



I N T E R W O V E N

**TeamXML<sup>TM</sup>**  
**Administration Guide**  
**Release 5.5**

Copyright 2002 Interwoven, Inc. All rights reserved.

No part of this publication (hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Interwoven. Information in this manual is furnished under license by Interwoven, Inc. and may only be used in accordance with the terms of the license agreement. If this software or documentation directs you to copy materials, you must first have permission from the copyright owner of the materials to avoid violating the law, which could result in damages or other remedies.

Interwoven, TeamSite, OpenDeploy and the logo are registered trademarks of Interwoven, Inc., which may be registered in certain jurisdictions. SmartContext, DataDeploy, Content Express, OpenChannel, OpenSyndicate, MetaTagger, MetaSource, TeamCatalog, TeamXML, TeamXpress, the tagline and service mark are trademarks of Interwoven, Inc., which may be registered in certain jurisdictions. All other trademarks are owned by their respective owners.



**I N T E R W O V E N**

Interwoven, Inc.

803 11th Ave.

Sunnyvale, CA 94089

<http://www.interwoven.com>

Printed in the United States of America

Release 5.5.2

Part # 40-00-10-14-00-552-300

# Table of Contents

## About This Book 9

- Intended Audience 9
- Overview of Manual 9
- Documentation on the TeamXML CD 10
- Other Documentation 10
  - Sample DTDs and Category Definition Files 10
  - Tibco XML Authority 10
  - Third-Party Integrated XML Editors 11
  - Readme Files 11
  - Information Available on the Web 11
  - OpenAPI Documentation 11
- Typographical and Notation Conventions 12
  - Typographical Conventions 12
  - Notation Conventions 13

## Chapter 1: Introduction to TeamXML 15

- Overview of TeamXML Functionality 15
  - Incorporating Files into TeamXML 16
  - Componentizing a Document – An Example 16
  - Definitions 18
    - Document 18
    - Component 18
    - Category 19
- Advantages of Componentized XML Files 20
- TeamXML as Part of TeamSite 21
  - End-User Perspective 22
- Detailed Product Description 23
  - Documents 23
  - Components 24
  - Categories 24
  - Searching 25
  - Mutation 26

## **Chapter 2: Installation and Initial Configuration 27**

TeamXML Software Components	28
Hardware and Software Requirements	28
Server Hardware	29
Disk Space	29
Number of CPUs	29
Memory	31
Inode requirements (UNIX systems only)	31
Disk Configuration	31
Server Software	32
Search Engine and TeamSite Server on Same System	32
Search Engine and TeamSite Server on Separate Systems	33
Client Hardware	33
Client Software	34
Software Component Locations	35
TeamXML System Files	35
XML Search Engine	35
Default File Locations	36
Tibco XML Authority	37
Integrated XML Editors	37
Pre-Installation Tasks	37
Installation Steps	38
Installation Task Overview	38
Search Engine and TeamSite Server on Same System	38
Search Engine and TeamSite Server on Separate Systems	39
Installation Task Details	39
Installing on a Windows System	39
Installing on a UNIX System	47
Post-Installation Tasks	52
Run iwreset	52
Install XML Authority	52
Install an Integrated XML Editor	53
Set Up Indexing	54
Indexing TeamXML Files	54
Indexing Non-TeamXML Files	54

Create Directory Structure and Sample Categories	55
Directory Structure Details	56
Populating the Directory Structure	57
Rules Directory	57
Document and Component Directories	57
Importing Existing XML Content	58
Verify Configuration Settings	58
Verify XML Search Engine Startup	58
Install Sample Content Manually	59
Configure the TeamSite Server	60
Uninstalling TeamXML	60
Windows Systems	60
UNIX Systems	61
Log Files	61
Additional Considerations	61

## **Chapter 3: Category Definition Files 63**

Guidelines for Creating Categories	64
Creating Document Categories	65
Document Characteristics	65
Creating Component Categories	66
Component Characteristics	67
Components and Reuse	68
Parts of a Category Definition File	71
Root Element and Category Display Name	71
Root Element Rule Set	72
Componentization Rule Set	73
Entity References	74
Excluding Elements from Componentization	74
Adding and Deleting Componentization Rules	74
How Componentization Rules are Applied	74
Component Display Name Rule Set	76
Indexing Rule Set	76
External Validation Specification	79
User-Defined Document Category Description	80
Component Directory Name	80

Prolog Section	81
Prolog Section Purpose	81
prolog Section Contents	82
prolog Section DTD	82
OpenAPI Support for prolog	82
prolog Example	82
XPath Conventions	83
Sample Category Definition Files	85
Document Category Definition Files	86
Component Category Definition Files	86
Sample File Details	86
Document Category Definition File for Marketing Requirements Example	87
Sample File Notes	89
Plain XML File for Marketing Requirements Example	91
Component Category Definition File for Products	96
Sample File Details	98
Generated Product Component File	101

## **Chapter 4: Search Statements 103**

Search Overview	103
Search Tips	104
Sample Searches	105
listwa.xml	106
find-jpegs.xml	107
available-products.xml	108
simple-ea.xml	109
complex-ea.xml	111

## **Appendix A: Command-Line Tools 113**

iwxmlcat	114
iwxmlfile	116
iwxmlindex.ipl	118
iwxmlinfo	120
iwxmlsearch.ipl	125

## **Appendix B: DTDs 127**

- Category Definition File DTDs 127
  - DTD for Document Category Definition Files 128
  - DTD for Component Category Definition Files 129
  - DTD for Indexing Rule Set 130
- Search DTD 132

## **Glossary 133**

## **Index 139**





# About This Book

---

This chapter contains the following topics:

- Intended Audience
- Overview of Manual
- Documentation on the TeamXML™ CD
- Other Documentation
- Typographical and Notation Conventions

## Intended Audience

Readers of this book should have knowledge of TeamSite® administrative tasks and advanced knowledge of XML, including DTDs.

## Overview of Manual

This book contains release notes about TeamXML software. This book discusses the following topics:

- **About This Book**—Provides an overview of this book's structure and intended audience; references to additional documentation about TeamXML or products that work with TeamXML; and the typographical and notation conventions used in this book.
- **Chapter 1: Introduction to TeamXML**—Provides an overview of TeamXML.
- **Chapter 2: Installation and Initial Configuration**—Provides installation instructions and recommendations for initial TeamXML directory structure and server configuration.
- **Chapter 3: Category Definition Files**—Provides a description of the parts of a category definition file and sample category definition files.
- **Chapter 4: Search Statements**—Describes search service queries.

- **Appendix A: Command-Line Tools**—Provides a complete description of the command-line tools (CLTs) that are part of TeamXML.
- **Appendix B: DTDs**—Provides a printed version of the DTDs that are part of TeamXML.
- **Glossary**—Provides definitions of TeamXML terms.
- **Index**—Provides cross references to TeamXML topics.

## Documentation on the TeamXML CD

The following manuals are available in the docs directory on the Interwoven® TeamXML CD-ROM:

- *TeamXML User's Guide*
- *TeamXML Administration Guide*
- *TeamXML Release Notes*

## Other Documentation

This section describes other documentation that is relevant to TeamXML.

### Sample DTDs and Category Definition Files

The TeamXML installer will install DTDs and example XML files, including category definition files, in the location shown in “Default File Locations” on page 36.

### Tibco XML Authority

Tibco XML Authority® is a schema and DTD development tool that allows creation and editing of category definitions in a graphical environment. This product has been modified to understand and work with TeamXML. This product and its online documentation are bundled with TeamXML. See “Install XML Authority” on page 52 for more information.

## **Third-Party Integrated XML Editors**

Documentation about third-party integrated XML editors that understand TeamXML is available separately, when those editors are downloaded. In this context, “integrated” does not mean that an editor is bundled with TeamXML; it only means that the editor has an awareness of TeamXML and has been modified to work with it.

## **Readme Files**

README files, located in many of the installed directories, provide additional examples or information on available features.

## **Information Available on the Web**

Current documentation and any available new information is located at:  
<http://support.interwoven.com>

## **OpenAPI Documentation**

OpenAPI documentation (PDF files and JavaDoc output) is available with the OpenAPI Software Developer’s Kit. Contact an Interwoven representative for information about downloading the SDK.

