **Next**.
 - **Basic Input Mapping**
 - a. Ensure that PARAGRAPH is set here.

-
- b. Click Next.**
 - **MapObject Style Mapping** — This will ask you if you want to set any particular style.
 - a.** In this case, the style selected will be `Normal`. Ensure `Normal` is selected.
 - b. Click Next.**
 - **MapObject Attribute Mappings** — In this case, you will see that the distinguishing characteristic of the subtitle paragraph is that it is italicized. This is indicated by the `emphasis-italics`, whose value is `true`.
 - a.** Select `emphasis-italics` from the list.
 - b.** Select the **MapObject Should Track This Attribute** option.
 - c.** Change **Type of check** to `IS_TRUE`. Doing so checks that the value of the attribute is `true`.
 - d. Click Next.**
 - You will see the **MapObject Text Verification Editor**. We are not relying on any of the text inside the subtitle `PARAGRAPH` for identification. Click **Next**.
 - You will see the **MapObject Input Hierarchy Refinement** window. This window is used to view the hierarchy of the Input document (ppXML), and in a special case, enables you to specify what ancestors must exist in the output document (in this case, the DocBook) at the current insertion point for this MapObject.

This is important because the input rules for the subtitle are: Style should be “normal” and the `PARAGRAPH` must be italicized. It is certainly possible that there can be another paragraph somewhere else in the source document that is italicized and `Normal` (although not true in this document). Therefore, you will apply these rules only if you are at the subtitle. How can you determine this?

One way is:

- a. The subtitle occurs after the title. The title will be under bookinfo in the output. In the **Only match if Output XML has Specified Ancestry** box, type in bookinfo, as shown in Figure 6.15.

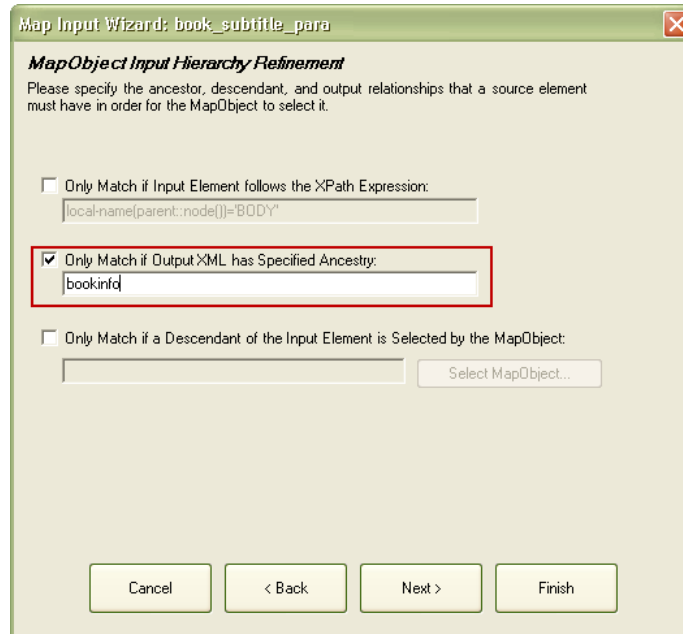


Figure - 6.15

- b. Click **Next**.

- The next window is **MapObject Advanced Rule Editor**. Here, you will see the XML input rules hierarchy. These additional qualifier rules are required only for exceptional cases. Click **Finish** without making any change.

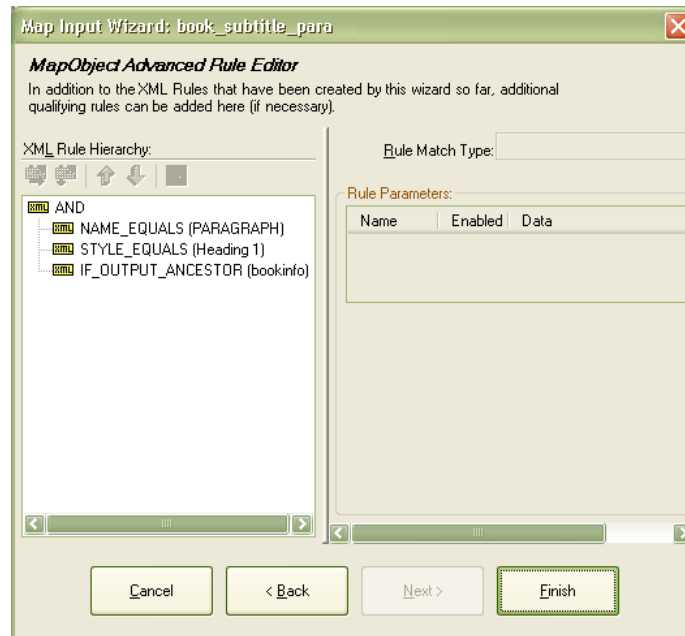


Figure - 6.16

You will see the following input rules for the book_subtitle_para:

```
SELECT element(s) :
( whose name equals 'PARAGRAPH ' )
AND ( whose Style attribute EQUALS 'Normal' )
AND ( STDCurrent.emphasis-italic IS_TRUE . )
AND ( . IF_OUTPUT_ANCESTOR STDConstant.bookinfo )
```

6. Now select **MapTemplate ► Save and Run** to test your changes. You will now see in your DocBook XML two <subtitle> elements under <bookinfo> element, the first having text “An Arbortext Whitepaper for the Tutorial” and the second being empty. You will also see one additional <para> element under <bookinfo>, after the two subtitles and before the <chapter> element.

You must now decide what to do with the empty subtitle and the additional para.

To get rid from these two paras from the XML you will have to create two new MapObjects.

- blank_para_bookinfo MapObject
- publishername MapObject

Creating blank_para_bookinfo MapObject:


Use the following procedure to create a new MapObject to be used for a blank subtitle appearing inside <bookinfo>.

1. Select any empty paragraph in the source document's ppXML tree on the left. PARAGRAPH (4) is used in this example.
2. Click **Map Input**.
3. On the **Map Input Wizard** using the following steps:
 - **MapObject Specification**
 - a. Choose **Create New MapObject** (the default).
 - b. Click **Next**.
 - **MapObject Properties**
 - a. Enter name, such as "blank_para_bookinfo".
 - b. Click **Next**.
 - **Basic Input Mapping**
 - a. This should be PARAGRAPH, that is the default.
 - b. Click **Next**.
 - **MapObject Style Mapping**
 - a. On this window, select **Don't Look for a Particular Style**.
 - b. Click **Next**.
 - **MapObject Attribute Mappings**

On this window, you can simply click **Next** without selecting any attributes.
 - **MapObject Input Hierarchy Mapping**
 - a. On this window, select **Only Match If Output XML has Specified Ancestry**.
 - b. In this text box, type in bookinfo, since this MapObject will only apply under bookinfo element.
 - c. Click **Next**.
 - **MapObject Advanced Rules Editor** — On this window, you will see the input rules that have been created so far. These are the **AND** rules with three children: **NAME_EQUALS**, **STYLE_EQUALS** and **IF_OUTPUT_ANCESTOR (bookinfo)**. These input rules will select any PARAGRAPH which is being processed when the insertion point in the Output XML is inside bookinfo.

We want this MapObject only to work on "blank" paragraphs.

 - a. Choose **AND** in the **XML Rule Hierarchy**.

- b. Click **Add Child XML Rule** .
 - c. Now change the **Rule Match Type** list to NOT.
 - d. Again click **Add Child XML Rule** to add a child rule to the **NOT**.
 - e. Change the matchtype to HASTEXT.

It is possible for a PARAGRAPH not to have any text, but still have child elements, such as IMAGE. (Although this particular source document doesn't have this scenario.)
 - f. Add another child of **AND**.
 - g. Change its matchtype to NOT.
 - h. Add a child to **NOT** called HASCHILDELEMENTS.
- Click **Finish**.

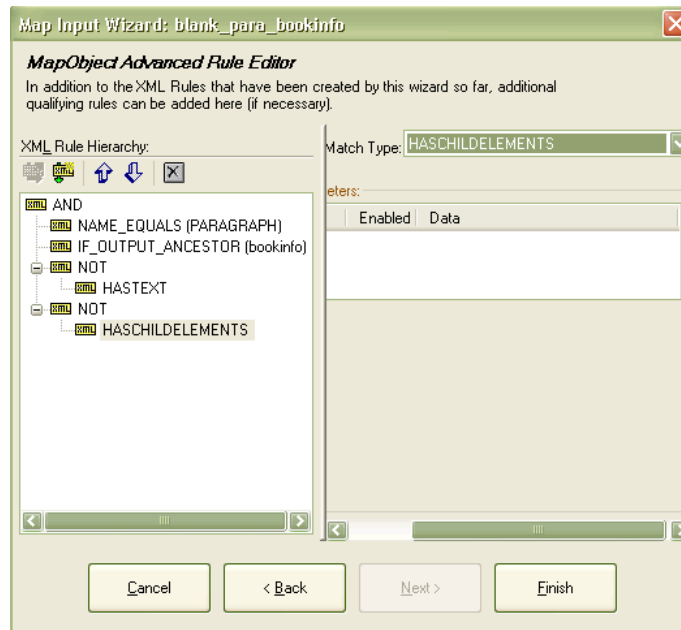


Figure - 6.17

When you click **Finish** in the **Map Input wizard**, you will see the following input rules. If your input rules do not match, you can click **Edit Input Rules**, and the input rules editor will open where you can modify the input rules.

```
SELECT element(s) :
( whose name equals 'PARAGRAPH ' )
AND ( . IF_OUTPUT_ANCESTOR STDConstant.bookinfo)
AND ( NOT ( which has some text contained within it.))
AND ( NOT ( . HASCHILDELEMENTS .))
```

4. Set its priority higher than the book_subtitle_para MapObject using the **MapObject Key Properties** tab.
5. If you run the transformation again by selecting **MapTemplate ► Save and Run**, will see that the blank subtitle inside the bookinfo is now the element blank_para_bookinfo, and there are two more blank_para_bookinfo elements under <bookinfo>. These two are created for the other two empty paragraphs in the source document, one before the text “By Arbortext” and the other after it.
6. To change what is output by a rule, we will need to change the output rules so the blank_para_bookinfo elements do not appear in output.
 - a. Select the blank_para_bookinfo.
 - b. Navigate to its **Output Rules** tab.
 - c. Select **Suppress All Output Steps** option located above the **Output Hierarchy**.
 - d. Go to the **Children** tab.
 - e. Select **Ignore All** under the **Child Text/CDATA Nodes**.

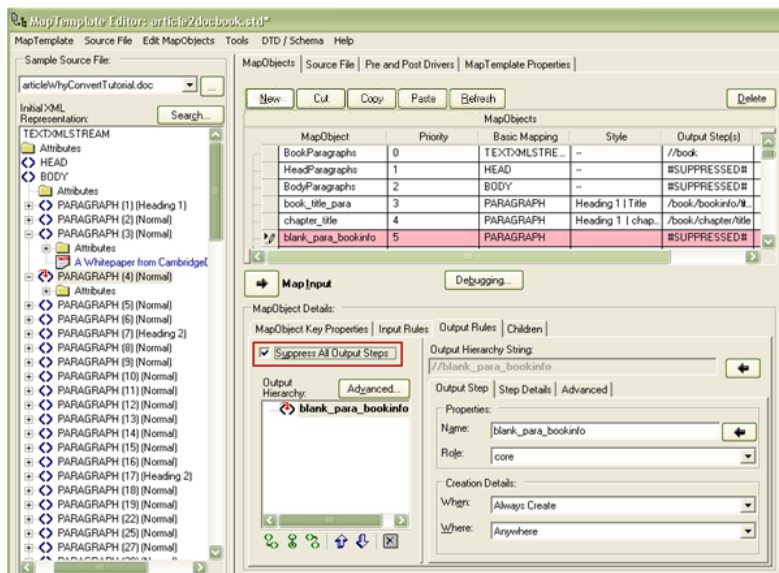


Figure - 6.18

7. To confirm that the blank_para_bookinfo is removed from the output, select **MapTemplate ► Save and Run**.

Creating publishername MapObject:

According to the DocBook DTD, there should not be any para in the bookinfo. However, here you will notice that there is still one para in <bookinfo>. You can get rid of it by creating a new MapObject that is called publishername.

To create publishername MapObject:

-
1. Select the text “By Arbortext” in the **Source File** tab.
 2. Click **Locate Selected Text**. The paragraph containing “By Arbortext” is highlighted in the ppXML representation tree.
 3. Click **Map Input**.
 4. On the **Map Input Wizard**, perform the following steps:
 - **MapObject Specification**
 - a. Select **Create New MapObject**.
 - b. Click **Next**.
 - **MapObject Properties**
 - a. Enter a name, such as `publishername`. “publishername” lowercase is a valid DocBook tag. By choosing the right MapObject name the correct output element can be chosen. If you enter any other name, then you will need to change the output rules to output the `<publishername>` element.
 - b. Click **Next**.
 - **Basic Input Mapping**
 - a. Ensure that `PARAGRAPH` is set here.
 - b. Click **Next**.
 - **MapObject Style Mapping** — This will ask you if you want to set any particular style.
 - a. In this case, the style selected will be `Normal`. Ensure `Normal` is selected.
 - b. Click **Next**.
 - **MapObject Attribute Mappings**

Click **Next**.
 - **MapObject Text Verification Editor** — Here, we are relying on the text in `publishername PARAGRAPH` for identification.
 - a. On this window, select the **Element Text** option.
 - b. Select `CONTAINS` from the list.
 - c. Select the **Text** option.
 - d. Type in `By Arbortext`.
 - e. Click **Next**.
 - **MapObject Input Hierarchy Refinement**

Here, in the **Only match if Output XML has Specified Ancestry** box, type in `bookinfo`.

- Click **Finish**.
5. To see that there is no more <para> element in the output, select **MapTemplate ► Save and Run**.

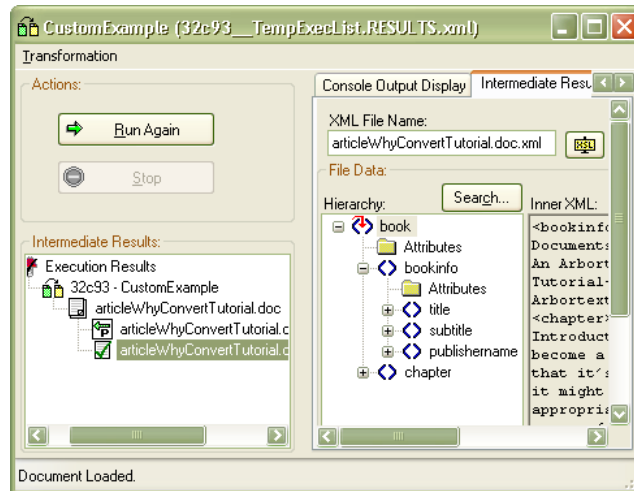


Figure - 6.19

Customizing the Chapter and Section MapObjects

Customizing chapter_title MapObject:

The next important thing is getting the chapter headings correctly for the document that contains one chapter with a few sections. However, to keep the perspective of working with a book, each of the headings should actually be its own chapter.

To do this, we have to modify the rules for the chapter_title MapObject.

1. Find a PARAGRAPH in the ppXML on the left that has one of the headings, such as the paragraph that says “Introduction”.
 - a. Find the chapter title para in the HTML view of the source file.
 - b. Click **Locate Selected Text** to select the PARAGRAPH in the ppXML file tree on the left.

You will notice by opening this PARAGRAPH and looking at its attributes that the Style attribute is “Heading 2”, that is how chapter titles will be distinguished. You can also distinguish them based upon the formatting. You will notice that the font-size is 14, and both emphasis-bold and emphasis-italic attributes are “true” for chapter headings.

2. Find the “chapter_title” MapObject in the main **MapObjects** grid, select it.
3. With this MapObject selected, and one of the chapter heading PARAGRAPH’s on the left selected, click **Map Input** that will open the **Map Input Wizard**.

-
4. The wizard will guide you through setting up the input rules for this MapObject. The following steps should be performed:
- **MapObject Specification:**
 - Choose **Overwrite Input Rules of Currently Selected MapObject** (chapter_title: if chapter_title is not the MapObject selected, cancel the wizard, find it, and re-run “MapInput”).
 - Click **Next**.
 - **Basic Input Mapping:**
 - This should be a PARAGRAPH, which is set by default.
 - Click **Next**.
 - **MapObject Style Mapping:**
 - On this window, it should automatically have the style attribute selected **Only Map Content with the Following Style**, and Heading 2 should also be selected.
 - Click **Next**.
 - **MapObject Attribute Mappings:** On this window, you can select individual attributes which will be used in addition to the style. In our case, as the “Style” is enough to distinguish chapter title paragraphs, it is instructive to select a few of these:
 - Choose **font-size**, and select **MapObject Should Track This Attribute**. This means an input rule will be created for font-size.
 - On the **Type of Check**, you can choose STRING EQUALS and the current value of font-size (14.0) will be filled. This means that the only PARAGRAPHS that have font-size 14 will match the criteria.
 - Now choose **emphasis-bold**, and select **MapObject Should Track This Attribute**.
 - For **Type of Check**, choose IS_TRUE. This input rule means that only PARAGRAPH’s that have emphasis-bold=”true” will match the criteria.
 - Do the same for **emphasis-italic**.
 - When you have tracked these three attributes, you can click **Finish**.

The input rules for the “chapter_title” MapObject will be modified based upon your selections. On the **Input Rules** tab you should see an English description of:

```
SELECT element(s) :
( whose name equals 'PARAGRAPH ' )
AND ( whose Style attribute EQUALS 'Heading 2' )
AND ( STDCurrent.emphasis-bold IS_TRUE . )
AND ( STDCurrent.emphasis-italic IS_TRUE . )
AND ( which has an attribute 'font-size' with a value of '14.0' )
```

- Now, test the new chapter title rules by selecting **MapTemplate ► Save and Run**.

You will now see four chapters appeared in the XML hierarchy: one for “Introduction”, second for “Some of SampleCompany Products”, third for “Why Create/Convert Documents to XML?” and fourth for “Conclusion”.

To learn how to edit input rules, perform the following steps.

If you want to remove any input rule:

- Click **Edit Input Rules**, and a widow displaying XML Rules similar to those shown in Figure 6.20 will open.