# Typographical and Notation Conventions

## Typographical Conventions

This book uses the following typographical conventions:

Convention	Definition and Usage
<b>Bold</b>	Text that appears in a GUI element (for example, a menu item, button, or element of a dialog box) and command names are shown in bold. For example:  Click <b>Edit File</b> in the Button Bar.
<i>Italic</i>	Book titles appear in italics. Terms are italicized the first time that they appear. Important information may be italicized for emphasis.
Monospace	Commands, command-line output, and file names are in monospace type. For example:  The <code>iwextattr</code> command-line tool allows you to set and look up extended attributes on a file.
<i>Monospaced italic</i>	Monospaced italics are used for command-line variables. For example:  <code>iwckrole <i>role</i> <i>user</i></code>  This means that you must replace <i>role</i> and <i>user</i> with your values.
<b>Monospaced bold</b>	Monospaced bold represents information you enter in response to system prompts. The character that appears before a line of user input represents the command prompt, and should not be typed. For example:  <code><b>iwextattr -s project=proj1 //IWSERVER/default/main/dev/WORKAREA/andre/products/index.html</b></code>

Convention	Definition and Usage
<b><i>Monospaced bold italic</i></b>	<p>Monospaced bold italic text is used to indicate a variable in user input. For example:</p> <p style="text-align: center;"><b>% iwextattr -s project=<i>projectname</i> workareavpath</b></p> <p>means that you must insert the values of <i>projectname</i> and <i>workareavpath</i> when you enter this command.</p>
[ ]	Square brackets surrounding a command-line argument mean that the argument is optional.
	Vertical bars separating command-line arguments mean that you can use only one of the arguments.

## Notation Conventions

### *Notation Conventions for Directory Paths*

This manual uses UNIX conventions for directory paths. These conventions mandate using forward slashes (/).

For example:

```
docroot/news/front.html
```

On a Windows system, you would type such a path with backward slashes:

```
docroot\news\front.html
```

This manual only uses Windows conventions for paths when referring to a Windows-specific directory.

### *iw-home Notation on Solaris and Windows Systems*

This manual does not use the Solaris™ notation (*iw-home*; note the lack of italics) except when specifically referring to procedures performed only in Solaris.

This manual uses the Windows version of *iw-home* notation (*iw-home*) when discussing both Solaris and Windows NT or Windows 2000 systems. The italics are an Interwoven convention

identifying *iw-home* as a variable. You should interpret the *iw-home* notation used in this manual as follows:

- On Solaris systems, *iw-home* is the literal name of the directory containing the TeamSite program files.
- On Windows systems, *iw-home* is the symbolic name of the directory that contains your TeamSite program files. The default value of *iw-home* on Windows systems is:

`C:\Program Files\Interwoven\TeamSite`

The administrator performing Windows installation may have chosen an installation directory different from the default.

## Chapter 1

# Introduction to TeamXML

---

TeamXML is a subsystem of the TeamSite server that intelligently manages XML-formatted data. XML provides a syntax for text representation of structured data. TeamXML, by taking advantage of XML's structured nature and in conjunction with TeamSite, creates a powerful system for intelligent management and reuse of XML data.


## Overview of TeamXML Functionality

With TeamXML, ordinary XML files can be converted into *componentized* XML documents. Such files become part of TeamXML as a result of this componentization process.

The process of incorporating ordinary XML files into TeamXML involves placing the XML file into a TeamXML *category*, optionally breaking the file into discrete sections (called *components*), and placing the components into additional TeamXML categories. Categories may be thought of as logical containers, each containing related documents or components. Category assignments and componentization specifics are based on rules that you define for your site and place in *category definition files*. After a file is componentized, you can manipulate each component individually. For example, you can use the same component in multiple documents or change the order in which components appear in a document.

Category definition files contain more than just componentization rules—they also contain rules for other aspects of TeamXML behavior. Category definition files are an essential part of TeamXML, and are explained in detail in Chapter 3, “Category Definition Files.” As with ordinary XML files, category definition files must be incorporated into TeamXML. The processes for incorporating both ordinary XML files and category definition files into TeamXML are described in the following section.

## Incorporating Files into TeamXML

After you have installed and configured TeamXML, you convert ordinary XML files into componentized TeamXML files using the `iwxmlfile` CLT. This process is called *componentizing* throughout this document. Componentized documents appear in the TeamSite GUI with this icon: . See the *TeamXML User's Guide* for an example of the TeamSite GUI containing componentized TeamXML documents. See “`iwxmlfile`” on page 116 for details about using the `iwxmlfile` CLT.

In some cases, it might be appropriate to make TeamXML aware of ordinary files without breaking the files into components. This process is referred to as *specialization* throughout this document.

Category definition files, which are shown throughout this document with a `.dfn` file name extension, are incorporated into TeamXML with the `iwxmlcat` CLT. Note that category definition files are specialized, not componentized, because they are not split into components when incorporated into TeamXML. After specialization, they also appear in the TeamSite GUI with the TeamXML file icon. See “`iwxmlcat`” on page 114 for details about using the `iwxmlcat` CLT. See “Category Definition Files” on page 63 for details about creating category definition files.

**Note:** Category definition files are not required to have a `.dfn` file name extension, but it is recommended that you use this convention.

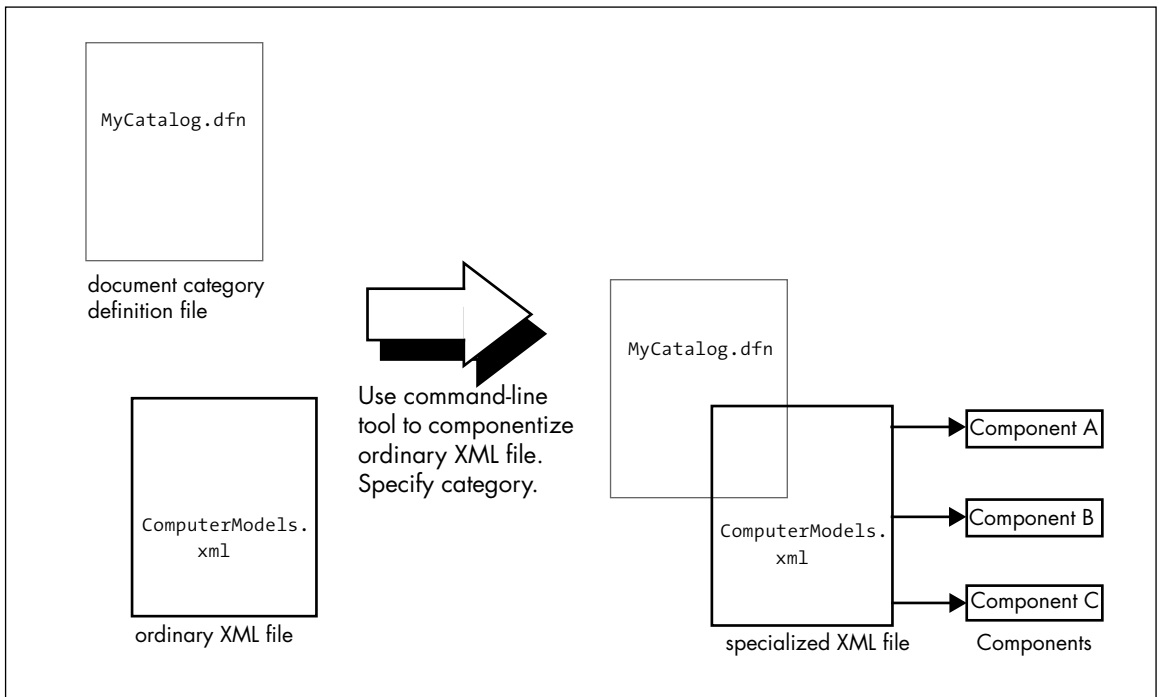
## Componentizing a Document – An Example

The following figure shows the componentization process, during which:

- An ordinary XML file is converted to one recognized by TeamXML.
- The XML file (now a TeamXML document) is bound, or linked, to a category and thus to a document category definition file.
- The TeamXML document is componentized using the componentization rules in the category definition file.

This example assumes that the category definition file, `MyCatalog.dfn`, has already been specialized. The XML document inherits other properties from its category definition file, including display name rules and indexing rules.





*Componentization Process—Converting an Ordinary XML File to a TeamXML Document*

In this figure, assume that Andre at the Acme Corporation has a file named `ComputerModels.xml` to bring into TeamXML. He wants to put all of the information from `ComputerModels.xml` into a category that will act as a logical container for the information. The `MyCatalog.dfn` file is the document category definition file that defines the category and contains other rules that might specify, for example, that TeamXML should create various components based on the content from `ComputerModels.xml`. These components are labeled Component A, B, and C in the diagram. The componentization process shown here is executed with the `iwxmlfile -c` command. An integrated third-party editor may also be used componentize a file when the file is saved. `ComputerModels.xml` becomes a TeamXML document and is linked to its category definition file.

The componentization process may be more complex than is shown in the preceding figure. The components shown—components A, B, and C—may also be componentized, using a different category definition file for each component type.

For example, suppose that a rule in `MyCatalog.dfn` specifies the creation of a component for every `<product>` element that is a subelement of the `<catalog>` element. After Andre executes the `iwxmlfile -c` command, TeamXML creates components for such `<product>` elements and assigns them to the Product category, which is defined by a component category definition file, `Products.dfn`. This file has its own rules, which cause the server to create a component from every `<vendor>` element that is a subelement of the `<product>` element. These components are assigned to the Vendor category. The Vendor category has a category definition file, `Vendor.dfn`, that causes conversion to stop because, in this example, its rules specify that Vendor components do not have subcomponents.

## Definitions

To understand this process more precisely, the terms *document*, *component* and *category* need to be defined.

### Document

A TeamXML *document* is an XML file that has been componentized. The main distinction between a document and an ordinary file is that portions of the document, known as components, may in fact be stored in other, separate files and are included in the document when it is read. A document represents a complete, standalone unit while other components typically represent portions of a document. A document cannot be contained by another TeamXML object such as a component or document. A document can contain any number of components.

### Component

TeamXML allows you to create new *components* through componentization, that is, dividing a document or component into component parts.

By dividing documents into components, you can:

- Reuse components in other documents.
- View and modify components independently of documents.
- Edit components directly inside containing documents.
- Allow parts of documents (components) to be distinct objects in terms of ownership.

Unlike documents, components can be located inside another TeamXML object (such as a component or document). Components may contain other components; a nested component is called a subcomponent. A given component can be contained by any number of documents and/or components. Any change made to a component is reflected immediately in all of the documents and components that contain it and that reside in that TeamSite area.

Component path names are not specified by the user because most components are extracted from documents and have no obvious name. Because they have no obvious name, it is not always easy to browse the file system to locate components. To allow fast access to components and other files, TeamXML provides a powerful search engine.

In practice, a document represents a complete, standalone unit whereas components typically represent portions of a document. Even though every component is an XML element, it is not necessarily true that every element in a stream of XML-formatted data is a component. The elements that TeamXML will convert into components are defined by the componentization rules.

## **Category**

A *category* is a set of related documents or components in TeamXML. The members of a category all share a common category definition. There are two types of categories: document categories and component categories.

When componentizing an XML document, you specify its category. A category has properties and rules that you have defined in a category definition file.

A document or component can belong to exactly one category, as defined by a category definition file. Componentizing a document causes the TeamSite server to execute the componentization rules specified in a category definition file. These rules specify the XML elements that the TeamSite server should convert into components. You can further componentize components to create subcomponents.

See “Glossary” on page 133 for more detail about these or other terms.

## Advantages of Componentized XML Files

After you have converted an ordinary XML file into a componentized XML document, the TeamSite server's TeamXML subsystem allows you to use the document in specific ways. Due to its understanding of XML as structured data, TeamXML can:

- Treat componentized XML files as documents that contain components, that is, reusable parts.
- Treat components as separate objects for purposes such as versioning or ownership.
- Group documents and components into categories.
- Make components available for reuse in other documents.
- Create components that contain other components.
- Automatically update documents so that they always contain the latest version of components.
- Render complete documents from components.
- Compose new documents that include existing components.
- Provide tools for indexing and querying components to facilitate component sharing.

Through its understanding of XML documents and components, TeamXML allows collaboration and reuse of documentation. Different users can contribute components to a document. Because XML components are treated as separate objects, organizations can share and reuse these components in many different documents. A single change to a component is reflected in all documents that contain this component.

In summary, by making components available for reuse in other documents, you can:

- Facilitate collaboration and sharing of resources.
- Ensure consistency in your documentation and Web assets.
- Implement changes rapidly.
- Save storage space.
- Save time across your enterprise.

At the same time, these opportunities pose new content management challenges. To take advantage of XML, your organization must organize, share, version, and search XML data effectively.

## TeamXML as Part of TeamSite

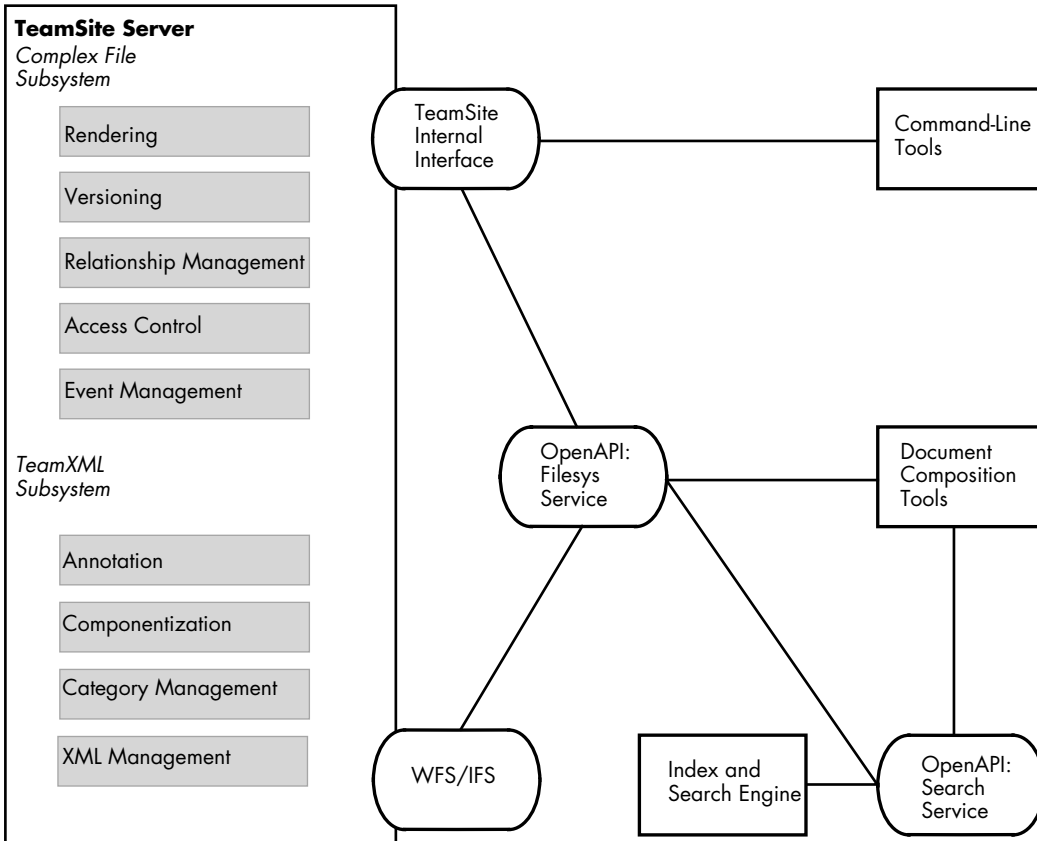
Before the addition of the TeamXML subsystem, TeamSite treats ordinary XML files like any other file in the TeamSite backing store, but it has no deeper understanding of XML beyond that.

TeamXML enhances TeamSite's understanding of XML. Because TeamXML is fully integrated with TeamSite, your organization can take advantage of its knowledge of TeamSite as well as TeamSite's abilities to version and manage Web assets.

TeamXML provides the following architectural features:

- The TeamSite server, `iwserver`, provides core content-management functionality.
- CLTs allow you to index, search, obtain information about, and manipulate XML documents and components.
- An application programming interface through OpenAPI gives access to all TeamXML functions, thus providing a mechanism to programmatically search and manipulate XML content through Java programs.
- A search engine, which is integrated with TeamXML but which processes data separately from `iwserver`.
- Standalone third-party XML and schema editors, integrated with TeamXML through OpenAPI, which provide GUIs to assist in various tasks:
  - Tibco XML Authority™ is a schema and DTD development tool that provides a graphical environment for creating and editing category definition files. The version of XML Authority on the TeamXML CD has been modified to work with TeamXML.
  - Integrated XML editors such as SoftQuad XMetaL™ and Arbortext Epic Editor™ allow document and component editing. The integrated XML editors are not bundled with the TeamXML CD, but can be downloaded from the vendor's Web site. A TeamXML adapter is available with each integrated XML editor. The adapter ensures that the integrated XML editor has an awareness of TeamXML and can provide a TeamXML-specific graphical environment for end users.