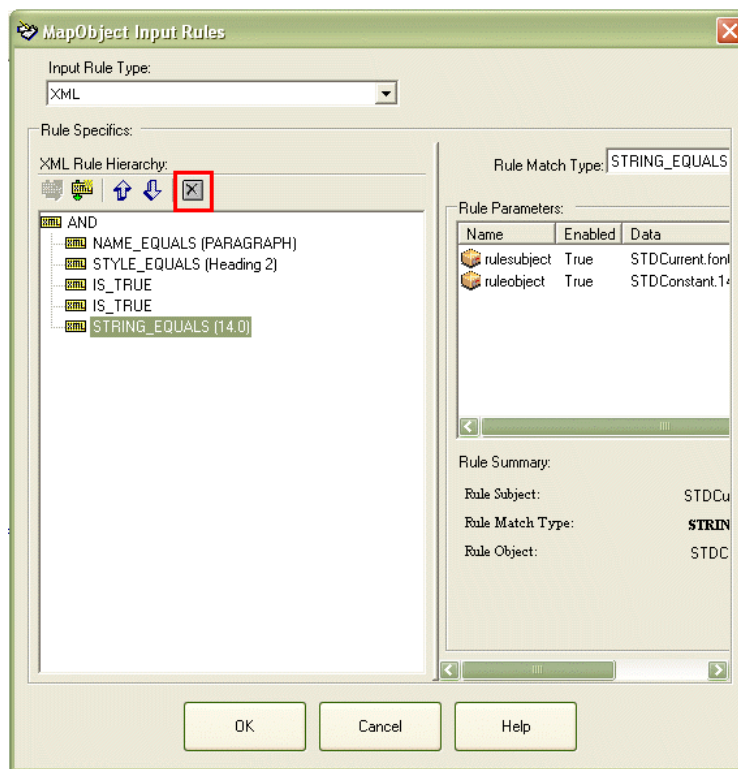


Figure - 6.20

- Suppose you want to remove input rule related to font-size then you will select the `STRING_EQUALS` rule
- Click **Delete** above the rules hierarchy, and this rule will be removed.
- Click **OK**, the resulting input rules English description will be as below:

```
SELECT element(s) :
( whose name equals 'PARAGRAPH ' )
AND ( whose Style attribute EQUALS 'Heading 2' )
AND ( STDCurrent.emphasis-bold IS_TRUE . )
AND ( STDCurrent.emphasis-italic IS_TRUE . )
```

This will be a good example of how to modify input rules once they are created.

Now examine the output rules for the `chapter_title` MapObject. You will see in the output hierarchy string as `/book/chapter/title`. These are shown in the **Output Hierarchy** tree.

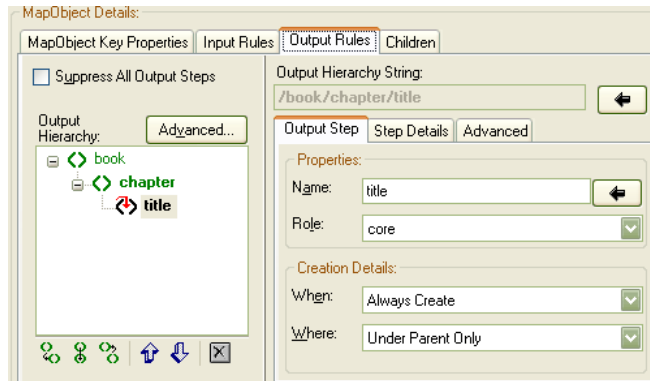


Figure - 6.21

If you select any element, you will see its details:

- **title** — This is the “core” output step. This means that the content of the PARAGRAPH will end up inside the “title” element. It has a creation type setting of “always” and a creation location of “under parent only”, which means that every time a PARAGRAPH is found that matches our input rules, then a “title” element will be created in our output XML, and it will always be created underneath its “parent”, which in this tree is “chapter”.
- **chapter** — This is the “parent” output step. It has a creation type of `Always Create`, which means that every time a matching PARAGRAPH is found, Arbortext Import will create a `<chapter>` element in the output XML, and it will always create it under the parent output step of this one (which is `book`).

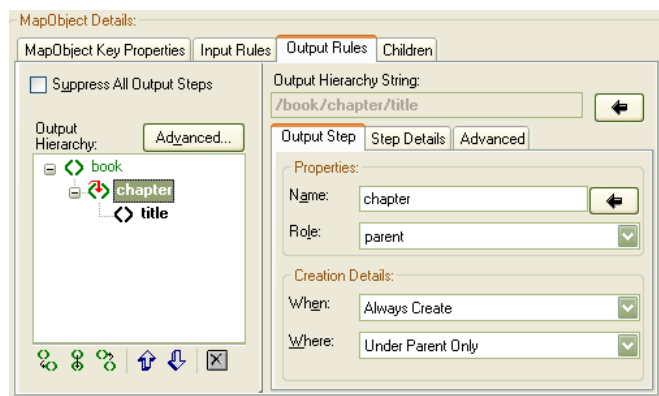


Figure - 6.22

- **book** — this is the “ancestor” output step. It has a **When** setting of `Create Only If Necessary`, and “Where” setting of “Top of Document” which means that a `<book>` element will only be created if there is not one in the output XML document. If there is already one, then Arbortext Import will move the insertion

point to just under <book>, and will insert “<chapter>” element underneath <book>, and then create <title> element underneath “chapter”.

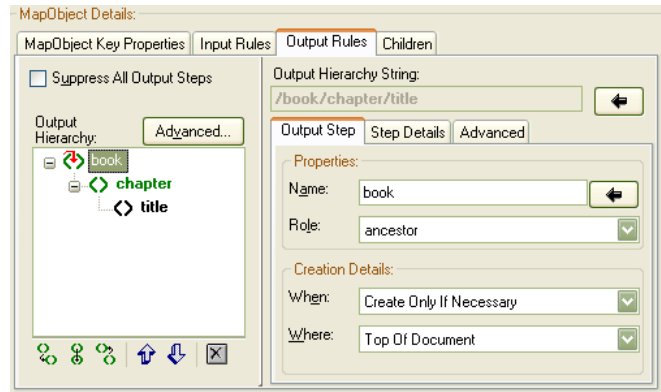


Figure - 6.23

You do not need to change either the **Output Hierarchy** of the chapter_title, or the output rules of any of the existing DocBook MapObjects, since they have been created to conform to the DocBook XML DTD.

However, if you were creating a MapTemplate to go to your own custom DTD, then you would need to create output rules that are consistent to that DTD.

Customizing sect1_title MapObject:

You will notice that in source document inside the third chapter there are some extra headings, that use the “Heading 3” style, and are meant to be sections within the chapters. You can find the sect1_title MapObject, and you will notice that its initial input rules set to:

```
SELECT element(s) :
( which has an attribute 'nodeName' with a value of 'PARAGRAPH ')
AND ( which has some text contained within it.)
AND (( whose Style attribute CONTAINS 'Heading 2')
OR ( whose Style attribute CONTAINS 'sect1 title'))
```

Find the “subsection” titles (“Initial Reasons”, “Subsequent Reasons” and “Final Reasons”) in the source document, and run the Map Input wizard, so the sect1_title MapObject can meet the following input rules:

```
SELECT element(s) :
( whose name equals 'PARAGRAPH ')
AND ( whose Style attribute EQUALS 'Heading 3')
```

To do this:

1. Select the subsection title PARAGRAPH in the source tree in the **Sample Source File** pane.
2. Select the sect1_title MapObject in the **MapObjects** tab.

-
3. Click **Map Input**, that will bring up the **Map Input Wizard**.
 4. The following steps should be done on each window of the **Map Input Wizard**:
 - **MapObject Specification**: Once again, rather than creating a new MapObject, we want to update the input rules of the currently selected MapObject, that is sect1_title.
 - Select option **Overwrite Input Rules of Currently Selected MapObject**.
 - Click **Next**.
 - **Basic Input Mapping**:
 - Ensure that “PARAGRAPH” is set here.
 - Click **Next**.
 - **MapObject Style Mapping**: This will ask you if you want to set any particular style.
 - In this case, the style selected will be “Heading 3”, so make sure that this is selected.
 - Click **Next**.
 - **MapObject Attribute Mappings**: In this case, the style is the main distinguishing characteristic, as no other PARAGRAPHS in this document use that style. We do not need any other input rule. You can click **Finish** on this window of the wizard.
 5. Also, because we do not plan to have any sub-subsections in this document, we can “disable” the sect2_title MapObject.

To disable any MapObject:

 - Select the MapObject.
 - On its **Key Properties** tab, deselect the **Enabled** option.
 6. If you **Save and Run** the template again, it will now transform <sect1> elements within the third <chapter>, each of which has a <title> element and content within it.

Similarly, to enable and customize sections within sections (sect2) and (sect3), the same procedure can be followed within the DocBook framework.

Customizing the List Objects

The next step in customizing the DocBook template is to customize the rules for lists. By examining the driver options of word2docbook template, you will see that **Include Lists** is set to `False`. This is because the DocBook templates look at the formatting of a Word paragraph to determine, should it be a list paragraph or not, and then the corresponding output XML is formed. In ppXML, if **Include Lists** is set to `False` for a Word

document, then PARAGRAPHS which are Word lists paragraphs, will have a label attribute distinguishing them as lists.

Lists may be nested, and the default DocBook templates have the following list related objects.

- **bullet_list_item1** — this MapObject finds the beginning of a first level list. If you examine its input rules, you will see that it uses the style, the left indent and the label attribute.
- **bullet_list_item2** — this MapObject finds the beginning of a second level nested list. The input rules also use a combination of styles, left indent and the label attribute.
- **bullet_list_item3** — this MapObject finds the beginning of a third level nested list.
- **list1_continue** — this MapObject is used to distinguish those paragraphs which are indented and should be part of a first level list item, but are not list paragraphs themselves in Microsoft Word.

Using this basic set of objects as starting point, you can build most types of list structures easily. In the document we are using (**articleWhyConvertTutorial.doc**), there are “numbered lists”.

When you run the standard DocBook template against this document, this list comes under the `<itemizedlist>` element, that is not correct.

Before customizing the input rules for this section the following key terms are to be explained:

- **rulesubject**: The rulesubject is the value that will be compared.
- **ruleobject**: The ruleobject is what we use for comparison. It is often a constant value. The ruleobject specifies the target MapObjects to apply.

For example:

- **Rulesubject**: STDCurrent.font-size
- **Rule Match Type**: GREATER_THAN
- **Ruleobject**: STDConstant.15

In this example, the rule MATCHTYPE is GREATER_THAN, the rulesubject is STDCurrent.font-size and the ruleobject is STDConstant.15.

- **STDCurrent** — STDCurrent is a prefix that can be used in either a rulesubject or ruleobject. It means that the rulesubject or ruleobject refers to the current element. For example, STDCurrent.font refers to the font-attribute of the current element
- **STDConstant** — this prefix can only be used in a ruleobject, and it is used to enter a constant value as the suffix. It is used often in comparison rules.

Let's modify the input rules for first level, second level and third level nested lists to output "ordered lists". The way to distinguish an ordered list is to either look inside the label attribute, or to use the type attribute on a list <PARAGRAPH>.

The type attribute is "UL" for an unordered list (bulleted list) and "OL" for a numbered list. The type attribute is either on the LIST element in ppXML, or on the <PARAGRAPH> element if **Include Lists** is set to "False", as it is in this case.

To customize the list MapObjects:

1. Clone `bullet_list_item1`, `bullet_list_item2` and `bullet_list_item3`, by using **Copy** and **Paste** to make clones of the MapObjects.
2. Using the **MapObject Key properties** tab change the name of all three copied MapObjects as `ordered_list_level1`, `ordered_list_level2` and `ordered_list_level3` respectively.
3. Modify the input rules for these three copied MapObjects. For this purpose:
 - Select the MapObject in the **MapObject** grid.
 - Select its **Input Rules** tab.
 - Click **Edit Input Rules**.
 - The **MapObject Input Rules** window opens. Here, add a new rule with the following setting:
 - **Rules Match Type:** `STRING_EQUALS`.
To specify the rulesubject, select **rulesubject** from the **Rule Parameters** section and on **Parameters Details for rulesubject** section select following:
 - **rulesubject type:** `STDCurrent`
 - **rulesubject value:** `SourceAttribute`
 - Enter **Source Attribute** as type.To specify the ruleobject, select **ruleobject** from the **Rule Parameters** section and on **Parameters Details for ruleobject** section select following:
 - **ruleobject type:** `STDConstant`
 - **ruleobject value:** `$enter_text$`
 - Enter **String Value** as OL.

This should distinguish the orderedlist paragraphs from the normal itemizedlist paragraphs.

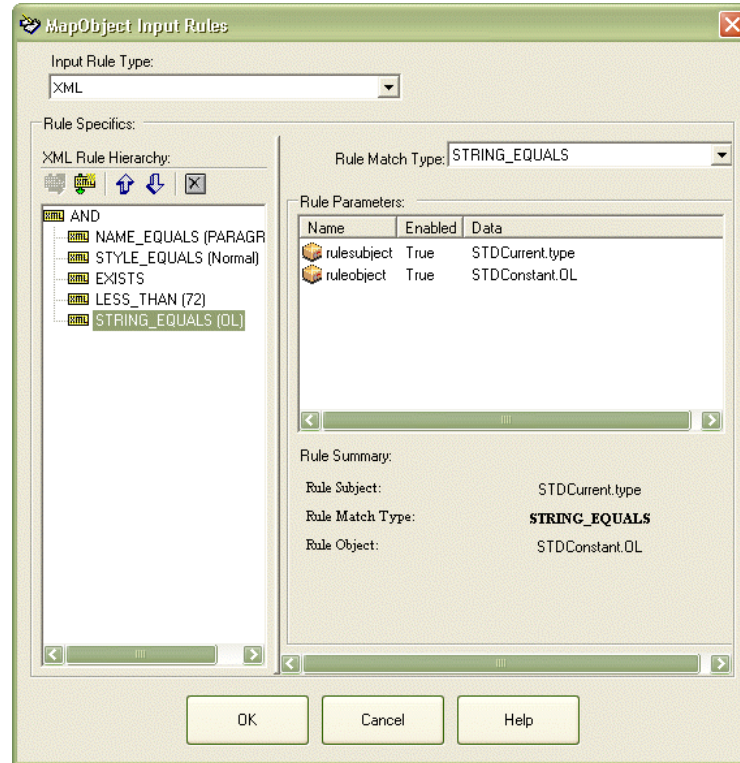


Figure - 6.24

4. In the output rules, change the output name for these three objects from “itemizedlist” to “orderedlist”.
5. It is important that each ordered list MapObject should appear before the corresponding bulleted list MapObject in the MapObject order, its rules are more

restrictive. Change the priority of these three MapObjects so they are greater than bulleted list MapObjects.

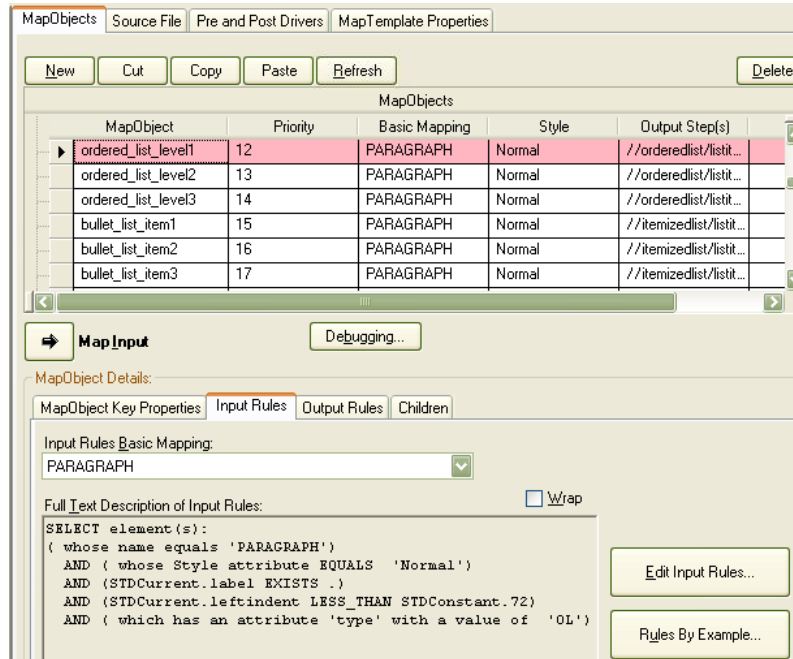


Figure - 6.25

- To test your modifications **Save and Run** the MapTemplate, as a result the **Intermediate Results Display** will look like Figure 6.26.

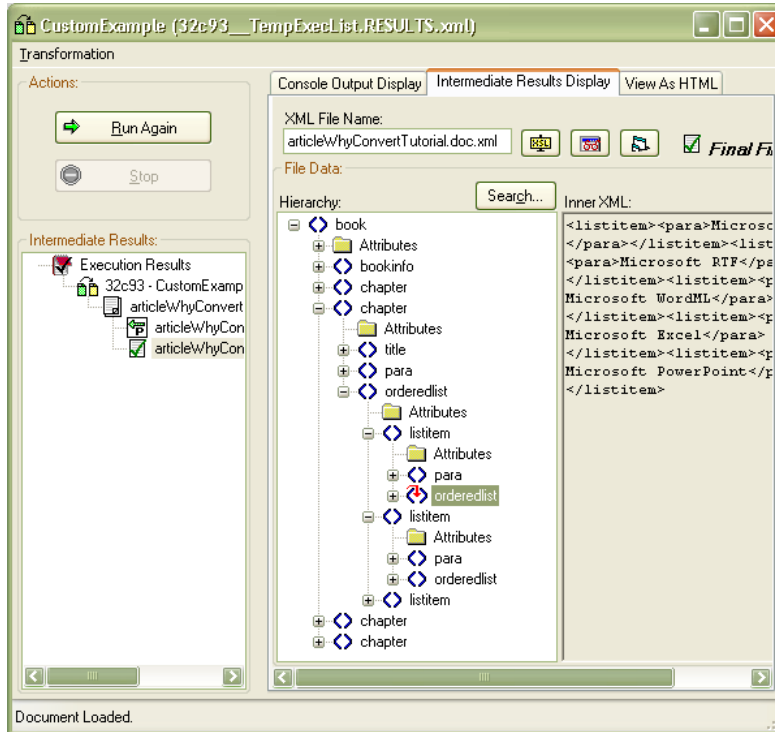


Figure - 6.26

Customizing the Table and Image MapObjects

The DocBook templates shouldn't need to be customized for tables. The `<informaltable>` element is used by default for each table. However, if a table has a caption after it, you might want to call this a `<formaltable>`, that consists of both a title and a table. In this source document (**articleWhyConvertTutorial.doc**), there is a table along with a title after it.

This can be implemented by creating a GRAMMAR_SEQUENCE of a table followed by a table_title.

Before proceeding further for this section the following key terms are to be explained:

- GRAMMAR_SEQUENCE** — This rule operates on a contiguous set of input XML elements; elements that are next to each other, and form a sequence. These could be child elements of the current element, child nodes (such as attributes) or sibling elements. This is the main rule used for grammatically defining a hierarchy.
- #SEQUENCE#** — The basic input mapping type for an advanced XML rule is “#SEQUENCE#”

-
- **STDOject** — The “STDOBJECT Token Grammar Rule” enables you to refer to another MapObject within the input rules of a given MapObject. This is very useful when you want to encapsulate functionality into a reusable object. An example encapsulating the month parsing into a MapObject, called Month is given in the *Text Parsing Rules* section of the *Arbortext Import Reference*.
 - **APPLY** — This rule enables you to call another MapObject to take over processing, it is often used in combination with an “OR” rule. The “APPLY” rule then returns true or false depending on what its child MapObject returned.

To create a GRAMMAR_SEQUENCE of a table followed by a table_title, first you have to create table_title MapObject. To create the table_title MapObject:

1. Select the table’s caption in the **Source File** HTML view.
2. Click **Locate Selected Text**. It will find the <PARAGRAPH> in the ppXML tree to the left of the window.
3. Once the caption is highlighted, click **Map Input**.
4. Perform the following steps on each window of the **Map Input Wizard**:
 - **MapObject Specification**:
 - Select the **Create New MapObject** option.
 - Click **Next**.
 - **MapObject Properties**:
 - In the **MapObject Name** box type in table_title.
 - Click **Next**.
 - **Basic Input Mapping**:
 - This will be PARAGRAPH.
 - Click **Next**.
 - **MapObject Style Mapping**: The caption in this document has a style of Caption, that is OK, click **Next**.
 - **MapObject Attribute Mappings**: There is no attribute that we need to map, since the style and the content will be enough. Click **Next**.
 - **Text Verification**: You will notice that the content of this paragraph will distinguish it from the content of a figure caption. The table title should always start with the word “Table” here.
 - Select the **Element Text** option.
 - Choose STARTSWITH as a rule.
 - Select the **Text** option.
 - Type the word Table.
 - You can click **Finish**.

When you are done with the **Map Input Wizard**, you should see rules as follows in the English Description:

```
SELECT element(s) :  
  ( whose name equals 'PARAGRAPH ' )  
  AND ( whose Style attribute EQUALS 'Caption' )  
  AND ( STDCurrent.Text STARTSWITH STDConstant.Table )
```

5. Before moving to the SEQUENCE for formal_table, we should see if this MapObject works.

However, there is another MapObject that looks for the Caption style, called “caption”, and if this table_title MapObject comes after the caption MapObject in the MapObject order, then it will not run because the caption MapObject will match the caption in the source document. Change the order using the **MapObject Key Properties** tab to change the order of the caption object. For example, if the caption MapObject has priority 23, then this table_title MapObject should have priority 22 or high (lesser the number higher the priority).

6. If you run the transformation by selecting **Save and Run**, you should get a single table_title. You can quickly find an element by using **Search**. To do this:
 - Click on the final destination file.
 - Select **Search**, which brings up the **Search** dialog box, as explained in [Search on page 27](#).
 - Enter the XPath to find the table_title element as //table_title. If the MapObject is set up correctly, you will see an <informaltable> element followed by a <table_title> element in the hierarchy pane.

We are now ready to create a GRAMMAR_SEQUENCE for formal_table.

1. First, find the informaltable MapObject, and select it.
2. Then, with the informaltable MapObject selected, create a new MapObject by clicking on **New** from the **MapObject** grid. This new MapObject will appear in the MapObject order just before informaltable. This is important because we don't want the informaltable MapObject to catch formal tables as well. We will place the formaltable MapObject first.
3. Change the name of this new MapObject to formal_table using the **MapObject Key Properties** tab.
4. Change the input rules basic mapping type of this MapObject to #SEQUENCE# from the **Input Rules Basic Mapping** list options under the **Input Rules** tab as highlighted in Figure 6.27. This will change the text description of the input rules to say:

Select a SEQUENCE OF SIBLINGS/CONSECUTIVE ELEMENTS,

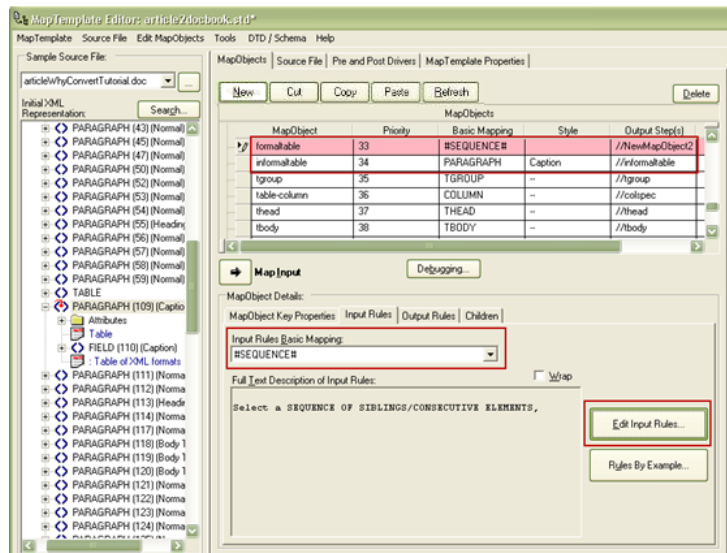


Figure - 6.27

5. You must now populate the sequence. This can be done by editing the input rules from **Edit Input Rules**. In the **XML Rule Hierarchy**, you will see a single XML Rule of matchtype **GRAMMAR_SEQUENCE**.
6. Click the **GRAMMAR_SEQUENCE** rule, and look at its parameters in the **Rule Parameters** section. When you click the parameter, you can see and edit its value. Click the **sequencetype**, and insure that **SIBLINGS** is selected. This specifies that you are looking for a SEQUENCE of sibling elements, namely a table followed by a table title.
7. Now add a child XML rule to the **GRAMMAR_SEQUENCE** rule. This can be done using **Add Child XML Rule**, while **GRAMMAR_SEQUENCE** rule is selected in the hierarchy. This will create a new XML rule as a child of the **GRAMMAR_SEQUENCE** with the type of **ANY**.
8. The first step of the sequence is a table. We will create a rule with a matchtype of **APPLY** and a **ruleobject** of **STDOBJECT**, and MapObject selected as **informaltable**. To do this:
 - Select the newly created rule.
 - Change the **Rule Match Type** of the rule to **APPLY**.
 - Then click its **ruleobject** parameter. You will see the **Parameter Details for ruleobject** section in the lower right of the window. Here, the rule prefix for **Summary** should be automatically set to **STDObejct**.
 - Select MapObject using **MapObject** browse (...).
 - Select **informaltable** MapObject on the **Choose MapObject** dialog box.
 - Click **OK**.

9. Now similarly add an **APPLY** rule with **ruleobject** of **STDOBJECT**, and **table_title** selected as **MapObject**. The resulting **MapObject Input Rules** window is similar that shown in Figure 6.28.

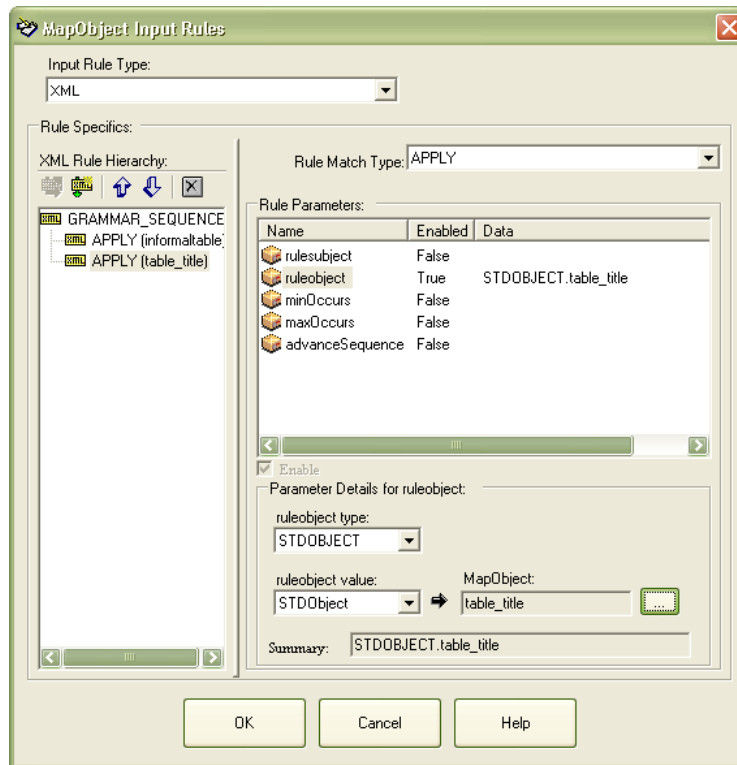


Figure - 6.28

10. Click **OK**.
11. You have updated input rules for the newly created **MapObject** but the output rules still need to be modified.
 - Select the **Output Rules** tab.
 - Select the **Output Step** tab, where you will see some default **MapObject** name.
 - Change the name of the newly created **MapObject** to **formal_table**.
12. Now **Save and Run** the transformation and you will see a **formal_table** element in the **DocBook XML**.
13. Since **formal_table** and **table_title** elements are not there in the **DocBook schema**, you will now disable **formal_table** and **table_title** **MapObjects**. Do this by deselecting the **Enabled** option for the two **MapObjects** in the **MapObject Key Properties** tab.
14. **Save and Run** the transformation again.

Now customize input XML rules for IMAGE elements. You will notice that there is a mediaobject MapObject that finds a ppXML paragraph with an “Image” in it, followed by zero or more occurrences of a para that is a caption.

In the default DocBook templates, the caption comes after the IMAGE. However, in our sample tutorial document (**articleWhyConvertTutorial.doc**), the captions come before the IMAGE. The change can be made by selecting mediaobject MapObject in the MapObject grid and modifying its input XML rules. To do this:

1. Select mediaobject MapObject from the grid.
2. Click **Edit Input Rules**.
3. Now use the up and down buttons to rearrange the child XML Rules of the GRAMMAR_SEQUENCE XML Rule as shown in Figure 6.29.

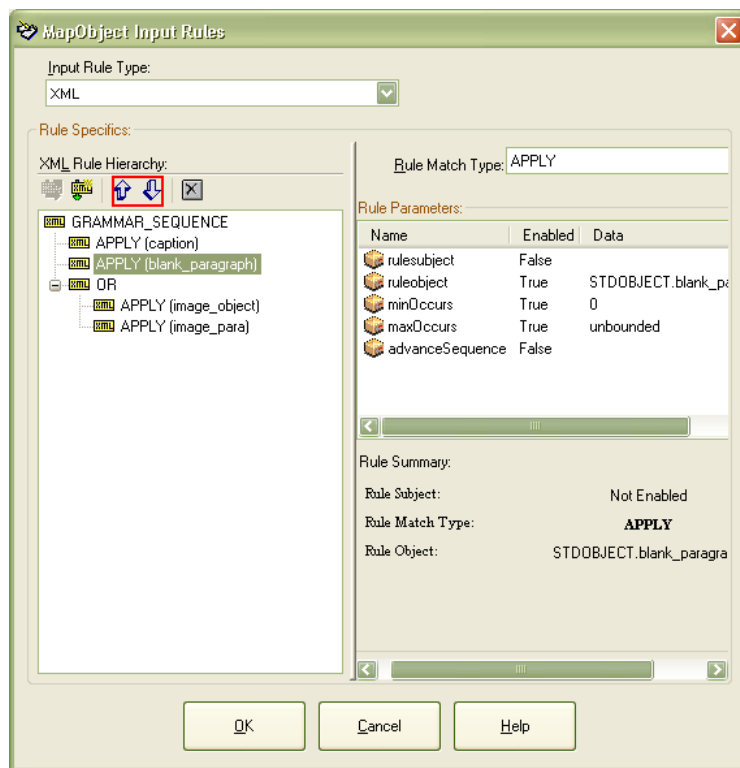


Figure - 6.29

After modifying these, the English description should come out:

```
Select a SEQUENCE OF SIBLINGS/CONSECUTIVE ELEMENTS,
beginning with 1 occurrence of an element:
 ( which matches the input rules for mapobject: caption)
ending with 0 or more occurrences of an element:
 ( which matches the input rules for mapobject: blank_paragraph)
followed by 1 occurrence of an element:
 (( which matches the input rules for mapobject: image_object)
```

OR (which matches the input rules for mapobject: image_para)

Adding or Customizing Remaining Other Objects

Remaining objects are customized in the same way as other objects. In the ppXML, when a <PARAGRAPH> is matched by a MapObject, the MapObject will then pass on each of the child nodes of the <PARAGRAPH>. The child nodes may be child elements or child Text/CDATA nodes. By default, the child elements are passed on to all MapObject rules in order, until one is found that matches. Child Text/CDATA nodes are treated in the same way, and their content is copied to the output. Both of these approaches will place their results inside the core **Output Step**. If a <PARAGRAPH> in ppXML maps to a <para> in DocBook, then all of the children of <PARAGRAPH> in the ppXML will by default appear under <para> in DocBook.

Before proceeding further in this section, the following key terms are defined:

- **SPECIALTEXT** — A <SPECIALTEXT> element describes a specific run of text within a <PARAGRAPH> that has different formatting than the <PARAGRAPH> itself.
- **MARKER** — A <MARKER> is a possible destination in a file; usually a Word Bookmark.

The most common inline element in ppXML is <SPECIALTEXT>. The DocBook templates should map <SPECIALTEXT> elements that are bold or italicized to <emphasis> elements in DocBook. Similarly, <SPECIALTEXT> elements which are super or sub scripted (indicated by the emphasis-superscript=true, or emphasis-subscript=true attributes in the ppXML), should be mapped to super or sub elements in DocBook. Other inline elements include LINKs, FIELDs and MARKER elements, representing Microsoft Word fields and bookmarks.

By default, the Arbortext Import templates for DocBook contains the include Fields option that is set to false, that means that FIELD elements will not be produced in the ppXML. You can however, change it. For example, in our sample document (**articleWhyConvertTutorial.doc**) all table and figure captions, and references use Microsoft Word FIELDs of several types:

- TheSEQfield is used to auto-generate a number for the Figure or Table.
- TheREFfield is used to refer to the a Figure or Table from the content.

To add these items to DocBook, we must strip the “Figure”, or “Table” at the beginning of each caption, and tell the FIELD that has the SEQ in it to not output anything. Moreover we need to be sure that the bookmark id for each caption is added to the caption para, and that the REF field translates into a DocBook xref element correctly. To do this, perform the following steps:

1. Ensure that in the preprocessing “Word to XML Driver (Java)” the “IncludeFields” and “IncludeBookmarks” options are set to true. For details consult the *Pre and*

Post Processor Drivers section in the *Arbortext Import Reference*. By default these options are set to `true`. If they are not set to `true`:

- Set these options to `true`.
- Save the template.
- Choose **Rerun Initial Preprocessing Driver Transformation** from the **Source File** menu item so the source file ppXML on the left hand side of the editor shows the updated options.

To confirm that option “IncludeBookmarks” is true:

- Click **Search** in the editor’s left pane.
- Type in the following **Search Expression** `//MARKER`.
- Select **XPath Search**.
- Click **Find**, it should find the first MARKER that is right before a FIELD element whose fieldcode attribute includes SEQ.

2. Let us create a MapObject that recognizes this field (which uses fieldcode SEQ).

- Select the FIELD element on the left hand side, and start the **Map Input Wizard**.
 - Create a new MapObject, call it `seq_field`.
 - Basic Input mapping should be `FIELD` by default.
 - You will see the **MapObject Field Data Mapping** wizard, which guides you through creating input rules for FIELD elements in ppXML. In this case the field name should be `SEQ` by default. It will have an identifier of `Figure` as shown in Figure 6.30, because the original field code that we have contains this. Here, we want our MapObject to catch either table or figure caption FIELDS, you can delete this **Identifier**.

- Click **Finish**.

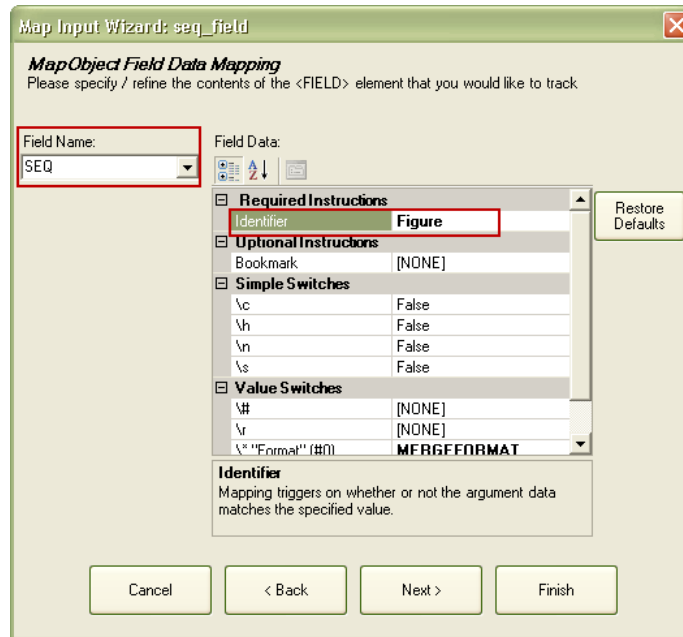


Figure - 6.30

The input rules will be created as follows:

```
SELECT element(s) :
( whose name equals 'FIELD ' )
AND ( which has an attribute 'FieldName' with a value of 'SEQ ' )
AND ( whose FieldArgument CONTAINS '' )
AND (STDField.\* EXISTS .)
AND ( whose \* CONTAINS 'MERGEFORMAT' )
```

- If you run the transformation again, you should see several seq_field in your output DocBook XML (it will not validate of course, because seq_field is not a valid DocBook element). In fact, you should find one for each figure and table caption that uses it.
 - We will soon change the output rules for this object to not output any elements, but leave them as they are for the moment, because we want to process MARKER.
3. Let us now create a MapObject for MARKER using following steps.
 - Select MARKER on the left hand side.
 - Click **Map Input** for the **Map Input Wizard** to open.
 - Create a new MapObject for processing the MARKER element.
 - The name of this MapObject should be marker.
 - The basic mapping type will be MARKER. No other input rules are needed so finish here.

The output rules for Marker will be a bit different, we want to get the value of the MARKER's name attribute, that points to the Word bookmark that the MARKER represents. To do this:

- a. Select the **Output Rules** tab.
- b. Select the **Step Details** sub-tab.
- c. Click **Edit** in the **Output Attributes**.
- d. The **Edit Output Attribute** window opens. From this window perform following steps:
 - Create a new attribute called `id`.
 - Give it an **Output Attribute Rule** of `Input`.
 - Select **Source Element Attribute**.
 - type the name of the source attribute as `name` from which the `id` attribute will get its value.
 - Click **OK**.

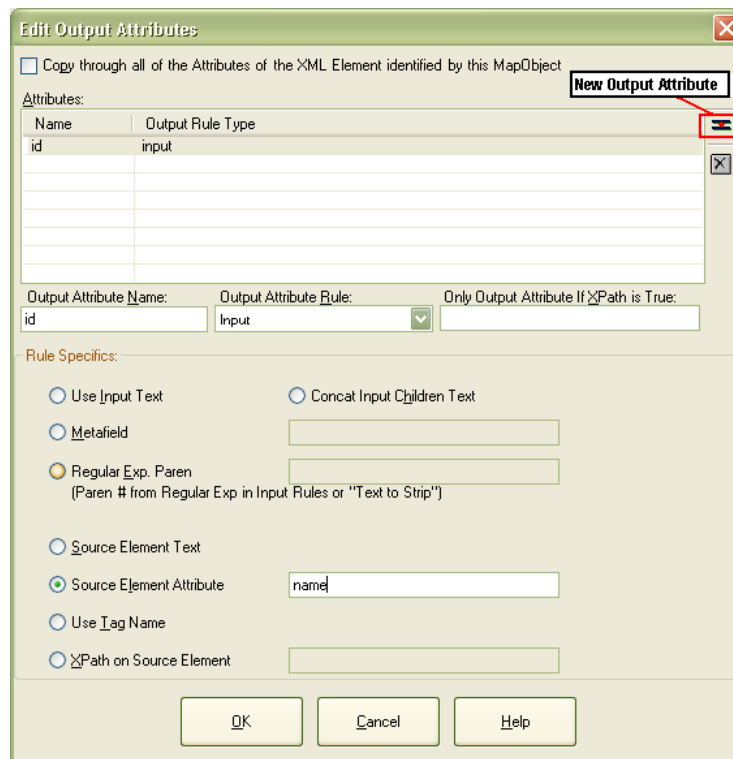


Figure - 6.31

- e. Click **Edit** in the **Output Tag/Element** on the **Step Details**.
- f. Set **Output Tag/Element** to **No Output**.