The following figure shows how TeamXML services are integrated with the TeamSite server:



*TeamXML Architecture*

## End-User Perspective

From the end-user's perspective, TeamXML is integrated with TeamSite in the following ways:

- You store TeamXML files in TeamSite directories.
- You have separate directories for staging, workareas, and editions because TeamXML uses the TeamSite interface.

- Even though TeamXML documents and components are accessible from the TeamSite file system, they are read-only when accessed in this way. Further, you should access TeamXML components through an integrated XML editor because components have system-generated file names that are difficult to interpret unless viewed through an integrated editor.
- TeamSite versions XML documents and components, and all file properties—such as owner, permissions, and so on—like other TeamSite content.
- TeamSite rules are applicable—for example, a document in a workarea cannot contain a reference to a component in an edition or in the staging area.

See the *TeamXML User's Guide* for more information specific to end-users.

TeamXML enhances this functionality by allowing TeamSite to treat componentized XML files in new ways, as described in further detail in the following section.

## Detailed Product Description

This section provides additional detail about important TeamXML features. It discusses the following topics:

- Documents
- Components
- Categories
- Searching
- Mutation

### Documents

TeamXML allows you to:

- Convert XML files to componentized XML documents or components.
- Create new documents from existing components.
- Access XML documents on a read-only basis through the file system.
- Use custom tools for read/write access to documents.
- Automatically update documents so that they always contain the latest version of components.

When documents are retrieved through the API, TeamXML uses XML processing-instruction annotations to identify components embedded in documents.

## Components

TeamXML allows you to create new components through componentization, that is, dividing a document or component into component parts.

By dividing documents into components, you can:

- View and modify components independently of documents.
- Edit components directly inside containing documents.
- Allow parts of documents (components) to be distinct objects in terms of ownership, and so forth.

TeamXML also allows you to:

- See component changes immediately in all containing documents.
- Identify components according to rules that you specify.
- Store components in category-specific directories.
- Create components that contain other components.

**Note:** In addition to creating components from TeamXML documents or other components, you can also create components directly from ordinary XML files. See “Components and Reuse” on page 68 for more information.

## Categories

Categories are an important organizational feature of TeamXML. TeamXML arranges both documents and components into categories. These categories:

- Provide a convenient grouping for related documents and components.
- Provide a mechanism to identify child components.

When componentizing an XML document, you specify its category. A category has properties and rules that are described in detail in Chapter 3, “Category Definition Files.”

You can create nested components—that is, a component that contains subcomponents, or at least is permitted to contain subcomponents according to its category definition.



After you have created your category definition files, you can view them through the file system, which provides read-only access. If you need to edit them, it is recommended that you do so with XML Authority, although editing is also possible through command-line tools.

## Searching

Component path names are not specified by the user because TeamXML programmatically generates them. Because components have no obvious names, it is not easy to browse the file system to locate components. To allow fast access to components and other files, TeamXML provides a powerful search engine.

Fast searches of components let you:

- Easily reuse components rather than having to repeat work.
- Ensure that you can find data when you need it.

The search engine processes data separately from `iwserver`. Search criteria can contain a mixture of metadata and file content. In general, metadata is information that describes content—for example, file properties and extended attributes—but you can index application-specific metadata as well. One of the main benefits of XML is that its structure contains meaningful information. The search engine allows users to perform advanced searches based on document structure.

In general, the search engine will return a list of file objects (that is, documents and components) that meet your search criteria. Because a file path might not provide enough information to help you identify content, you can also receive other indexed information from fields within documents and components. For example, you could configure the search engine to return the value of a display-name field with each result.

For details on performing searches, see the *TeamXML User's Guide* and the documentation provided with third party integrated XML editors. For details about how TeamXML executes searches, see “Search Statements” on page 103, “`iwxmlsearch.ipl`” on page 125, and “Search DTD” on page 132.

## Mutation

TeamXML introduces the concept of mutation:

- Unmodified document or component files that contain modified components are *mutated*.
- The file system's `Modified` time is equivalent to the mutation time.

The concept of mutation is important because it indicates that a containing document or component has been changed indirectly, through a change to one of its components.

# Installation and Initial Configuration

---

TeamXML consists of several software components. These components can reside in different locations on different systems depending on your site's size, configuration, and performance requirements. This chapter covers the following topics:

- An overview of TeamXML software components.
- Hardware and software required to support TeamXML.
- How you can determine the appropriate location for each TeamXML software component.
- Pre-installation tasks.
- Detailed installation steps.
- Post-installation tasks.
- Initial directory, server, and indexing configuration.
- How to uninstall TeamXML.

**Note:** Category definition files define the rules that control much of the behavior of TeamXML. Therefore, you should read Chapter 3, “Category Definition Files,” in conjunction with this chapter.

## TeamXML Software Components

Install the following TeamXML software components to ensure that all TeamXML features are available:

Software Component	Available From	Install On	Notes
TeamXML system files	TeamSite CD	TeamSite server system	Installed automatically when you install TeamSite 5.5.1 or later.
XML search engine	TeamXML CD	TeamSite server system or a separate system	See “TeamXML System Files” on page 35 for guidelines about where to install these files.
Tibco XML Authority	TeamXML CD	Any system used to create or edit TeamXML category definition files	See “Install XML Authority” on page 52 for more information.
Third-party XML editor and adapter software	Third parties	All TeamXML client systems	

## Hardware and Software Requirements

This section describes the server and client hardware and software required to run TeamXML. In many cases, the requirements are the same as those for TeamSite. Differences or similarities with the TeamSite requirements are noted throughout this section. Topics included in this section are:

- Server Hardware
- Server Software
- Client Hardware
- Client Software

## Server Hardware

TeamXML system files reside on the TeamSite server system and do not require additional hardware. Therefore, the hardware requirements for the TeamSite server apply even after the TeamXML system files and the XML search engine are installed and enabled on the TeamSite server system. These requirements are included for reference later in this section.

If you install the XML search engine on a separate system (rather than on the TeamSite server system), the hardware requirements will depend on the size of your site's XML repository, indexing parameters, performance needs, and numerous other factors. In general, the requirements for a separate search engine system are lower than the TeamSite server hardware requirements listed in this section. It is recommended that you contact an Interwoven representative for specific hardware recommendations should you choose to install the XML search engine on a separate system.

The following sections describe the hardware required for the system containing the TeamSite server, TeamXML program files, and XML search engine.

### Disk Space

Your system must have at least 760 megabytes (MB) (for UNIX systems) or 500 MB (for Windows systems) of disk space for the TeamSite, TeamXML, and XML search engine files, plus an additional five to 10 times the total amount of disk space you expect the TeamXML documents and components to consume.

### Number of CPUs

The following recommendations are based on the concurrent numbers of different types of users who will be using TeamSite and TeamXML. This is because some types of users tend to do CPU-intensive operations such as Get Latest, Submit, or Compare, while other users tend to do less CPU-intensive operations such as editing files or browsing directories.

All CPUs should be at least 400 MHz (for UNIX systems) or 700 MHz (for Windows systems).

**Note:** Solaris systems must be UltraSparc platforms.

To determine the number of CPUs your system needs:

1. Determine how many users will be using TeamSite and TeamXML intensively (such as members of a Web development team), moderately, or mildly (such as occasional contributors to the company intranet).
2. Next, use these numbers to determine how many of each type of user will be using TeamSite concurrently, TeamXML concurrently, or both. That is, those users who are actively interacting with TeamSite or TeamXML through one or more interfaces at the same time.
3. Locate the number of concurrent users in the chart below to determine the number of CPUs your system needs for each type of user, and add them together to get your total number of CPUs. You must have a minimum of two CPUs.

		Concurrent Users		
		Intense	Moderate	Mild
<b>CPUs</b>	<b>1</b>	25	50	100
	<b>2</b>	50	100	200
	<b>4</b>	100	200	400
	<b>8<sup>1</sup></b>	200	400	800

1. If the number of expected concurrent users is greater than the numbers in this row, please contact an Interwoven representative.

For example:

If you have 60 intense, 250 moderate, and 2000 mild users, and you expect 40 percent of the instance users, 20 percent of the moderate users, and 10 percent of the mild users will be using TeamSite or TeamXML concurrently, then your concurrent users and total number of required CPUs would be as follows:

$$\begin{array}{rclcl}
 60 \text{ intense users} * .4 & = & 24 \text{ concurrent intense users} & = & 1 \text{ CPU} \\
 250 \text{ moderate users} * .2 & = & 50 \text{ concurrent moderate users} & = & 1 \text{ CPU} \\
 2000 \text{ mild users} * .1 & = & 200 \text{ concurrent mild users} & = & 2 \text{ CPUs} \\
 & & \text{Total} & & \underline{4 \text{ CPUs}}
 \end{array}$$

## Memory

A good rule of thumb for calculating memory requirements is 1 gigabyte (GB) of memory for each CPU (see above). To calculate memory requirements more precisely, use the formula below.

(1 GB for TeamSite/TeamXML and associated programs) + (cache size setting \* 4 KB) + (4.6 MB \* total number of concurrent users)

where the cache size setting is specified using the `cachesize` parameter in `iw.cfg`. By default, this setting is 30,000. For information on changing this setting, see the *TeamSite Administration Guide*.

For example, the memory requirements for the preceding example system (which has 24 + 50 + 200 concurrent users), with the default cache size setting, would be:

$$1 \text{ GB} + (30000 * 4 \text{ KB}) + (4.6 \text{ MB} * (24 + 50 + 200)) = 2.38 \text{ GB}$$

If you encounter a significant amount of memory swapping, you should either reduce the `cachesize` setting in `iw.cfg` or install more memory.

## Inode requirements (UNIX systems only)

Use the following formula to estimate how many inodes your server will require:

$$\# \text{ inodes} = (\# \text{ branches})(\# \text{ average files in staging area per branch})(\# \text{ average historical versions/file}) \\ (1 + (\% \text{ of files having extended attributes}) / 100)(\text{safety-factor}) / 100$$

For example, if your TeamSite/TeamXML server has three branches, with 20,000 files in the staging area of each branch, (on average), 10 versions of each file in its history list (on average), seven percent of files have extended attributes, and you want to use a 1.5x safety factor:

$$\begin{aligned} \# \text{ inodes} &= 3 * 20,000 * 10 * (1 + .07) * 1.5 / 100 \\ &= 9630 \text{ inodes} \end{aligned}$$

## Disk Configuration

For maximum disk space efficiency, the TeamSite backing store should be installed on drives formatted with a 512 byte cluster (on Windows systems) or 512 byte block or fragment size (on UNIX systems). For ease of maintenance, you might want to install the backing store on its own partition.

It is recommended that you use RAID 0+1 to configure your environment. RAID 5 can also be used for environments with a relatively low number of writes as a percentage of total accesses. Because TeamSite environments generally have a large percentage of writes, RAID 0+1 should provide better overall performance.

In addition to using RAID configurations, it is recommended that you use the fastest available SCSI controllers (160 MB/second transfer rate) and SCSI drives (10,000 RPM).

**Note:** Software RAID solutions are not recommended because they are very CPU-intensive.

## Server Software

Server software requirements differ depending on whether you install the XML search engine on the same system as the TeamSite server or on a separate system.

### Search Engine and TeamSite Server on Same System

If you install the XML search engine on the TeamSite server system, the system must run the following software:

- One of the following operating systems:
  - Windows 2000, Service Pack 1.
  - Windows NT 4, Service Pack 5 or later.
  - Solaris 2.6, 7, or 8.
- TeamSite 5.5.2, plus all released TeamSite service packs.

If you will use this system to create or edit category definition files, you should also install Tibco XML Authority (bundled on the TeamXML CD).



### Search Engine and TeamSite Server on Separate Systems

If you install the XML search engine on a separate system from the TeamSite server, the systems must run the following software:

TeamSite Server System	XML Search Engine System
<ul style="list-style-type: none"> <li>• One of the operating systems listed in the preceding section.</li> <li>• TeamSite 5.5.2, plus all released TeamSite service packs.</li> <li>• Tibco XML Authority (bundled on the TeamXML CD; install on systems used to create or edit category definition files).</li> </ul>	<ul style="list-style-type: none"> <li>• One of the operating systems listed in the preceding section. It does not need to match the operating system on the TeamSite server system.</li> </ul>

See “Software Component Locations” on page 35 for more information about installing the XML search engine on the TeamSite server system or a separate system.

### Client Hardware

TeamXML end users must be able to access both TeamSite and TeamXML. TeamSite access is provided through browser-based thin-client technology. The only client-side hardware requirements for TeamSite access are that the RAM, CPU, local storage, and networking capability must be sufficient to operate a Web browser and the editing applications of the user’s choice.

TeamXML access is provided through third-party integrated XML editors (such as SoftQuad XMetaL or Arbortext Epic Editor). Contact an Interwoven representative for details about which integrated XML editors support TeamXML. Refer to the documentation provided with those products for details about possible additional client-side hardware requirements.

## Client Software

The following software must run on each client system to provide TeamXML access:

- One of the following operating systems:
  - Windows 95, 98, NT, or 2000.
  - Solaris 2.6, 7, or 8.
  - MacOS 8.6 through 9.x.
- Tibco XML Authority (bundled on the TeamXML CD; install if the client system will be used to create or edit category definition files).
- A third-party integrated XML editor.
- Adapter software to integrate the integrated XML editor with TeamXML (available from third parties).
- One of the following browsers:

	<b>Netscape</b>	<b>Internet Explorer</b>
<b>Windows 95, 98, and NT</b>	4.7x	4.x-5.5 <sup>1</sup>
<b>Windows 2000</b>	4.7x	5.0-5.5 <sup>1</sup>
<b>Solaris (Sparc) 2.6, 7, and 8</b>	4.7x	Not supported
<b>MacOS 8.6–9.x</b>	4.7x <sup>2</sup>	5.0

1. Some versions of Internet Explorer 5.5 do not include the Java Virtual Machine. If you do not have the Java Virtual Machine you can download it from the Microsoft Web site at [www.microsoft.com](http://www.microsoft.com).
2. Interwoven Merge not supported on Netscape for MacOS.

**Notes:** If you are using Netscape browsers to display multibyte characters, you must select **Edit > Preferences > Appearance > Fonts** and set the **Use my fonts, overriding page-specified fonts** option.

See the *TeamXML Release Notes* for information about possible client support limitations.

## Software Component Locations

This section discusses the things you should consider when determining installation locations for the TeamXML system files, XML search engine, Tibco XML Authority, and third-party integrated XML editors. It also lists default installation locations for TeamXML files.

### TeamXML System Files

Most TeamXML system files reside on the TeamSite server system and are installed automatically with TeamSite.

Additional TeamXML system files, mostly related to the XML search engine and product activation, are distributed on the TeamXML CD and are installed during TeamXML installation. The following sections describe these files' locations in detail.

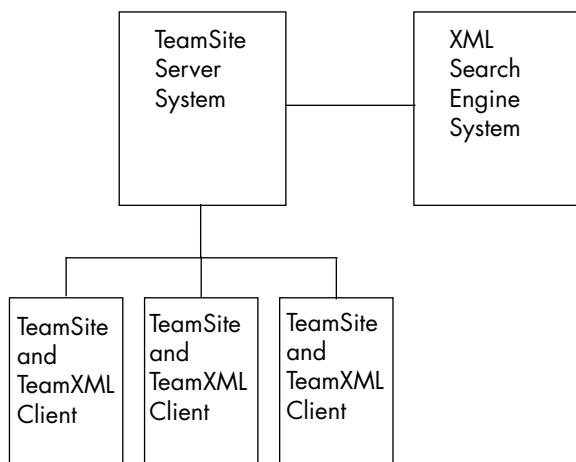
#### XML Search Engine

Two general types of files are associated with the XML search engine: program files that execute to run the search engine, and a set of index files that are generated when the search engine runs. The program and index files must reside on the same system. You can choose to install these files on the TeamSite server system or on their own system separate from the TeamSite server.

It is generally advisable to install the XML search engine on a separate system from the TeamSite server. The XML search engine can generate very large index files. The process of searching through and updating these index files can have a significant impact on the performance of the system on which they reside. It is therefore highly recommended that you install the XML search engine on a system that can tolerate periodic slowdowns.