- f. After selecting **MapTemplate ► Save and Run** you will see the text “Figure : ” and “Table : ” are stripped from the output.

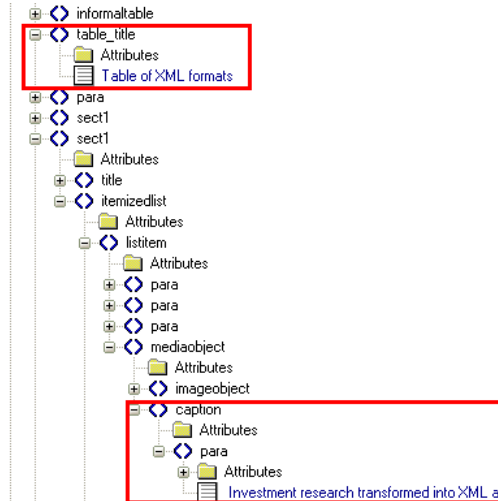


Figure - 6.32

 **Note**

Enabling the `tabel_title` MapObject and applying the strip text rule is just for illustration purposes. Disable `tabel_title` MapObject again before proceeding further.

5. Now you will create a MapObject for the REF field, and map it to DocBook’s xref element. To do this:
 - a. Find the FIELD element (search for `\\FIELD XPATH`) that is of `fieldcode=REF` or `type=3` (these are the text in the paragraphs which refer to Figure 1 or Figure 2). You will see that the `fieldcode` attribute of the FIELD element in the ppXML is “REF_XXXXX” where `_XXXXX` is the name of the bookmark in Word.
 - b. Start the **Map Input Wizard** by clicking **Map Input**.
 - c. Using the **Map Input Wizard**:
 - i. Create a new MapObject, and call it `xref`.
 - ii. It’s basic mapping type will be `FIELD`.
 - iii. On the **MapObject Field Data Mapping**, delete the Bookmark name from the “Bookmark”.
 - iv. Click **Finish**.

You will see that there is still an input rule that looks at the FieldArgument; you can delete this input rule from the input rules editor, and then your input rules description should be:

```
SELECT element(s) :  
  ( whose name equals 'FIELD ' )  
  AND ( which has an attribute 'FieldName' with a value of 'REF ' )  
  AND ( whose FieldArgument CONTAINS ' ' )  
  AND (STDField.\h EXISTS .)
```

- v. Now you will go to the xref MapObject's **Output Rules** tab, and create an attribute on the xref element that is called "linkend", and its **Output Attribute** rule should be "Input Field" that this means that the input fields go to the output fields (attributes). You will then choose "fieldarguments".
- vi. Furthermore, xref elements in DocBook are not allowed to have any text inside them. Go to the **Children** tab, and select **Do Not Apply MapObjects**.
- vii. If you run it by selecting **MapTemplate ► Save and Run**, you will find xref elements in several places, wherever there was a reference to a figure in the normal text.

Validating the Final DocBook Template

At this point you will have to validate your final DocBook template. For this purpose:

1. Select the **MapTemplate Properties** tab.
2. Click **More MapObject Options**. A new **Options** window is opened just as Figure 6.33.

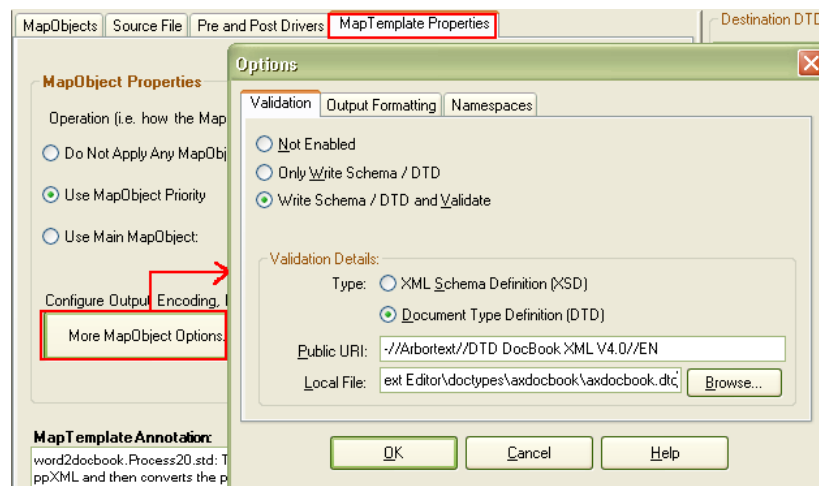


Figure - 6.33

3. Here, select **Write Schema/DTD and Validate**.
4. Select **Document Type Definition (DTD)** under **Validation Details**.

-
5. Browse to the location where your validating DTD is located (*Arbortext-path* \doctypes\axdocbook\axdocbook.dtd).
 6. Click **OK**.
 7. Run the template by selecting **MapTemplate ► Save and Run**. On the **Transformation** window you will see no validation errors.

Conclusion

This chapter shows how to customize the DocBook templates for Microsoft Word. The DocBook templates for HTML and MIF are the same as the Word templates, and the exact same methods used here can be used for those transformations as well. Furthermore, even if you are building your own MapTemplates, you can use the same method as described here.



Text Files

Parsing Text Files using Text Rules

Goals

The goals of this appendix are to:

- Get experience by creating MapTemplates that can parse text files
- Get experience by using SIMPLE Text input rules such as:
 - Line
 - Text Match
 - Regular Expression
 - Fixed Length
 - HTML Tag
- Get experience by using TOKEN GRAMMAR Text input rules
- Understand how to create MapObjects, child MapObjects
- Understand how to use output rules

Overview

This chapter will enable you to create MapObjects that parse text input files, and produce XML output. You can use the sample text input file, included in the installation under ***Arbortext-path\samples\importexport\text***.

The following two files should be in this directory:

- **patients_extra.txt**
- **patients_extra.dtd**

The following Figure A.1.1 shows the sample file that is opened in Notepad. This is an extremely useful file to demonstrate the different types of text parsing rules.

Once you understand how to use these text-parsing rules, you can use them in the context of parsing, for example, your own Microsoft Word or HTML files to extract key informations.

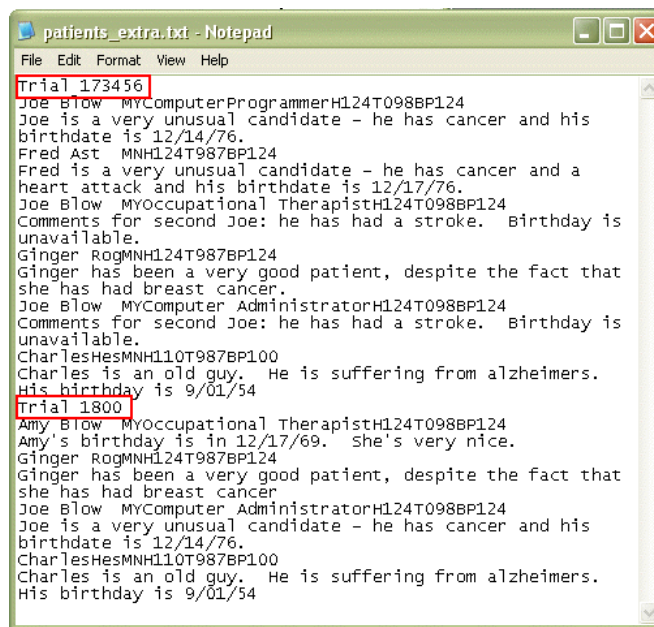


Figure - A.1.1

This sample text file contains a number of lines and information that spans on multiple lines. It is basically a simulation of a patient record file.

There are two trials, each of which has one or more patient records. Each patient record has one line that contains character based information about the patient: Name, Gender, Occupation Available and Readings, that include heart rate, temperature and blood pressure, and one line that contains comments about the patient.

Tutorial: Destination XML File

Suppose we want to create an XML file for this that looks like the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE PatientData SYSTEM "C:\Arbortext\samples\text\patients_extra.dtd">
<PatientData>
<Trial number=1800>
<PatientRecord>
```

```

<PatientName>Joe Blow</PatientName>
<PatientGender>M</PatientGender>
<PatientOccupation available=yes>Computer Programmer</PatientOccupation>
<PatientReadings>
<HeartRate>124</HeartRate>
<Temperature>098</Temperature>
<BloodPressure>124</BloodPressure>
</PatientReadings>
<Comments>
Joe is a very unusual candidate he has cancer and his birthdate is 12/14/7
</Comments>
</PatientRecord>
<!-- MORE PATIENT RECORDS >
</Trial>
<Trial number=1800>
<!-- ONE OR MORE PATIENTRECORDS >
</Trial>
</PatientData>

```

Following is an example of the target XML we want. This is a relatively simple DTD.