## Network Connections

If you install the XML search engine on a separate system, you must ensure that there is a network connection between the TeamSite server system and the search engine system, and between the TeamSite server system and all client systems:



## Default File Locations

The following table shows the default locations of TeamXML program, index, and sample files on UNIX and Windows systems. See “Create Directory Structure and Sample Categories” on page 55 for more information about installation locations.

	UNIX Systems	Windows Systems
<b>XML Search Engine Program Files</b>	/usr/local/teamxml-home	C:\Program Files\Interwoven\TeamXML\search
<b>XML Search Engine Index Files</b>	parent_dir/teamxml-home/xml-search/index	C:\iw-index
<b>Sample XML Files</b>	iw-home/iwopenapi/example/teamxml	C:\Program Files\Interwoven\TeamSite\iwopenapi\example\teamxml
<b>TeamXML DTDs</b>	iw-home/iwopenapi/dtd	C:\Program Files\Interwoven\TeamSite\iwopenapi\dtd

You can move the program and index files to locations of your choice (for example, to optimize disk space or performance).

## **Tibco XML Authority**

The Tibco XML Authority schema editor is included on the TeamXML CD. You should manually install XML Authority on client systems that will be used to create or edit category definition files as described in “Install XML Authority” on page 52.

## **Integrated XML Editors**

You must install an integrated XML editor (such as SoftQuad XMetaL or Arbortext Epic Editor) and its Interwoven-specific adapter software on all client systems. The editors, adapter software, and installation instructions are available directly from their manufacturers. Contact an Interwoven representative for details about which integrated XML editors support TeamXML. See the documentation provided with the integrated XML editor software for information about installation locations.

# **Pre-Installation Tasks**

Perform the following tasks prior to installing TeamXML software components:

1. Ensure that the latest version of TeamSite 5.5.2 (including service packs) is running on the TeamSite server system. See the *TeamSite Administration Guide* for details about installing TeamSite.
2. If you are installing the XML search engine on a separate system from the TeamSite server system:
  - Note the XML search engine system’s domain, host, and port. You will be prompted for that information during installation.
  - Ensure that the XML search engine system meets the hardware, software, and networking requirements described earlier in this chapter or provided by an Interwoven representative.

3. If you are installing the XML search engine on the TeamSite server system, note the location (including the partition) of the TeamSite backing store. The XML search engine's index files and the TeamSite backing store *should not* reside on the same partition because of performance considerations. During installation you will be prompted to specify a location for the XML search engine's index files. You should not specify the TeamSite backing store location at this prompt.

## Installation Steps

This section describes how to install the TeamXML software components. Unless otherwise noted, steps apply to both UNIX and Windows installations.

### Installation Task Overview

This section contains a high-level list of the activities to perform during a TeamXML installation. See “Installation Task Details” on page 39 for more detailed instructions.

The installation tasks that you perform, the locations from which you perform them, and the order in which you perform them depend on where you install the XML search engine.

#### Search Engine and TeamSite Server on Same System

Perform the following steps if you install the XML search engine on the TeamSite server system:

- Initiate the TeamXML installation program on the TeamSite server system.
- Specify a port for the XML search engine.
- Specify a disk location for the XML search engine's index files.
- Install the TeamXML sample content (optional).
- Exit from the TeamXML installation program.
- Reboot the system (Windows systems only).
- Verify that the TeamXML system files were installed and configured correctly (optional).
- Verify that the XML search engine is running (optional).
- Install Tibco XML Authority.
- Install a third-party integrated XML editor on client systems.
- Run the `iwreset` command-line tool.

## Search Engine and TeamSite Server on Separate Systems

You will perform the following steps if you install the XML search engine on a separate system from the TeamSite server:

- Initiate the TeamXML installation program on the TeamSite server system.
- Specify the domain, host, and port for the XML search engine system.
- Install the TeamXML sample content (optional).
- Exit from the TeamXML installation program.
- Initiate the TeamXML installation program on the XML search engine system.
- Specify a port for the XML search engine.
- Specify a disk location for the XML search engine's index files.
- Exit from the TeamXML installation program.
- Reboot the XML search engine system (Windows systems only).
- Verify that the TeamXML system files were installed and configured correctly (optional).
- Verify that the XML search engine is running (optional).
- Install Tibco XML Authority.
- Install a third-party integrated XML editor on client systems.
- Run the `iwreset` command-line tool on the TeamSite server system.

## Installation Task Details

The following sections contain step-by-step instructions for installing TeamXML on Windows and UNIX systems.

### Installing on a Windows System

You must have **Administrator** privileges to install TeamXML on a Windows system. Perform the following steps on the TeamSite server system to install TeamXML:

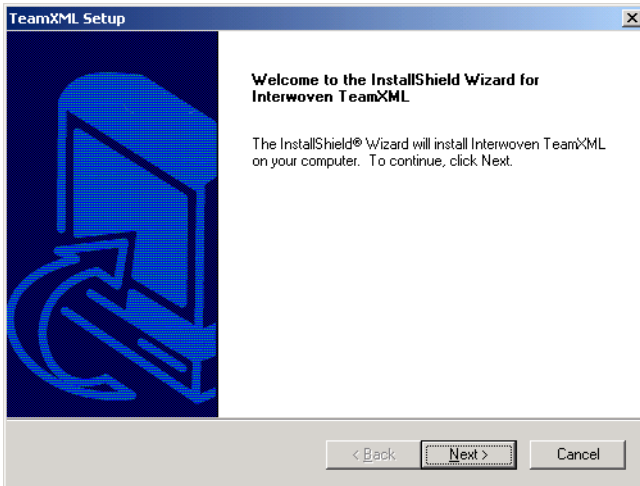
1. Insert the TeamXML CD in the TeamSite server system and navigate to the TeamXML `install` folder on the CD.

**Note:** If you are installing the XML search engine on a separate system from the TeamSite server, you will perform additional installation tasks on that system later in the installation procedure.

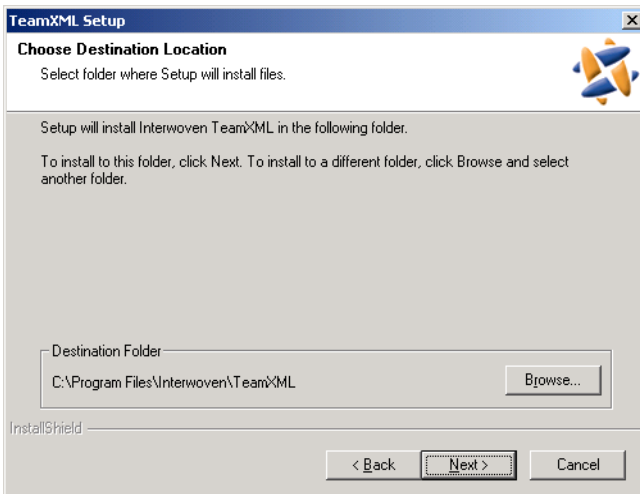
2. Double-click the installation file:

TeamXML551Build6215.exe

Click **Next** in the screen that appears:

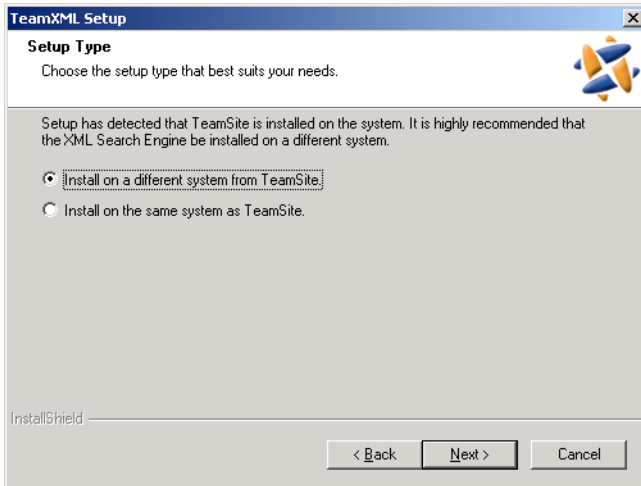


3. Select a location for the TeamXML program files.

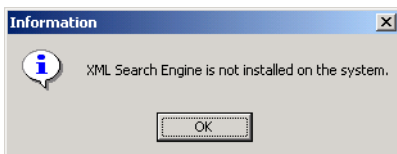




4. The installation program verifies that you are logged into the TeamSite server system by checking for the presence of the TeamSite program files. You are prompted to specify whether to install the XML search engine on the TeamSite server system or on a separate system.

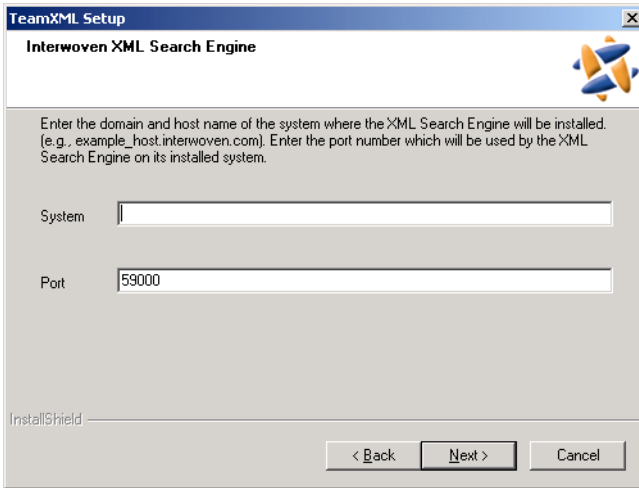


- a. To install the XML search engine on the TeamSite server system, click the bottom radio button, click **Next**, and go to Step 5 to continue the installation.
- b. To install the XML search engine on a separate system, click the top radio button as shown above and click **Next**. You see the following dialog box:



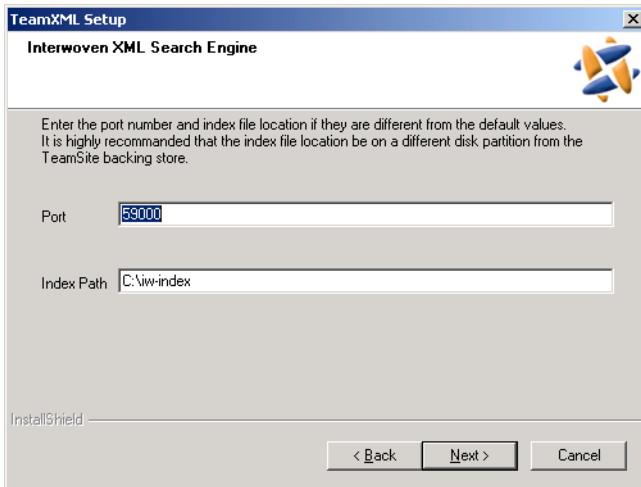
Click **OK** and proceed to Step 5.

5. You are prompted for system information.
  - a. If you are installing the XML search engine on a separate system, you are prompted for the domain and host name of the search engine system, and the XML search engine port on that system as shown in the following screen. Enter the information, click **Next**, and proceed to Step 6.

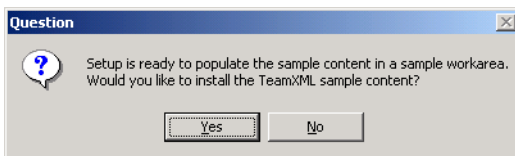


The image shows a Windows-style dialog box titled "TeamXML Setup" with a subtitle "Interwoven XML Search Engine". It contains instructions: "Enter the domain and host name of the system where the XML Search Engine will be installed. (e.g., example\_host.interwoven.com). Enter the port number which will be used by the XML Search Engine on its installed system." There are two input fields: "System" (empty) and "Port" (containing "59000"). At the bottom, there are three buttons: "< Back", "Next >", and "Cancel". The "InstallShield" logo is visible in the bottom left corner.

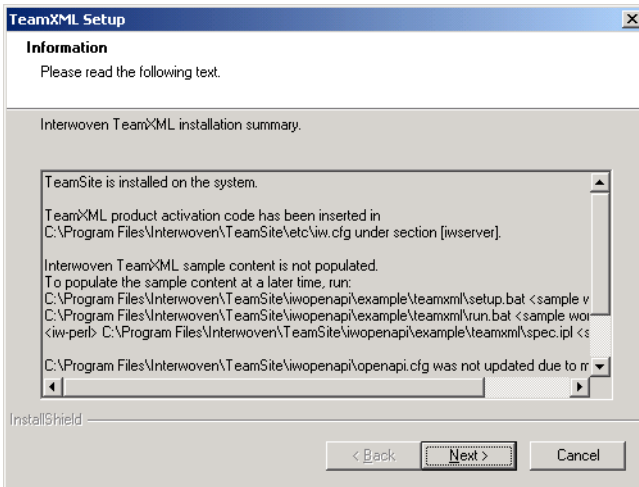
- b. If you are installing the XML search engine on the TeamSite server system, you are prompted for the XML search engine port and a path for the index files that the XML search engine generates as shown in the following screen. Enter that information and click **Next**.



6. You see the following prompt giving you the option of installing sample XML content in a sample workarea. The sample content consists of small directory structure containing several category definition files, documents, and components that you can view, edit, and reuse as you wish. It is recommended that you install the sample content at this time. If the sample content cannot be found or installed (if, for example, the TeamSite server is not currently running on this system), you see one or more error messages describing the problem. If the sample content cannot be installed now, or if you elect not to install it now, you can install it later by running a sample content installation program manually as described in “Install Sample Content Manually” on page 59.

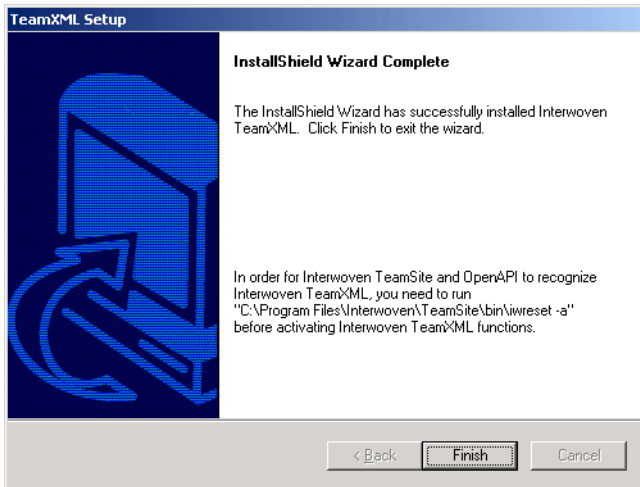


7. System information similar to the following is displayed:



Click **Next**.

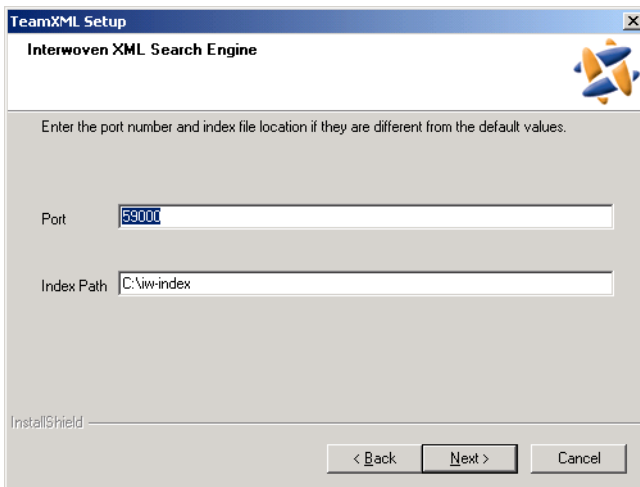
8. The TeamXML installation program displays a screen similar to the following when the installation tasks specific to the TeamSite server system are complete.
  - a. If you are installing the XML search engine on the TeamSite server system, you are also prompted to reboot the system. If you see the reboot prompt, it is recommended that you select the reboot option and click **Finish** now. After you perform this step, the TeamXML installation is complete. Proceed to “Post-Installation Tasks” on page 52.
  - b. If you are installing the XML search engine on a separate system, you do not see the reboot prompt. Instead, you see instructions for running the `iwreset` command-line tool. You will run `iwreset` later as described in detail in “Run iwreset” on page 52. For now, click **Finish** and proceed to Step 9.