```

<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT PatientData (Trial*)>
<!ELEMENT Trial (PatientRecord*)>
<!ATTLIST Trial number CDATA #IMPLIED>
<!ELEMENT PatientRecord (PatientName, PatientGender, PatientOccupation,
PatientReadings, Comments)>
<!ELEMENT PatientName (#PCDATA)>
<!ELEMENT PatientOccupation (#PCDATA)>
<!ELEMENT PatientOccupation available ( Y | N | y | n | yes | no ) #IMPLIED>
<!ELEMENT PatientGender (#PCDATA)>
<!ELEMENT Comments (#PCDATA)>
<!ELEMENT BirthDate (#PCDATA)>
<!ELEMENT Condition (#PCDATA)>
<!ELEMENT PatientReadings (HeartRate?, Temperature?, BloodPressure?)>
<!ELEMENT HeartRate (#PCDATA)>
<!ELEMENT Temperature (#PCDATA)>
<!ELEMENT BloodPressure (#PCDATA)>

```

This appendix walks step by step through the process of using simple and advanced text input rules to parse the text file to create the desired XML file.

Create a new MapTemplate

If you have already created a new project, then you can open and reuse it. If not, then launch Arbortext Import, and choose **New** from the **File** menu on the main Arbortext Import Workbench window. Further select **Project**, and give a name to the new project,

such as “tutorial”, as explained in [Creating a New Project Using Arbortext Import Workbench on page 18](#).

The first thing to do is to create new transformation. The procedure in detail is explained in [Create a Transformation on page 19](#).

Choose the source file `patients_extra.txt` from `Arbortext-path\samples\importexport\text`.

While creating transformation through the wizard, it will ask for a MapTemplate, where you should click **New**, and it will launch the **New MapTemplate Wizard**.

Create a new subfolder of STDTemplates called "Tutorials" by clicking **Browse**, and then give a name to the template, such as `textParsingTutorialPatients`, as shown in Figure A.1.2.

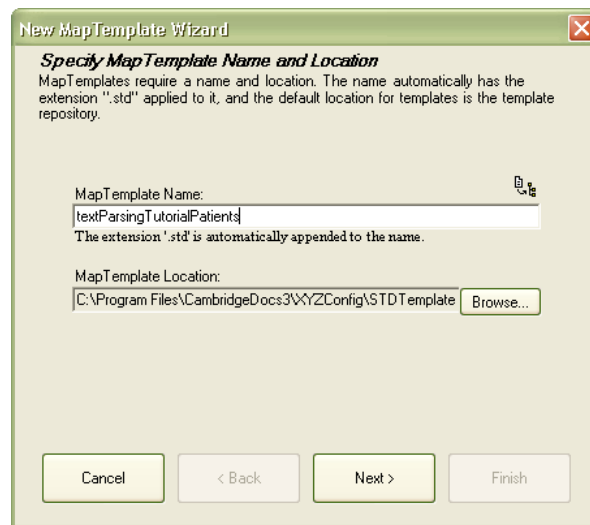


Figure - A.1.2

The template should not be based on existing template. Select the **Create New MapTemplate** option on the next window and click **Next** to continue.

On the **Specify MapTemplate Driver** window, you should choose “None” from the list, because we are parsing the text file directly, and do not need a driver (the drivers typically parse a file, such as a Word document into ppXML, that we don’t need to do here).

Now when you click **Finish**, it will take you back to the **New Transformation Wizard**, along with the name of the new MapTemplate that was just created. Click **Next**, on the next window add a nickname for this new transformation, such as `patients_extra_textParsingTutorialPatients`, and click **Finish** on the wizard.

After completing all the steps on the wizard, a new transformation of `patients_`

extra_textParsingTutorialPatients is present in the Arbortext Import Workbench as shown in Figure A.1.3.

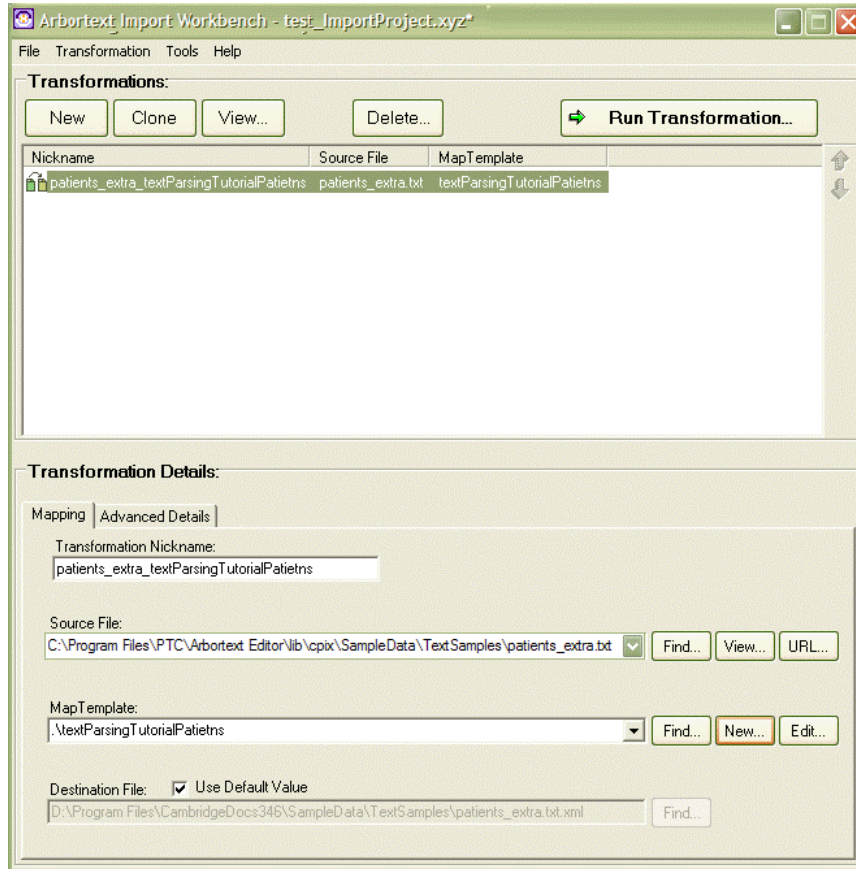


Figure - A.1.3

You can then edit the MapTemplate by clicking **Edit** next to the MapTemplate name in the

Transformation Details section of this screen. The MapTemplate Editor window is shown in Figure A.1.4.

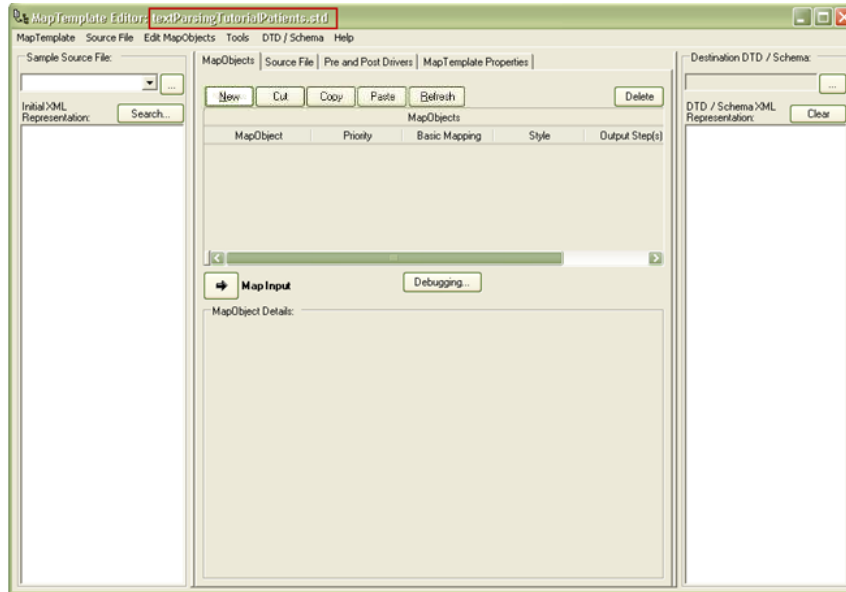


Figure - A.1.4

The MapTemplate Editor is displayed with no source file selected on the left, and no target DTD showing on the right. On the top right of MapTemplate Editor, you will see **Destination DTD/Schema**.

Click **Browse** on the **Destination DTD/Schema** pane, that will show you the list of target schemas/DTD's that have been parsed to date in a new window. Click **Load New**, and find patients_extra.dtd from the same directory where the source file was located (**Arbortext-path\samples\importexport\text**).

This will cause the product to compile this DTD into the Arbortext Import internal format. Once this is done, click **Choose Main Element**, and in the opened window choose "PatientData". Click **OK**, and select patient_extra in the list of **Available DTD/Schema**, and click **OK**.

In the left most pane of the **Sample Source File**, you can choose the **patients_extra.txt** file from the list of source files. The product will try to convert this document using a generic template to ppXML, but since we don't convert text files into ppXML, no XML tree will be displayed. You can click the **Source File** tab, that shows the source file in an embedded Internet Explorer window. You can click **Reload**, and you

will see the source text file in the HTML window. You can use this for reference while parsing, as highlighted in Figure A.1.5.

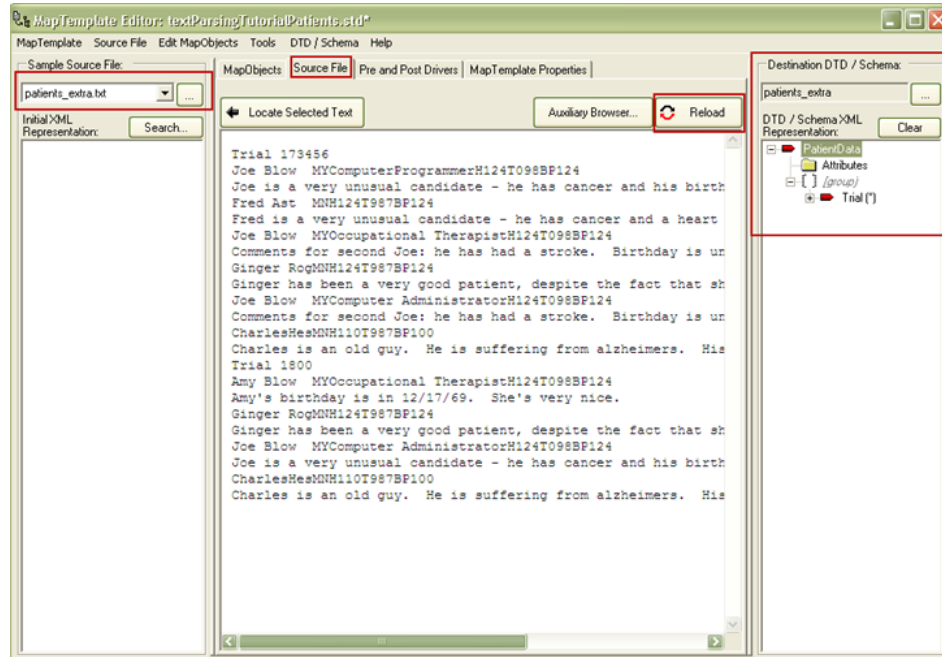


Figure - A.1.5

Now click the **Pre and Post Drivers** tab. We are dealing with a text file, so we do not need any preprocessing. (Preprocessing drivers are generally used to transform Word, PDF, HTML files into ppXML that can then be parsed.) However, we need a main element that can start the MapTemplate transformation. The root element in our output XML is PatientData, so it makes sense to create a MapObject that understands how to create an output element of PatientData. We have not created any MapObjects yet, so we can't assign a main element at this point.

Let us start with defining text parsing MapObjects. The first thing we need to do is to create a MapObject to start our parsing at the top of the file.

Create a Main MapObject

Click the **MapObjects** tab in the **MapTemplate Editor**. Then click **New** under the **MapObjects** tab. This will create a new MapObject that will be the first MapObject in our MapObjects tree, along with a default name of `NewMapObject1`.

First of all, we will change the name of the object to the top level object in our target DTD, that is "PatientData". This can be done on the **MapObject Key Properties** tab in the **MapObject Details** section of the window.

Secondly, we will change the source type, that is set to XML by default, to input text, because we will be parsing a text file. This is done by selecting `#PASSTHRU#(text)` from the **Input Rules Basic Mapping** list in the **Input Rules** tab. This will display a warning

dialog box about changing the basic mapping type from an XML based type to a text parsing type. To disable this warning, go to the MapObject editing options by choosing **MapObject Editing Options** from the **Tools** menu, and set the **Default New MapObjects to text**.

If you look at our target XML (shown earlier), and our target DTD, <PatientData> is the top level element of our Target XML, but it doesn't have any text or attributes.

This means that the input rules for this MapObject should select all of the source text, and pass it on to its child MapObjects for further processing. The PatientData MapObject does not perform any processing of the text itself.

Click **Edit Input Rules** on the **Input Rules** tab, and as a result, the **MapObject Input Rules** window will open. You will notice that by default, the **Input Rule Type** is "Select Current Text", that is the default for the basic mapping type of "#PASSTHRU#(text)". This means that when this input rule runs against our source file (**patients_extra.txt**), it will select all of the text in the file. Click **OK** in the **MapObject Input Rules** window. After completing the steps for a text input file, the MapTemplate Editor will appear as shown in Figure A.1.6.

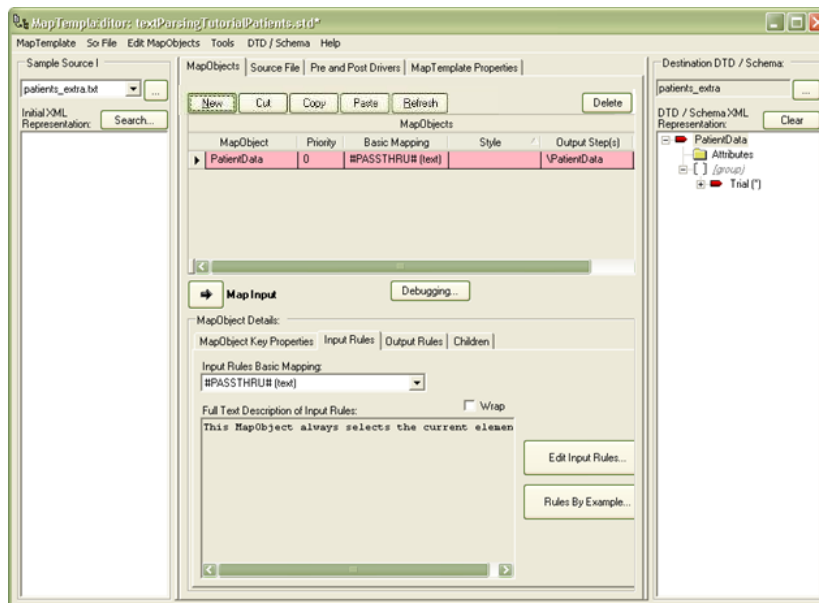


Figure - A.1.6

Now let us look at the output rules for this MapObject. First, click the **Output Rules** tab, the MapObject is already selected in the **Output:** box, and click **Edit Output Attributes/Rules** to get access to these three output rules:

- **Output Tag Name** tab
- **Output Value** tab
- **Output Attributes** tab

The **MapObject Output Rules** window is shown in Figure A.1.7.

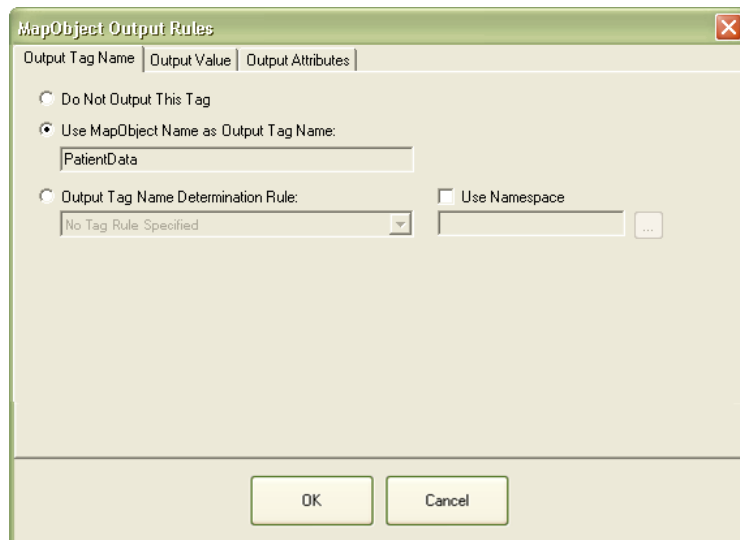


Figure - A.1.7

Unlike XML Rules, which might have an output hierarchy, each text parsing MapObject creates only one output XML element.

If you click the **Output Tag Name** tab as shown in Figure A.1.7, you will see the name of the element that is created by this MapObject. This tab enables you to change the name of the output element. By default, the output name of the XML element is the same as the name of the MapObject “PatientData”. There is no need to change the default values on this tab.

If we look at the **Output Value** tab, the default is **No Output Rule specified**. This is OK because the <PatientData> element in our target output file doesn’t have any text nodes underneath it. It contains only child elements, that will be produced by MapObjects on their own. The same is true for the **Output Attributes** tab; the default works fine, because there are no attributes needed in the <PatientData> output element

Setting the Main MapObject and Testing

Before we move further, we can test our MapTemplate to see if it parses the source file correctly so far. Logically, it should create XML that has a <PatientData> element with no output value, and no attributes or children.

Before we run the MapTemplate, we need to specify a Main MapObject. Click the **MapTemplate Properties** tab, change the MapObject properties to be **Use Main MapObject**, and select the “PatientData” MapObject from the list which we just created.

You can now choose **Save and Run** from the MapTemplate menu in the **MapTemplate Editor**, and the **Transformation** window will appear. When the transformation is completed, the results of this MapTemplate running against the source file **patients_extra.txt** should be displayed.

You should see a PatientData element, and no other output in the **Transformation** window.

Creating and Configuring the Child MapObject: Trial

Our output requires that we have <Trial> elements as children of the <PatientData> element. So, let us create a child MapObject for Trial. This is done by clicking **Child** on top of the editor, as highlighted in Figure A.1.8.

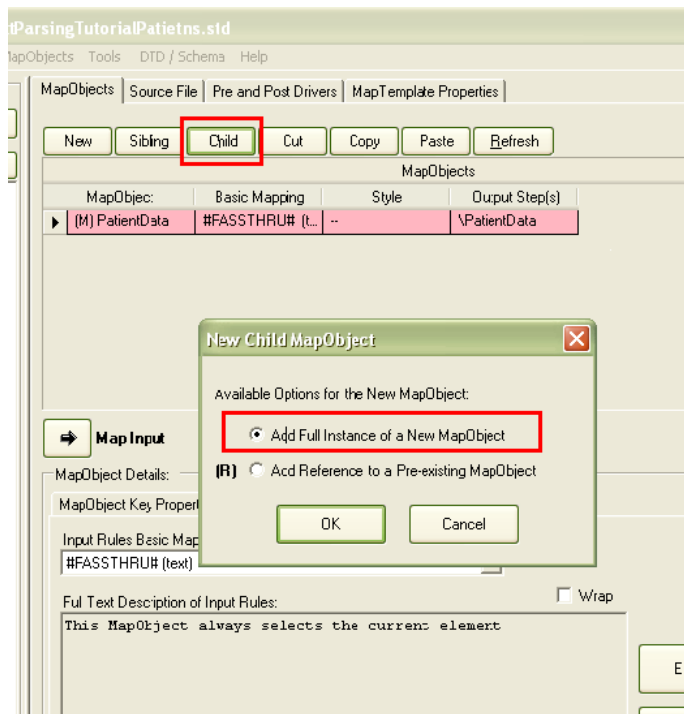


Figure - A.1.8

When you click this button, you will see the **New Child MapObject** dialog box. This dialog box asks if you really want to create a new MapObject, or if you want to refer to an existing MapObject. If the MapObject already exists, we could create a reference to it here. In this case, we want to create a full instance of a new MapObject.

This will create a new child MapObject, that has a default name of `NewMapObject2`.

Because the goal of this MapObject is to create an output element called <Trial>, we will

change the name of child MapObject to “Trial”. That change can be made on the **MapObject Key Properties** tab.

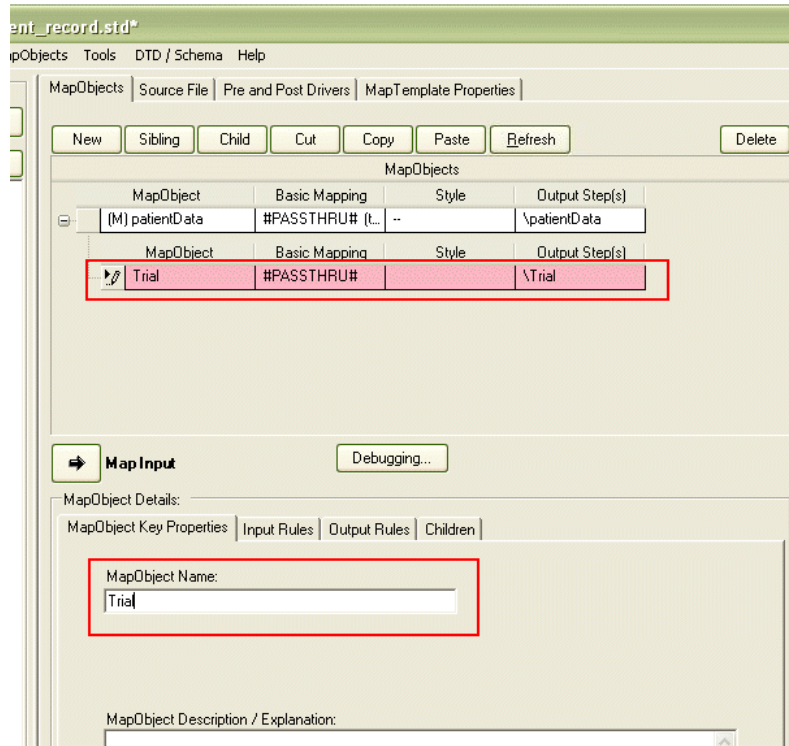


Figure - A.1.9

Now go back to the PatientData MapObject, and click **Edit Child MapObjects...** under the **Children** tab. You will see the Trial MapObject listed as the only child of PatientData. The **MapObject Children** window will appear as shown in Figure A.1.10.

The **MinOccurs** and **MaxOccurs** values tell you how many times this child can appear. In our example, there can be any number of trials within a file, so a **MinOccurs** value of 0,

and **MaxOccurs** value of unbounded is fine. The **Pass Type** defaults to None, and the **Insert** type defaults to None. Both of those values are also fine.

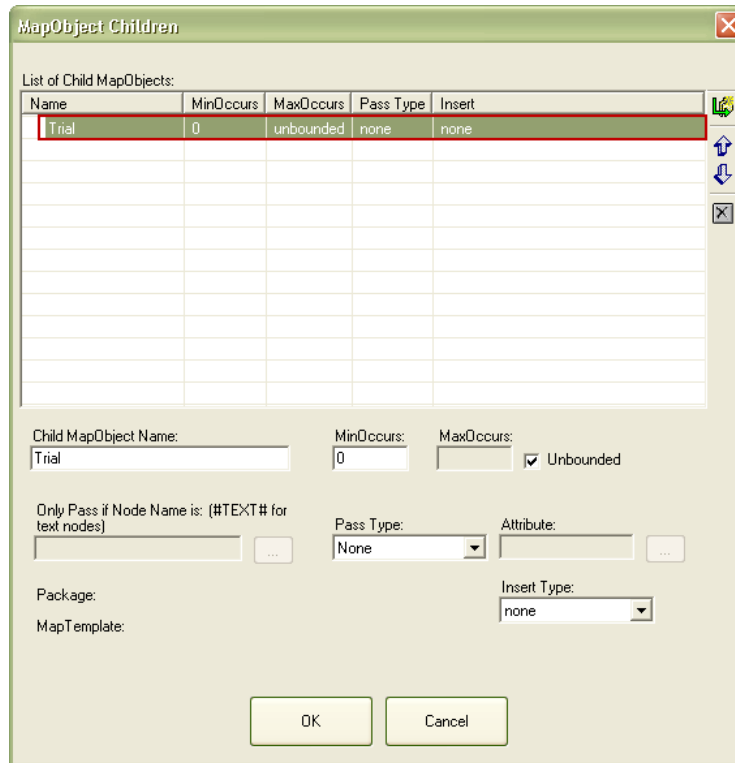


Figure - A.1.10

We are now ready to work on the Trial MapObject.

Select the Trial object in the **MapTemplate Editor** window, as highlighted in Figure A.1.9. On the **Input Rules** tab, change the **Input Rules Basic Mapping** type by choosing #TEXTPARSING# in the list. You will need to do this for all text parsing MapObjects in this tutorial.

If you look at our original source file, one source file can contain multiple Trial sections. We need to configure input rules in the Trial MapObject that can parse out a single trial.

Using the terminology of Arbortext Import, we want the Trial MapObject to find an input selection that looks like the highlighted section in Figure A.1.11.

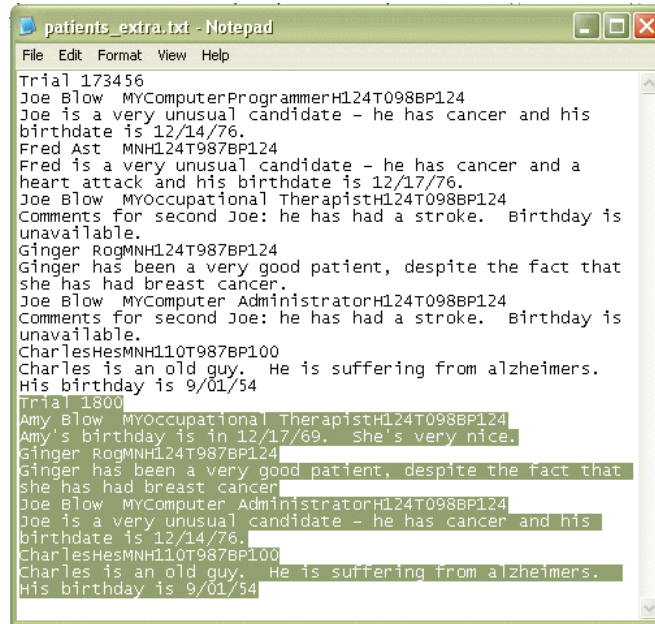


Figure - A.1.11

This can be done by adding input rules that can parse the source file appropriately. In this example, we will use a simple text rule of type "Text Match"; which is a rule that matches a Start token and an End token (Think of each token as simply a word, specified string or regular expression). Each Trial begins with the word "Trial", and ends with the word "Trial" or at the <eof> (End of File).

To do this, click **Edit Input Rules** to open the **MapObject Input Rules** window, and select **Simple Text** from the **Input Rule Type:** list options. In the **Rule Specific:** pane, you will see different types of simple text rules, each of which is referred to as a matchtype. Choose the **Text Match** matchtype, and you will see that you can type in Start and End tokens, and can select any option from **Inclusion Pattern:** options.

Type in "Trial" as the **Start token:**, type in "Trial" as the **End token:**, and choose an **Inclusion Pattern:** of "includestart". An inclusion pattern tells the MapObject whether the start token, the end token, or both the start and end tokens should be considered as part of the found selection. In this case, we want to include the start token, because the word "Trial" is considered a part of the current Trial text, but not the end token (because that is

considered part of the next trial section). After these steps, the **MapObject Input Rules** window is displayed.

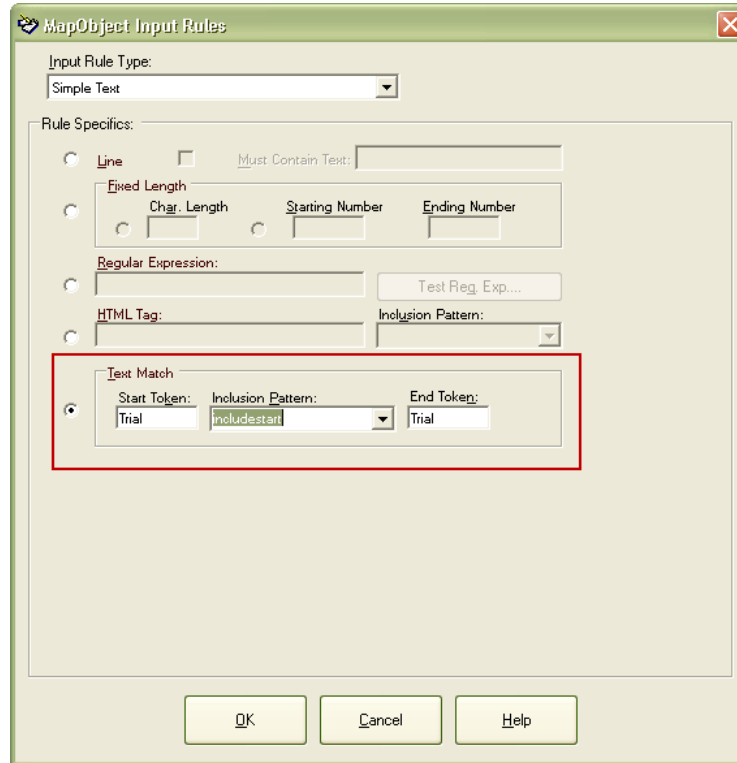


Figure - A.1.12

Now, when we have configured the input rules for this MapObject, click the **Output Rules** tab, and then **Edit Output Attributes/Rules** that will open the **MapObject Output Rules** window. Here, you can click the **Output Tag Name** tab. Since the name of the output element should be <Trial>, the same as the name of the MapObject, we do not need to change the default.

If you look at the **Output Value** tab, you will see that the **No Output Rule specified** option is selected. To test our MapTemplate at this point, let us set an output rule. Click the **Apply Output Rule** option, and then choose a rule from the list options as **Passthru Input Text**. This will enable us to see the selection made by the input rules of the Trial MapObject. Click **OK**.

Before moving further, we can test our MapTemplate to see that if the source file is parsed correctly. Logically, it should create an XML that has a <PatientData> element, along with two child <Trial> elements, one for each trial in the file.

Now you can select **Save and Run** from **MapTemplate** menu item to run the transformation. This will display the **Transformation** window for this template in the console view.

Once the transformation is completed, you will notice that on the left there is one destination file, called **patients_extra.txt.xml**. Select this file, and you will see

that the file is displayed in the **Intermediate Results Display** tab. If you expand the PatientData top level element, you should see two Trial elements below it, each of that contains all of the text in each Trial. If you see both Trial elements, then this was a successful run of the MapTemplate as shown in Figure A.1.13.

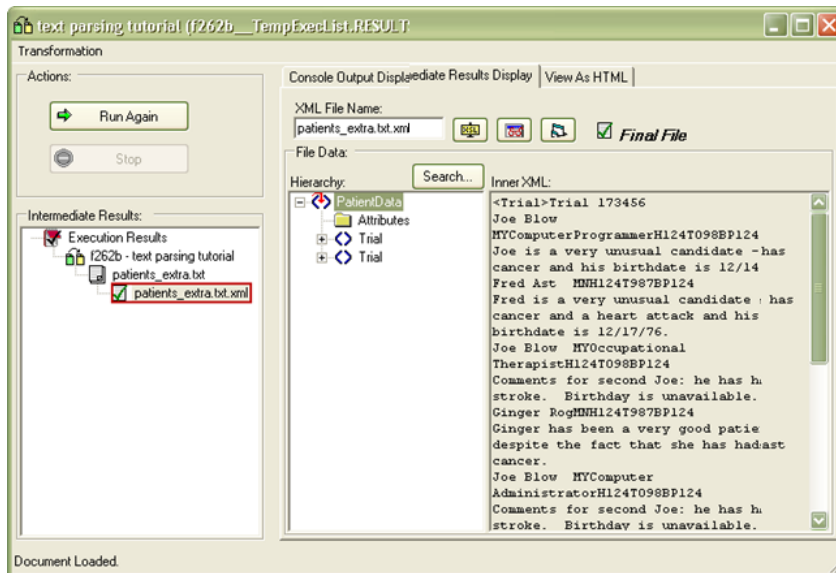


Figure - A.1.13

Parsing the Trial Number

Now we know that our MapTemplate is parsing the file into Trials correctly, we can start with parsing within a Trial. Note that the first line of a trial is the Trial Number, that if you examine the source file in Figure A.1.11 is “173456” for the first trial, and “1800” for the second Trial.

Creating TrialNumberLine MapObject

First, let us create a child MapObject of Trial called TrialNumberLine to parse out this line.

Go back to the **MapTemplate Editor**, and select the Trial MapObject. Once you have selected this Trial MapObject, click **Child**.

You will see the **New Child MapObject** dialog box. You should create a full instance of a new MapObject again, that will be added with a default name. Change the name (on the **MapObject Key Properties**) to TrialNumberLine.

ChildrenSelect the Trial MapObject, and go back to the tab (click **Edit Child MapObjects** that opens the **MapObject Children** window) of the Trial MapObject. The child MapObjects dialog box should list the newly created MapObject, that is TrialNumberLine. In this case, we want at least one TrialNumberLine per Trial. For

minimum and maximum values, set the **MinOccurs** and **MaxOccurs** equal to 1. (Uncheck the **Unbounded** option, and enter the **MaxOccurs** value.)

Once you have completed these steps, select the TrialNumberLine object in the MapObjects grid (you will see that you will have expand Trial MapObject to see TrialNumberLine), and under the **Input Rules** tab, once again, change the rule to #TEXTPARSING#. Click **Edit Input Rules** to set the input rule type to “Simple Text”, and choose **line** as the matchtype. This means it will parse out the whole line.

Now set MapObject output rules (by clicking **Edit Output Attributes/Rules** under the **Output Rules** tab), click the **Output Value** tab, and set the output rule for the MapObject to “Passthru Input Text”.

Now, if you save the MapTemplate and run it again, you will see that each Trial has a TrialNumberLine element, as shown in Figure A.1.14 (this extracts the line with the number in it, rather than the number itself). We will further parse this to extract just the number piece within the line. You should see only 1 TrialNumberLine per Trial; if you see multiple TrialNumberLine elements in a single Trial in your Output XML, then you forgot to set the minOccurs and maxOccurs to 1 in the Child MapObjects dialog box for Trial.

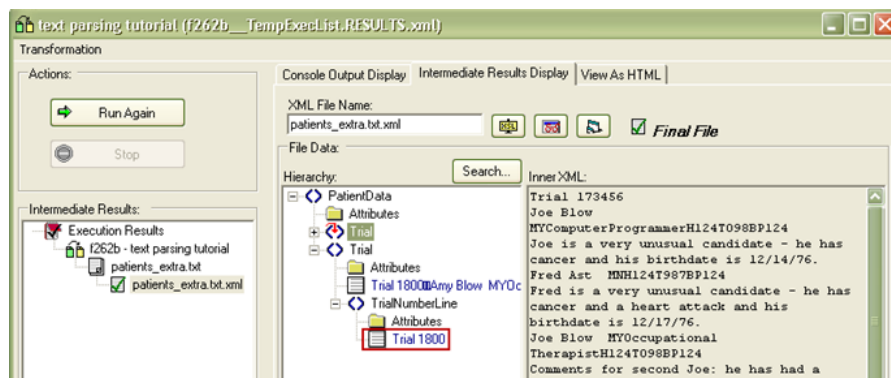


Figure - A.1.14

Creating TrialNumber MapObject

Now we have the Trial Number in a line, and we can further parse it to extract just the number. Let's reopen the MapTemplate and create a child MapObject of TrialNumberLine called TrialNumber. Let's set the minOccurs and the maxOccurs to 1 in the **Child**

MapObjects dialog box of TrialNumberLine. The Number MapObject becomes a child MapObject of TrialNumberLine MapObject, as shown in Figure A.1.15.

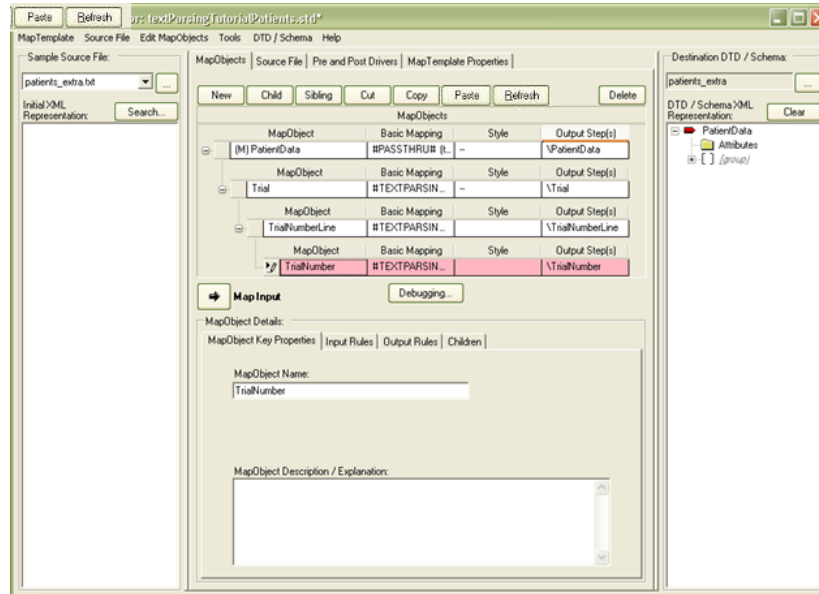


Figure - A.1.15

Now select this new MapObject in the MapObjects tree, and set its basic mapping type by choosing #TEXTPARSING# in the **Input Rules Basic Mapping**.

The **Input Rule Type** (after clicking **Edit Input Rules**) should be set to “Simple Text”. Let us use a regular expression to parse out the number. Regular expression is a well defined language for parsing character data; you can specify which characters will occur in a sequence. The character to find a single digit is the escape character \d. The way to say that there can be one or more digits in a row is to insert a plus sign (+) after the character, like \d+. We will also place parens around it, like (\d+)

From the matchtypes listed, select **Regular Expression**, and type in the regular expression (\d+). This shows you how to use a simple regular expression to extract a value from a string of text. You can also test your regular expression at this point by clicking **Test Reg. Exp**. In the new window, enter a sample text, and click **Run Test** to verify your regular expression.

You also need to set the output value for the TrialNumber Map Object. On the **Output Value** tab for the TrailNumber MapObject, select "Input" for the **Apply Output Rule**, and choose the **Regular Exp. Paren** option.

Then type in 1 in the text box, which means that we want the first set of parenthesis in our

regular expression. In our case, the regular expression was: `(\d+)` which only has one set of parenthesis.

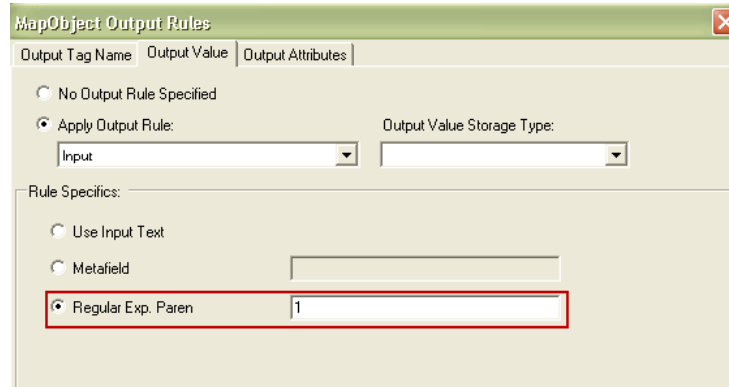


Figure - A.1.16

If you save the template and run it again, you will see that the TrialNumber is extracted.

Now we have parsed the TrialNumber element, there is no need to save the TrialNumberLine element, as this element was simply a convenient way to parse out the TrialNumber.


Go back to the MapObjects, select the TrialNumberLine MapObject, and go to the **Output Element Name** tab (by clicking **Edit Output Attributes/Rules**), and then select the option: **Do Not Output This Tag**.

Also, switch to the **Output Value** tab, and set the output value of this MapObject to **No Output Rule Specified**. Now if you run the same template again, you will see the TrialNumberLine is removed from the output XML.

You see that if you choose the **Do Not Output This Tag** on the **Output Value** of a MapObject, then that MapObject will still execute. It will still process its input rules, and pass on the found input selection to its child MapObjects. The only difference is that it will not output any element. This means that any child elements that child MapObjects produces will become children of the parent of the MapObject that has **Do Not Output This Tag** set. In our case, the parent of TrialNumberLine was Trial, so the children of TrialNumberLine (there is only one TrialNumber here) will become children of Trial in the output.

Creating TrialNumber Attribute

Now for a little more processing, if you look at our desired output schema, we want the TrialNumber to be an attribute of the Trial object, rather than a child element. To accomplish this, click Trial MapObject's **Edit Output Attributes/Rules**, and select the **Output Attributes** tab. It will be blank.

Click **New Output Attribute** . The output attribute name should be "number" in the **Output Attribute Name:** box. But what output attribute rule we will use to get the value of the number?

The number actually resides in a child MapObject, the TrialNumber MapObject. There is an **Output Attribute Rule** called "Child Result", that you can select. The **Output Attribute Rule** should be "Child Result" as shown in Figure A.1.17.

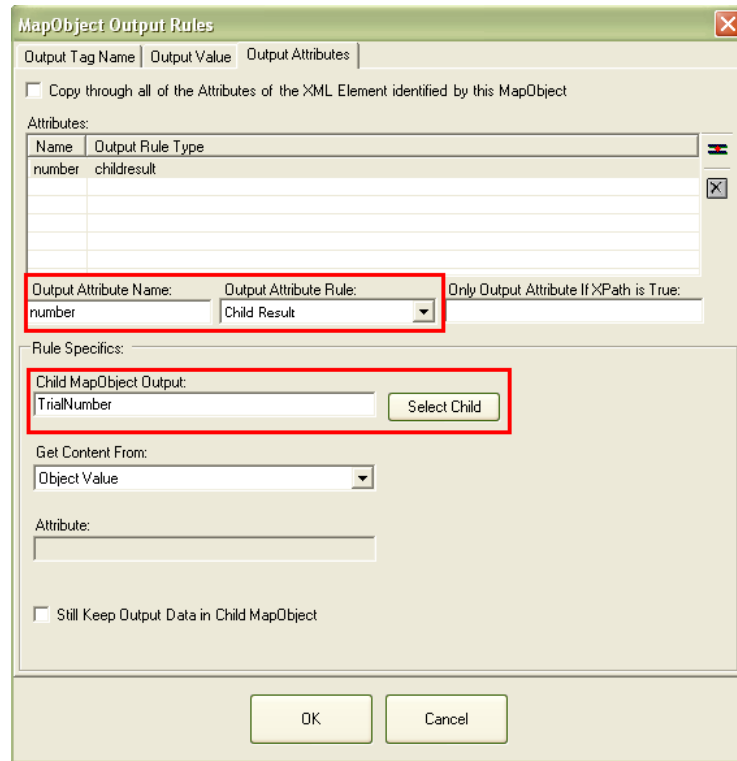


Figure - A.1.17

This **Output Attribute Rule** (Child Result) means that the value of this attribute should not come from Trial MapObject, but instead from one of the child MapObjects. In this case, we want to choose the TrialNumber MapObject (which in reality is a child of the TrialNumberLine MapObject, that is a child of the current MapObject Trial).

If you click **Select Child** (under the **Rule Specifics:** as highlighted in Figure A.1.17), you

will see the **Choose MapObject** dialog box as shown in Figure A.1.18. Select the "TrialNumber" MapObject from this list, and click **OK**.

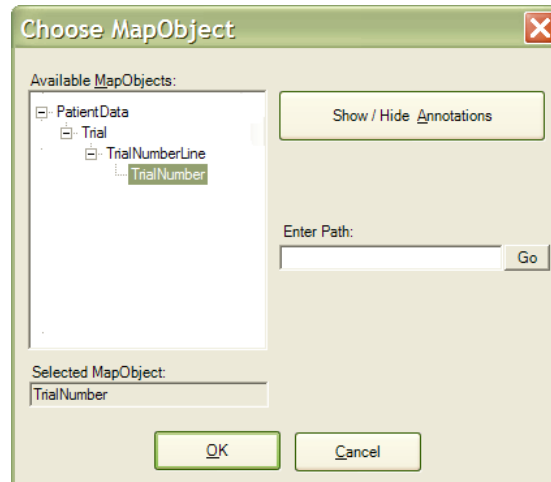


Figure - A.1.18

In short, the Trial MapObject should produce an output element (called <Trial>), along with an attribute called “number”. The number attribute should get its value from a child element, called <TrialNumber>. By default, when you get a value from a child element, then that element and its output is no longer needed (unless you specify the **Still keep Output Data in Child MapObject**, as shown in Figure A.1.17).

Now if you run the template, you should get the <Trial> element, along with the appropriate attribute (number) added to it. You will now notice that both the <TrialNumberLine> and <TrialNumber> elements have been removed, and our output matches very closely to the output we want to get in our final XML.

We have now successfully transformed each <Trial> element, along with its number attribute. We will move to the PatientRecord element next, which is the crux of this transformation. So far, we have used several simple text rules, including line based input rule, text based matching input rule, and regular expression based input rules. We have also used some output rules, including extracting the value of an attribute from a child element, and not outputting certain elements. By not outputting certain elements, we have seen that sometimes you can have MapObjects that are very useful for parsing, but don't need to appear them in the output, such as TrialNumberLine and TrialNumber in our example. This is a very useful technique that you can use often. Remember, a MapObject at its core, is a set of parsing rules that can be re used in various contexts.

Keeping the concept of a MapObject as a reusable set of parsing rules and output rules, we will now get into slightly more advanced text based parsing, that can be done using TOKEN GRAMMAR text rules.

The Patient Record Element

If you recall, the individual `<PatientRecord>` elements are the heart of our output. Note that from our target schema, a `<PatientRecord>` element should look like the following data:

```
<PatientRecord>
<PatientName>Joe Blow</PatientName>
<PatientGender>M</PatientGender>
<PatientOccupation available=yes>Computer Programmer</PatientOccupation>
<PatientReadings>
<HeartRate>H124</HeartRate>
<Temperature>T098</Temperature>
<BloodPressure>BP124</BloodPressure>
</PatientReadings>
<Comments>
Joe is a very unusual candidate he has cancer and his
birthdate is 12/14/76.</Comments>
</PatientRecord>
```

A single `<PatientRecord>` element in the Output

If we examine our source file again, a `PatientRecord` consists of a line that contains values about the patient (patient name, gender, etc.), followed by a line that has a description or comments about the patient.

There are many different ways in which we can parse this `PatientRecord`. However, to keep things simple, let us create two `Child MapObjects` of a `PatientRecord`, each of which uses the **Line** simple text input rule to parse. The first one will be called `PatientInfoLine`, that parses the first line of a `PatientRecord`, and the second one will be called `CommentsLine`, that parses the second line.

The first step is to create a `PatientRecord` child `MapObject` of `Trial`. Do this by opening the `MapTemplate` again. We will show you another way to create a `MapObject`, and referring it from the child `MapObjects` tab.

Click **New** (and not **Child**), and a new `MapObject` will be created at first level of the `MapObjects` grid with a default name.


You can now go to this `MapObject` in the grid of the `MapTemplate Editor`, and select it. The advantage of placing this `MapObject` at the top level of the tree is that it can be reusable. `MapObjects` at the top level of the tree can be referenced from anywhere within the template, and can also refer other `MapObjects`.

Now change the name of this `MapObject` to `PatientRecord`, and change its basic mapping type by choosing `#TEXTPARSING#`.

You will see that no input rules are defined yet, because a `PatientRecord` is defined as consisting of a `PatientInfoLine` and a `CommentsLine`, and we will not be ready to work on the input rules for this `MapObject` unless these other two `MapObjects` have been created.

We will now make this new MapObject “PatientRecord”, child of the Trial MapObject, because a Trial consists of one or more PatientRecords.

Now click the Trial MapObject, and bring up its **MapObjects Children** window. You will see the TrialNumberLine MapObject listed as a child on this window.

Click **New Child MapObject (Reference)** . You will be prompted whether to use an existing MapObject, or to create a new one. This time we want to choose **Add Reference to a Pre-existing MapObject**.

After clicking **OK**, you will get the **Choose MapObject** prompt, that will show the MapObjects you can select.

You will notice that only top level MapObjects are shown. Choose the newly created PatientRecord MapObject from this **Choose MapObject** dialog box. This makes it possible to start reusing MapObjects across different projects. When you choose PatientRecord, and click **OK**, a new child will be added to the list of child MapObjects of the Trial MapObject.

You will also notice in the MapObjects tree that PatientRecord has been added under Trial along with an \mathbb{R} , which means that this is actually a reference to a MapObject that is defined elsewhere. In our case, it is defined above as shown in Figure A.1.19.

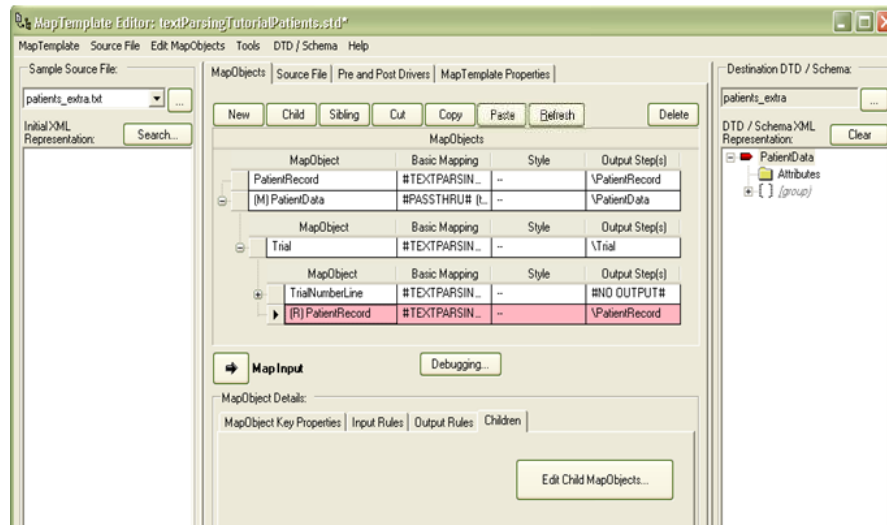


Figure - A.1.19

These two parts of the tree (PatientRecord at the top level with the normal MapObject, and PatientRecord located in the tree underneath Trial along with R) refer to the same underlying MapObject. You will notice that we have not set a **MinOccurs** or **MaxOccurs** value for this new child of Trial. In our source file, there can be any number of PatientRecords in a Trial. Let us set the **MinOccurs** to 1, to say that there must be at least one, and the **MaxOccurs** to unbounded, to say that there can be any number of these. Remember, you can do this by selecting the parent Object (Trial), and clicking **Edit Child MapObjects** under the **Children** tab.

Why the “MinOccurs” and “MaxOccurs” are set in the child MapObjects tab of the parent MapObject (Trial in this case) rather than being set in the PatientRecord MapObject itself? This is because we can reuse the PatientRecord MapObject by referring to it from another element; in that case the rules for how many times the PatientRecord can occur may be different.

Creating MapObject for the Line Parsing and Referring to Them

We are now going to create two MapObjects that we will refer from PatientRecord. We will define a PatientRecord as a sequence of these two objects:

- PatientInfoLine MapObject
- CommentsLine MapObject

This type of sequence is referred to as a GRAMMATICAL DEFINITION of the source input that is, if the parser finds a PatientInfoLine object followed by a CommentsLine object, then it has found a PatientRecord Object. If however, it only finds a PatientInfoLine object, or only a CommentsLine Object, then this doesn’t work, and the PatientRecord will not be found.

Let us create these two MapObjects.

First click the PatientRecord MapObject, and look at its input rules. They are still set to the default. This is fine; we will define the input rules after we have created the other two MapObjects: PatientInfoLine and CommentsLine.

Creating PatientInfoLine

These two new MapObjects should also be top level MapObjects, because they need to be referred to by the PatientRecord MapObject, that can be created using **New** at the top of the MapObjects grid.

Click the newly created MapObject. Change its name to PatientInfoLine, and its basic mapping type by selecting #TEXTPARSING#.

This MapObject will simply parse the first line of a patient record, and then pass it on to child MapObjects to do the actual parsing of the line into its various components (like PatientName and PatientGender).

Set the **Input Rule Type** list item (using **Edit Input Rules**) to “Simple Text”, and choose the **Line** input rule matchtype.

Notice that this MapObject is on the top level of the tree inside the grid rather than appearing underneath the PatientRecord. This is because we don’t want to pass the PatientRecord selection to this object; rather we want to use this object to find the PatientRecord selection. This will become clear when we go back, and define the input rules for PatientRecord in a moment.

First, let us finish up with the PatientInfoLine MapObject. Go to the **Output Value** tab, and set the output rule for the value of this element to “Passthru Input Text”.

Creating CommentsLine

Now similarly, create a CommentsLine top level MapObject with an input rule type of simple text, and the **Line** input rule selected. This can be done by **New** at the top of the MapObjects grid. This will add a new top level MapObject with a default name. Change the default name to CommentsLine, change the source type to input text by selecting “#TEXTPARSING#” under input rules, set the input rule type to "Simple Text", and set the rule matchtype to Line.

You might have noticed that in our output DTD, there is neither the <PatientInfoLine> element nor the CommentsLine MapObject, the actual purpose of CommentsLine MapObject is to produce an output element called <Comments>.

Once again, we are using a technique of using MapObjects as parsing objects, that may or may not appear in the output.

After you have created a top level CommentsLine Object, click **Edit Output Attributes/ Rules** under the **Output Rules** tab, and you will see the default view. Change the selection to **Output Tag Name Determination Rule**, choose the output rule called “Constant”, and type in the name we want it to output as `Comments` in the **Constant Output Value** box.

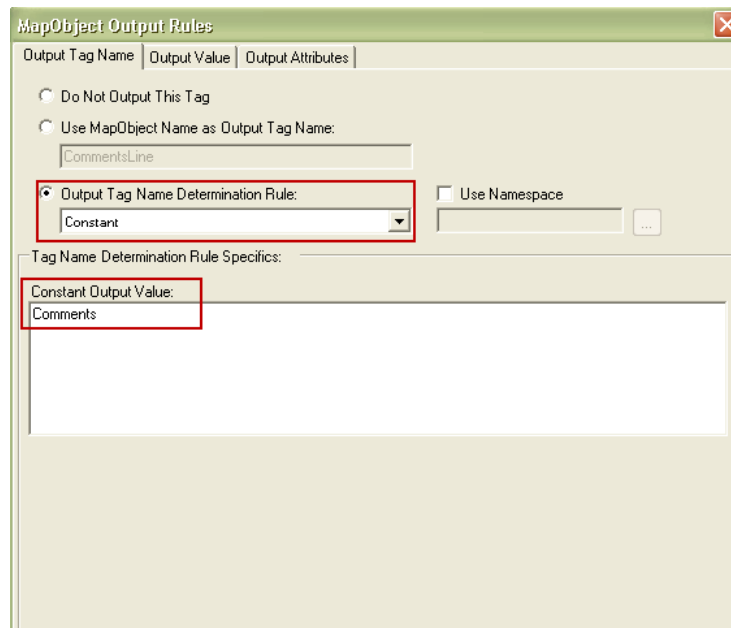


Figure - A.1.20

You will also set the **Output Value** of the CommentsLine object to “Passthru Input Text” (in the same way we set the Output value of the PatientInfoLine MapObject, by clicking the **Output Value** tab, and setting the output rule to "Passthru Input Text").


We are now ready to work with the input rules for the PatientRecord.

Set GRAMMATICAL Input Rules for the Patient Record

A GRAMMATICAL input rule set is different than the simple input rules, we have been using to date. We will now go back to the (R) PatientRecord MapObject, and select the “#TOKENGRAMMAR#” option for the basic mapping type of input rules. These are also explained in the *Text Parsing Rules* section of the *Arbortext Import Reference*.

There are no GRAMMATICAL rules defined yet. Remember that our definition of a PatientRecord is a SEQUENCE of PatientInfoLine Object, followed by a PatientComments Object.

A *SEQUENCE* is a type of GRAMMAR input rule that is used often in parsing documents.

For PatientRecord MapObject input rules, click **Edit Input Rules**, and select “Token Grammar” from **Input Rule Type** options. As a result you will see a section for **Grammar Hierarchy**, that is empty. If you click **Add Child Token Grammar Rule** , you will see that a single Token Grammar Rule of rule type SEQUENCE has been added to the **Grammar Hierarchy**. You will see that the **Rule Type** list option is automatically filled in as “SEQUENCE+”.


The “+” means that a sequence can have child elements. In fact, a SEQUENCE must have child elements because it is a sequence of something followed by something else.

Notice that there is a custom name box that lets you type a name for this grammar rule. You can leave it blank for now.

Now again, with SEQUENCE selected, click **Add Child Token Grammar Rule** again, and you will notice that a new child “Grammar Rule” is added. The type is again set to SEQUENCE+ by default, but this time we want to change it to the first item in our sequence. In the **Rule Type** list options, choose “STD OBJECT+”, because the first item in our sequence is going to be the MapObject called PatientInfoLine that we have created above. STD OBJECT is the old name for MapObjects, and it is still used internally by the Arbortext Import.

Notice that though we have set the first item in our sequence to be a MapObject, we have not yet defined which MapObject it is. Click (...) in **Rule Data**, and choose a MapObject from the dialog box that will be PatientInfoLine.

On your return to the input rule, you will see that the MapObject name has been filled in to the MapObject you just selected. It is also worth changing the **Custom Name** of this “Grammar Rule” to PatientInfoLine, so that it is shown in the rule tree. The **Custom Name** is a label used to refer to a specific “Token Grammar Rule”. In our sequence, for example, we might want to refer to either the first or second part of the sequence by name.

Now add another item in our sequence, by clicking **Add Sibling Token Grammar Rule** , and add another Grammar Rule that will be a child rule of SEQUENCE. Change the Rule Type to be STD OBJECT+, and choose the MapObject CommentsLine. You also want to give it a custom name that is descriptive, such as commentsline, as shown in Figure A.1.21.

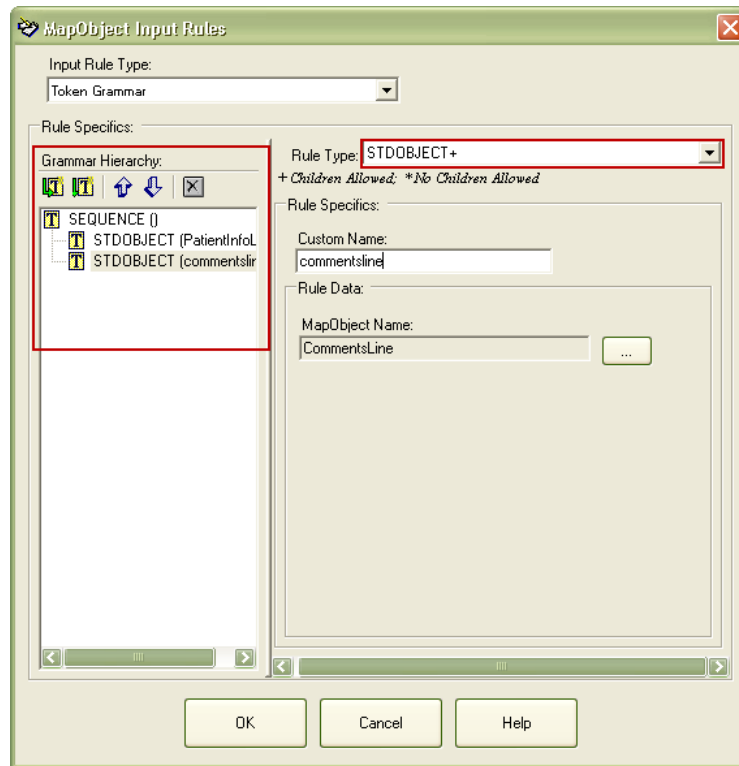


Figure - A.1.21

You have now defined the PatientRecord with a TOKEN GRAMMAR input rule to be the following:

- A SEQUENCE of two items
 - Item #1 in the sequence is a MapObject called PatientInfoLine
 - Item #2 in the sequence is a MapObject called CommentsLine

In addition to defining a TOKEN GRAMMAR input rule, you have also seen how to define top level MapObjects that can be referred to within other MapObjects: both in the MapObjects’s **Children** tab, and within “Token Grammar” input rules.

You will now save the template, and try out your grammatical rules to see how they work!

Running the Template with Patient Records

Now go to the main window of the Arbortext Import Workbench, and click **Run Transformation** again. When the **Transformation** window is appeared, you will notice that we have parsed the file into `<Trial>` and `<PatientRecord>` elements. The `<PatientRecord>` element has two children: `<PatientInfoLine>` and `<Comments>`. The template has successfully parsed out Patient Records. The transformation window shows the output as shown in Figure A.1.22.

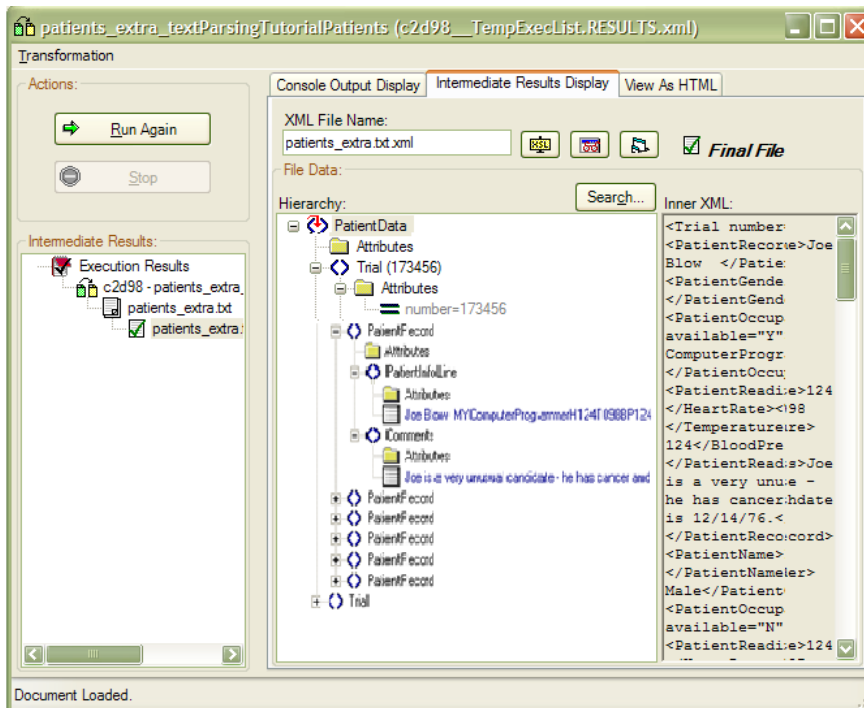


Figure - A.1.22

Create Remaining MapObjects for PatientInfoLine

If your template has successfully performed this parsing, then we are almost done. We now have to parse the `<PatientInfoLine>` element into the building blocks of a Patient Record:

- PatientName
- PatientGender
- PatientOccupation
- PatientReadings
 - HeartRate
 - Temperature

– BloodPressure

Because PatientInfoLine has already parsed out a line, we can simply pass this line down to the child MapObjects, that should be named after these elements.

Because we really do not need to reference these objects elsewhere, you can simply go to the PatientInfoLine MapObject, bring up its **MapObjects Children** window, and then create full instance of child MapObjects for PatientName, PatientGender, PatientOccupation and PatientReadings. Each of these has a minOccurs of 1 and a maxOccurs of 1, and you can change their names here in this window using **Child MapObject Name:** box, as shown in Figure A.1.23. Once they have been created and modified, you can click **OK**, and then edit the input rules for each one individually.

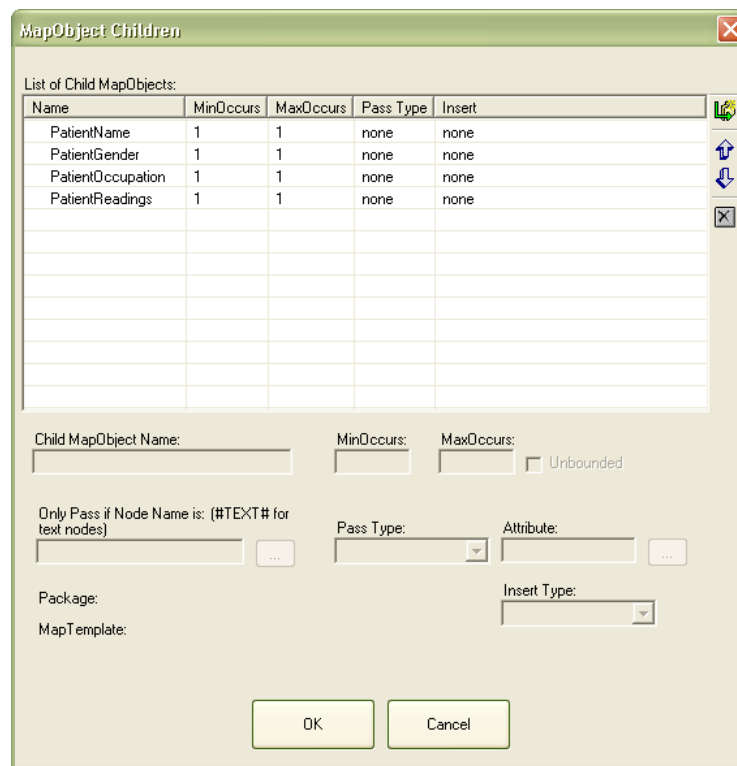


Figure - A.1.23

Now let us create the input rules for each of these MapObjects. You will have to change the basic mapping type to “#TEXTPARSING#” for each of them before bringing up the input rules editor.

The PatientName in our source file consists of the first 10 characters of the PatientInfoLine. The PatientName MapObject should be set to its basic mapping type by choosing “#TEXTPARSING#”, by applying **Input Rule Type** of “Simple Text”, and a matchtype of **Fixed Length** of **Char. Length** A.1.

The **Output Value** of the PatientName object should be set to “Passthru Input Text”.

The PatientGender is a single character of M or F. The Apply input rule should again be set to “Simple Text”, and the rule matchtype should be **Fixed Length** again, this time with a value of 1 character.

Now suppose, we wanted to map the M that occurs in the source file to the word Male in the output, and the F to the word Female. We can set an **Output Value** output rule to do this.

Start by clicking **Edit Output Attributes/Rules**, select the **Output Value** tab, and then select “Lookup Map” as the **Apply Output Rule:** type.

You can now type in the mappings of M to Male, and F to Female. After typing in the first line, press enter, and the cursor will be on the next line, as shown in Figure A.1.24.

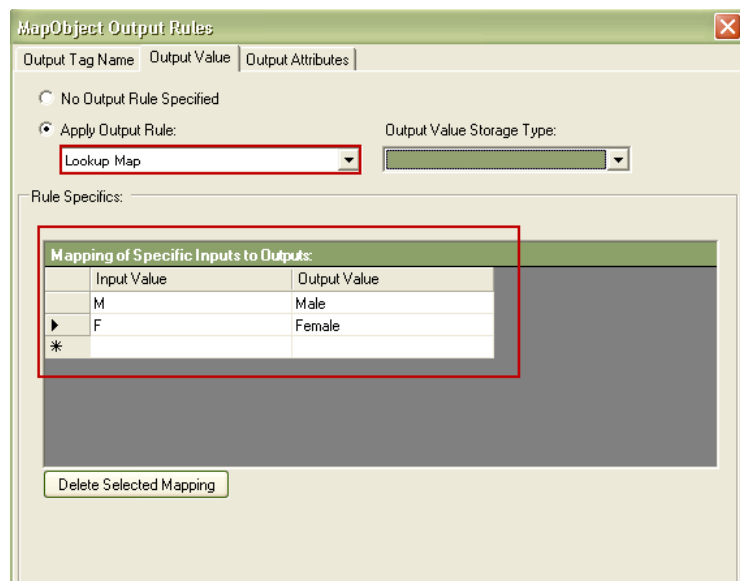


Figure - A.1.24

This is an example of how to map source text values into output values. This list can be as long as you want it to be. You can also delete specific mappings as needed.

The next MapObject, that is PatientOccupation is a bit tricky. In the source file, if the next character is Y, then the next few characters up to H are considered part of the occupation. If there is an N, then there is no occupation, and H is right after the N. This is an example of a conditional rule that is quite common in COBOL files and older files.

The following regular expression will produce this effect:

```
( [YN] ) ( [^\n] * ) ( H )
```

This regular expression is really a sequence of characters:

- A SEQUENCE of characters
 - Start with either a Y or an N ([YN])
 - Followed by any character that is not a new line, zero or more times ([^\n] *)

-
- Followed by H

So, for PatientOccupation, you should do the following:

- Set the source type to input text by selecting “#TEXTPARSING#” in the **Input Rules Basic Mapping** box, and click **Edit Input Rules**.
- Set the **Input Rule Type** list option to “Simple Text”.
- Choose the rule matchtype **Regular Expression**.
- Type the regular expression into the **Regular Expression** text box: ([YN]) ([^\n]*) (H)

You may wonder, how we can use a complex regular expression like this to get the appropriate values we need for our output element. Remember that the output element we want looks like this:

```
<PatientOccupation available="Y">Computer Programmer </PatientOccupation>
```

or, if there is no occupation, like this:

```
<PatientOccupation available="N"></PatientOccupation>
```

The key is the parenthesis in the regular expression, which defines sub-expressions. Our regular expression has three sets of parenthesis (called subexpressions or parens):

- First sub-expression: ([YN])
- Second sub-expression: ([^\n]*)
- Third sub-expression: (H)

You can select any of these sub expressions as the value for the **Output Value** or an **Output Attribute** of the current element. So, we should:

- For the Output value, get the value from the second parenthesis, that will be blank if there is no such value (**Apply Output Rule** should be set to “Input”, select **Regular Exp. Paren** and type in ‘2’).
- Create an Output attribute, called *available* that gets its value from the first parenthesis (Go to the **Output Attributes** tab of the **Output Rules Editor** dialog box, add a new attribute with **Output Attribute Name:** of “available”, and **Output Attribute Rule** of type “Input”, and then select **Regular Exp. Paren** and type in ‘1’).

In both cases, the output rule is set to type Input, and the Input is taken from a numerical sub-expression (identified by parenthesis #) of the regular expression.

The PatientReadings element can be parsed using the TOKEN GRAMMAR. If you examine the PatientReadings, as in the following example:

```
“Joe Blow MYComputerProgrammerH124T098BP124”
```

You will notice that the readings portion (H124T098BP124) is a sequence of characters that we could parse using one big regular expression, or we could parse using smaller

MapObjects, and using a “TOKEN GRAMMAR” rule of SEQUENCE. If we follow this approach, then PatientReadings consists of:

- A SEQUENCE of:
 - HeartRate
 - Temperature
 - BloodPressure

We would need to define top level MapObjects for HeartRate, Temperature and BloodPressure, and then define the PatientReadings input rules. You can do this as below:

- **HeartRate** — Create a new top level MapObject called HeartRate. Its basic mapping type should be “#TEXTPARSING#”, input rules type should be "Simple Text", and the matchtype should be "Regular Expression". The Regular Expression to use should be $(\d+)$, that is a sequence of digits. Notice that we don’t have H here, because H has already been consumed by the PatientOccupation MapObject. The Output value of the HeartRate object should be set to Input, and use a regular expression paren# 1.
- **Temperature** — Similar, except that regular expression should be $T(\d+)$, and output rule should also point to paren#1 of the regular expression.
- **BloodPressure** — Similar, except that regular expression should be $BP(\d+)$, and output rule should also point to parent#1 of the regular expression.

Now we can create the sequence in the PatientReadings MapObject. Set the basic mapping type by choosing “#TOKENGRAMMAR#”, then apply input rule type to "Token Grammar". Start with a Token Grammar SEQUENCE rule, along with three children, each of type STDOBJECT+, one for each MapObject. Also enter **Custom Names** for each of the items in the sequence. The method for doing this is identical to the step-by-step directions, we gave for creating the PatientRecord input rules, that consisted of a SEQUENCE of two other MapObjects.

The **Output Value** of the PatientReadings MapObject should be set to **No Output Rule Specified**, because all it does is to contain the output elements created by the three MapObjects: HeartRate, Temperature and BloodPressure.

Some Cleanup Items

If you save the MapTemplate, and run it again, you will see the results in the **Transformation** window. You will notice that we are almost done, but there are some discrepancies between this XML and our intended XML.

One thing we can do to find these discrepancies is to validate the resulting XML against a

DTD. To assign a DTD file, go to the **MapTemplate Properties** tab, and click **More MapObject Options**. This will display the advanced settings for the template.

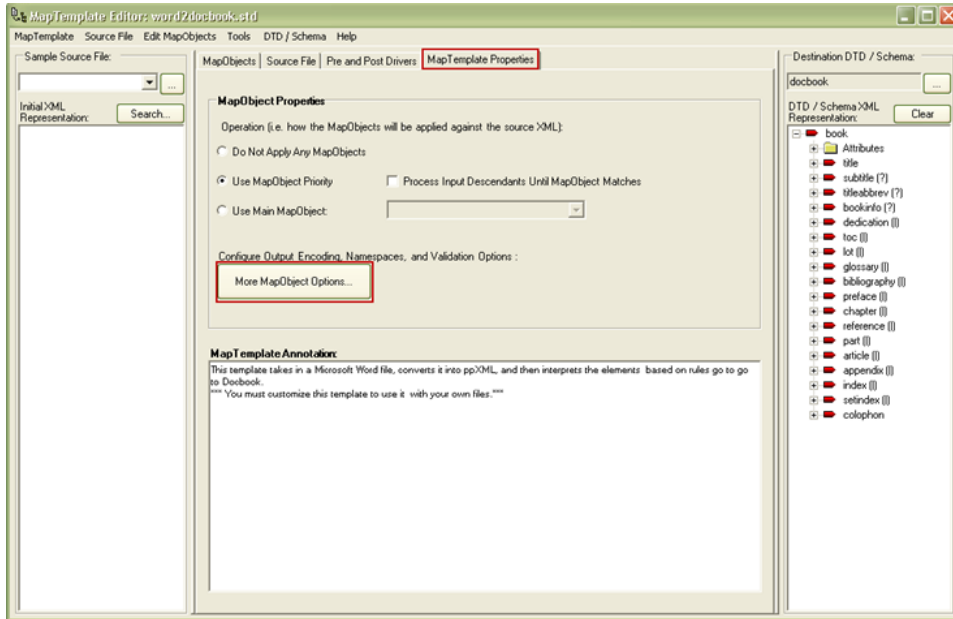


Figure - A.1.25

These settings show you how you can control the XML that is output. Under the **Validation** tab you will see the ability to enter a public system file for DTD or XSD (XML schema file) used to validate the resulting XML.

Select **Write Schema/DTD and Validate**, and under the **Validation Details** select **Document Type Definition (DTD)** as type, and click **Browse**, and then find the DTD file locally (**patients_extra.dtd**).

If you save and run the transformation, you will see a list of parsing errors in the console error window. In particular, the errors are related to:

- PatientInfoLine, which we used to get a line, but do not exist in our output DTD
- The Trial element shouldn't have any text nodes

We can fix these relatively quickly:

- Choose the Trial MapObject, and go to the **Output Value** tab. Set the option to **No Output Rule Specified**, that will get rid of the extra text under Trial.

- Choose the PatientInfoLine MapObject, and go to the **Output Tag Name** tab. Set the option **Do Not Output This Tag**. Also go to the **Output Value** tab, and set the option to **No Output Rule Specified**.

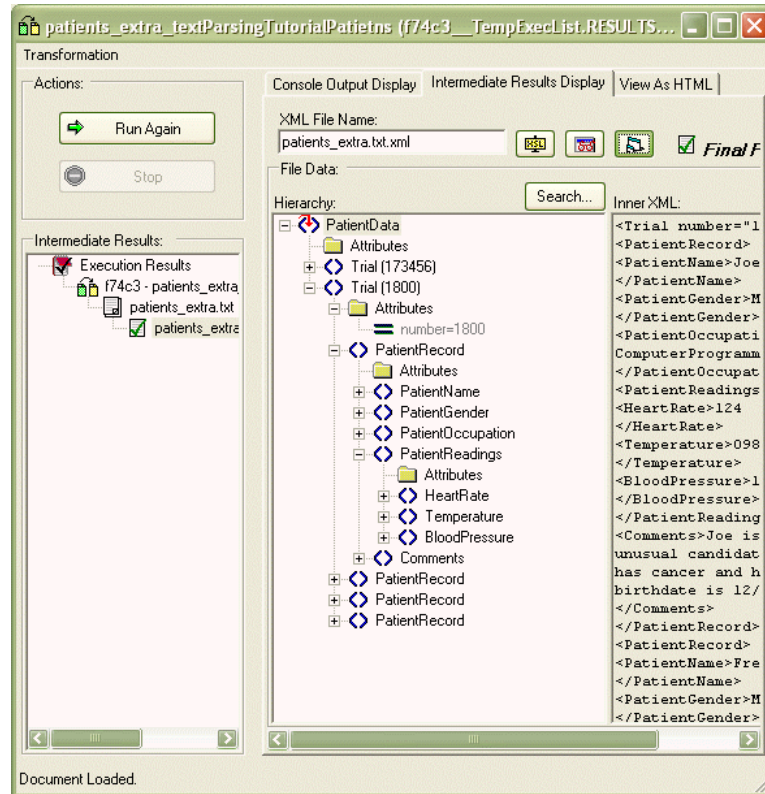


Figure - A.1.26

After resolving all validation parsing errors, you will have a valid XML document that conforms to the DTD, and have no errors. Final result will be as shown in Figure A.1.26.

Conclusion

This tutorial walked you through the process of creating a new MapTemplate, and using text parsing rules (both "Simple Text" and "Token Grammar" input rules) to parse a source file, and then convert it into meaningful XML. The source file was a sample included with the Arbortext Import, called **patients_extra.txt**, and the "meaningful XML" is defined by the DTD that also comes with the sample files.

How to Customize Configuration Files & Locations

Introduction

You can define customized variables and paths to reuse in multiple projects, transformations, and MapTemplates.

Variables can be used for relative paths of MapTemplates during transformations, and can also be used to call child MapObjects. Rather than defining the absolute path of MapTemplates, user will be able to define the variable which points to directory containing MapTemplates, and then eventually use that variable to access the particular MapTemplate. If the location of the MapTemplates changes, the user just needs to update the value of the variable rather than updating the paths from the multiple projects, transformations and MapObjects.

Paths can be used to locate the MapTemplates during transformations. If the path of MapTemplates directory is defined, then you will only provide the name of MapTemplate during the transformation rather than providing the complete path. You will be able to define multiple paths. If MapTemplates with same name are located in more than one path, then the MapTemplate located at first instance will be used for transformation.

How Edit Config File works

You can use this feature by selecting the option **Edit Config File** within the **Tools** menu. This feature uses an XML file to save variables and paths. This XML file is created as `<home>\STDTemplates\config.xml`.

This **Config.xml** file contains all the information about the variables and paths. You can edit the **Config.xml** file using the **Edit Config File** window, or simply by opening this **.xml** file in edit mode.

When you select **Edit Config File** from the **Tools** menu, a window will open as shown in Figure A.2.1.

Edit Config File Interface Overview

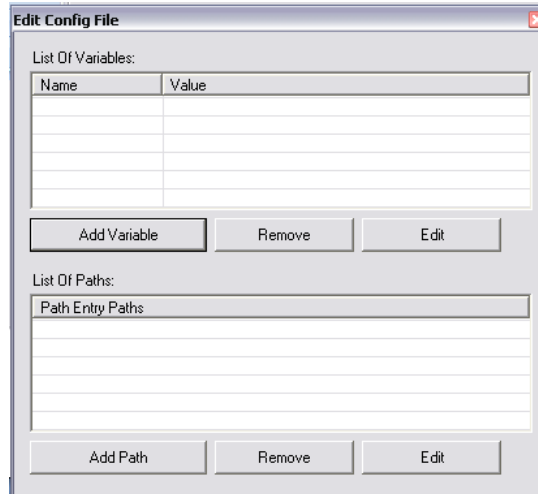


Figure - A.2.1

The **Edit Config File** window contains two sections: **List Of Variables** and **List Of Paths**.

List Of Variables contains the list of all variables defined in that **config.xml** file. The name of the variable is shown under **Name**, and its value is shown under the **Value**. The following buttons are available in this section:

- **Add Variable** is used to define a new variable.
- **Remove** is used to remove the selected variable.
- **Edit** is used to edit the selected variable.

List Of Paths: contains the list of all paths defined in that **config.xml** file. The following buttons are available in this section:

- **Add Path** is used to define new path.
- **Remove** is used to remove the selected path.
- **Edit** is used to edit the selected path.

How to use Edit Config File

With the help of the **Edit Config File** window, you can add variables and paths easily.

Adding Variables to the file

Use the following steps to add a variable:

1. Click **Add Variable**. A small window similar to the highlighted one in Figure A.2.2 opens.

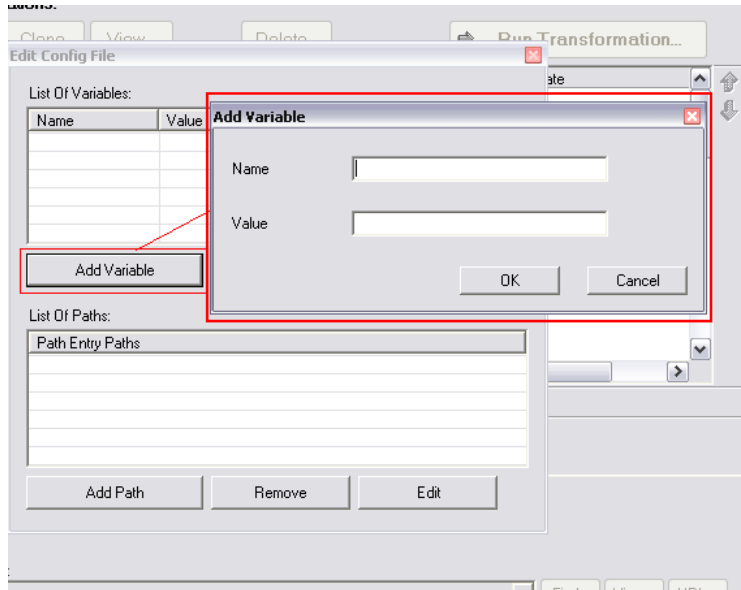


Figure - A.2.2

2. Enter a variable name in **Name**, and value in **Value**. (Value will be the path of directory which contains MapTemplate)
3. Click **OK**.
4. You will be back on the main **Edit Config File** window, where you will see the newly added variable under the **List Of Variables**.
5. Click **Cancel**, if you don't want to add a variable. Doing so will clear both the name and value, and you will be back to the main **Edit Config File** window.
6. Now close the **Edit Config File** window.
7. On the main Arbortext Import Workbench window, type in MapTemplate as **% Name of Variable%\Name of MapTemplate**.

Example:

Suppose you have a MapTemplate **abcTemplate.std**, saved in temp directory located at **E:\temp**, and you want to have a variable named as **var1**. For the variable name you will type **var1**, and for the value you will type **E:\temp**, and click **OK**. Close the **Edit Config File** window.

Now on the main Arbortext Import Workbench window, set the value in MapTemplate to **%var1%\abcTemplate**

When you run the transformation, Arbortext Import transformation engine will resolve the variables value, and locate the MapTemplate accordingly, and then execute the transformation.

Using this method you can define many variables, and use them within the main window during transformations by giving their names and MapTemplates.

Adding Paths to the file

Use the following steps to add a path:

1. Click **Add Path**, it will open a small window as shown in Figure A.2.3.

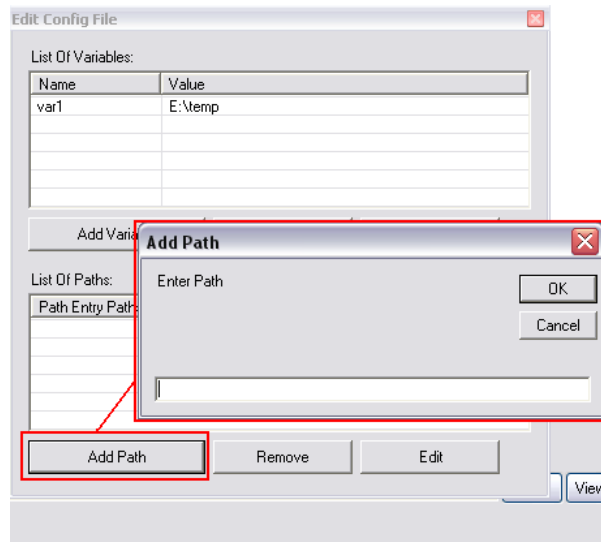


Figure - A.2.3

2. Type the required path in **Enter Path**, and click **OK**.
3. You will be back on the main **Edit Config File** window, where you will see the newly added path under the **List Of Paths**.
4. Click **Cancel**, if you don't want to add path. It will clear the path, and you will be back to the main **Edit Config File** window.
5. Now close the **Edit Config File** window.
6. On the main Arbortext Import window, type in the MapTemplate name.

Example:

Suppose you have a MapTemplate **abcTemplate.std**, saved in temp directory located at **E:\temp**, and you want to add its path. For the **Enter Path** value, you will type **E:\temp** and click **OK**. Close the **Edit Config File** window. Alternatively, you can also type a variable name, for example **%var1%** in the path dialog box. The variable needs to be defined before using it in paths.

Now on the main **Arbortext Import Workbench** window, you can provide the name of the MapTemplate. For example, **abcTemplate** instead of the complete path.

When you run the transformation, the Arbortext Import transformation engine will search the MapTemplate on the available paths, and execute the transformation.

Using this method you can define multiple paths, and use them within the main window during transformations. In case of multiple paths, the transformation engine will check for the specified MapTemplate in all the paths starting from first to last, and stop searching where it finds that template, and complete the transformation.

 **Note**

*While typing the MapTemplate name in the main **Arbortext Import Workbench**, don't enter it with `.std` postfix; just give name of MapTemplate without `.std`.*

Index

A

add child XML rule, 103, 119
Advanced Details Tab, 36

B

Basic Mapping Type, 57, 94, 118, 124,
127, 137–138, 142, 147, 151, 153, 155,
158, 161

C

Children Tab, 69, 104, 126, 145, 152
Console Output
 Display, 24, 78
contacting technical support, 6

D

Destination DTD, 48, 90, 136
DocBook Templates
 Customizing DocBook Templates, 88
Driver Options, 48, 50, 54–55, 92, 111

E

Edit Child MapObjects, 141, 145, 152

F

Field Code, 123

H

HTML View

Display, 29, 78

I

Inclusion Pattern, 143
information resources, 5
input rules basic mapping, 68, 118, 137,
147
Input Rules basic mapping, 160
Intermediate Result
 Display, 26–27, 78, 93, 144

M

Map Input, 59, 96, 111, 117, 124
Map Templates
 DocBook MapTemplates, 82
MapObject Rules
 Input Rules Output rules, 86
MapObjects
 Creating MapObjects, 58
 Customizing the List MapObjects, 111
MapTemplate Editor
 Pre and Post Drivers, 52–53, 84, 137
MapTemplates
 DocBook MapTemplates, 29, 83

N

New Child, 140, 145, 152
New Project, 18

O

Output Hierarchy, 87, 139

P

ppXML, 11, 23

Pre and Post Options

- Include Lists, Use CALS table Model, 92

Pre-Processed XML, 11

product support contact information, 6

R

resources for more information, 5

S

Source Element Attribute, 125

Source File, 20

support contact information, 6

T

Transformation

- Run Transformation, 19

Transformation Object

- Run Transformation Object, 23

Transformation Wizard, 19

X

XML Rule Hierarchy, 102, 119