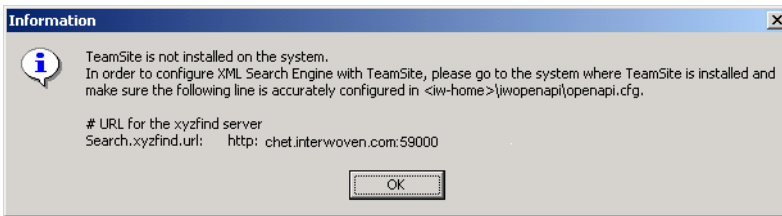
9. **Note:** The remaining steps in this procedure apply only if you chose to install the XML search engine on a separate system in Step 4.

Repeat Steps 1 through 3 on the XML search engine system.

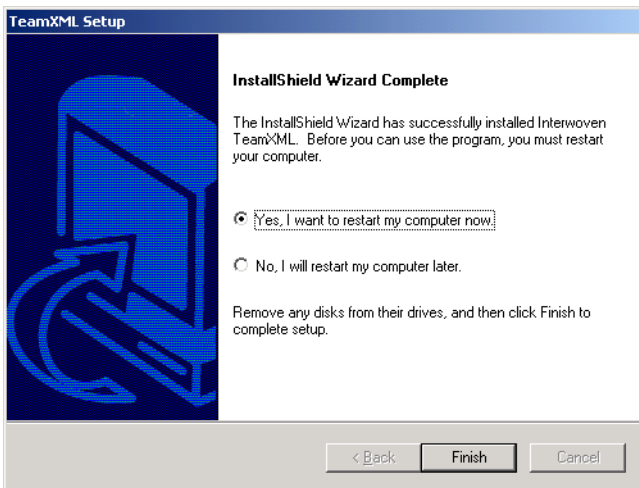
10. You are prompted for the XML search engine port and a path for the index files that the XML search engine generates. Enter that information and click **Next**.



11. You are prompted to verify that the OpenAPI configuration file on the TeamSite server system is properly configured for the XML search engine. Under normal circumstances this configuration is performed automatically by the installation program. The prompt that you see now is for informational purposes only. Clicking **OK** does not automatically verify that configuration was performed. To verify proper configuration, open the `openapi.cfg` file on the TeamSite server system and check for the presence of the string shown in the last line of text in the dialog box on your screen.



12. The TeamXML installation program displays the following screen when installation tasks specific to the XML search engine system are complete. It is recommended that you reboot your system at this time.



13. The TeamXML installation is complete. Proceed to “Post-Installation Tasks” on page 52.

## Installing on a UNIX System

You must have **root** privileges to install TeamXML on a UNIX system. Perform the following steps on the TeamSite server system to install TeamXML:

1. Insert the TeamXML CD in the TeamSite server system. Copy the TeamXML installation file to a directory of your choice. This directory does not need to be the permanent home directory for TeamXML; you will specify that directory in the next step.

**Note:** If you are installing the XML search engine on a separate system from the TeamSite server, you will perform additional installation tasks on that system later in the installation procedure.

2. Unzip and untar the TeamXML installation file and place it in the TeamXML parent directory by entering the following command. The parent directory is a directory of your choice; typical locations are `/` or `/usr`.

```
gunzip -c IWOVteamxml-sol.5.5.2.Build7xxx.tar.gz | (cd  
/parent_directory; tar xvpf -)
```

This command creates a `teamxml-home` directory in *parent\_directory*. The `teamxml-home` directory contains subdirectories and the TeamXML program files.

3. Go to the `install` directory in *parent\_directory/teamxml-home* and execute the `teamxml_install` script:

```
cd parent_directory/teamxml-home/install  
./teamxml_install
```

4. The installation program verifies that you are logged into the TeamSite server system by checking for the presence of the TeamSite program files. You are prompted to specify whether to install the XML search engine on the TeamSite server system or a separate system:

Setup has detected that TeamSite is installed on the system. It is highly recommended that the XML Search Engine be installed on a different system.

Enter 1 or 2 for your setup choice:

- 1 - Install on a different system from TeamSite
- 2 - Install on the same system as TeamSite



- a. To install the XML search engine on the TeamSite server system, enter **2** and go to Step 5 to continue the installation.
- b. To install the XML search engine on a separate system, enter **1**. You see the following information and prompt:

The XML Search Engine will not be installed on the current system.

Enter the domain name and host name of the system where the XML Search Engine will be installed. This information will be added to the OpenAPI configuration file located on the TeamSite server.

Enter domain name and host name (e.g., example\_host.interwoven.com):

At the preceding prompt, you can enter the domain and host name of the search engine system, or you can leave the field blank and press **Return**. If you entered a domain and host name for the search engine system, proceed to Step 5.

If you left the field blank, you see the following messages:

WARNING: You did not enter the information needed for configuring the XML Search Engine with TeamSite. You can manually perform this task at a later time.

Please make sure the following line is accurately configured in iw-home/iwopenapi/openapi.cfg:

```
# URL for the xyzfind server
Search.xyzfind.url:      http://test.interwoven.com:59000
```

The prompt that you see here is for informational purposes only. To ensure proper configuration, you will need to enter the domain and host information manually in iw-home/iwopenapi/openapi.cfg. Upon completion of the TeamXML installation, open the openapi.cfg file on the TeamSite server system and add the last two lines of text similar to that shown in the preceding prompt.



5. You are prompted to specify the XML search engine port.

- a. If you are installing the XML search engine on a separate system, you see the following prompt. Enter the port number and proceed to Step 7.

```
Enter the XML Search Engine port number on the system where
it will be installed. This information will be added to OpenAPI
configuration file located on the TeamSite server.
Enter the port number if it is different from the
default value [59000]
```

- b. If you are installing the XML search engine on the TeamSite server system, you see the following prompt. Enter the port number and proceed to Step 6.

```
Enter the XML Search Engine port number if it is different from the
default value [59000]
```

6. If you are installing the XML search engine on the TeamSite server system, you are prompted for a path for the index files that the XML search engine generates:

```
Enter XML Search Engine index file location if it is different
from the default value. [parent_dir/teamxml-home/xml-search/index]
```

7. Additional messages are displayed. Message contents depend on where you are installing the XML search engine.

- a. If you are installing the XML search engine on a separate system, you are notified that you will need to perform additional installation steps on the search engine system:

```
TeamSite server has been configured to work with the XML Search
Engine. Please continue the installation process of the XML Search
Engine on test.interwoven.com.
```

However, before you can perform those steps on the search engine system, you must complete Step 8 while you are still logged into the TeamSite server system. Proceed to Step 8 and continue from there.

- b. If you are installing the XML search engine on the TeamSite server system, all TeamXML program files are now installed. You see the following information:

```
Installation of Interwoven TeamXML is completed.
```



8. The installation script gives you the option of installing sample XML content in a sample workarea. The sample content consists of a small directory structure containing several category definition files, documents, and components that you can edit and reuse as you wish. It is recommended that you install the sample content at this time. If you elect not to, you can install it later by running a sample content installation script manually. You see the following prompts. The log file path names might differ from what is shown here:

```
Would you like to install sample content in a sample workarea? [y]
```

```
Enter the name of the sample workarea if it is different  
from the default value. [teamsiteworkarea]
```

```
See iw-home/iwopenapi/example/teamxml/setup.log and  
iw-home/iwopenapi/example/teamxml/run.log and  
iw-home/iwopenapi/example/teamxml/spec.log for status  
on the sample workarea.
```

If the sample content cannot be found or installed (if, for example, the TeamSite server is not currently running on this system), you see one or more error messages describing the problem.

9. All TeamXML installation activities specific to the TeamSite server are now complete. When the installation script finishes, execute the `iwreset` command on the TeamSite server system as described in the following prompt:

```
In order for Interwoven TeamSite and OpenAPI to recognize Interwoven  
TeamXML, you need to run <iw-home>/bin/iwreset -a on the TeamSite  
server before activating TeamXML functions.
```

```
The output of the installation process is logged in  
parent_dir/teamxml-home/install/teamxml_install.log.
```

10. Your next step depends on where you chose to install the XML search engine.

- a. If you chose to install the XML search engine on the TeamSite server system in Step 4, the TeamXML installation is now complete. Proceed to “Post-Installation Tasks” on page 52.
- b. If you chose to install the XML search engine on a separate system in Step 4, proceed to Step 11.

11. Repeat Steps 1 through 3 on the XML search engine system.

12. You are prompted for the XML search engine port and a path for the index files that the XML search engine generates:

Enter the XML Search Engine port number if it is different from the default value [59000]

Enter the XML Search Engine index file location if it is different from the default value. [teamxml-home/xml-search/index]

13. You are prompted to verify that the iw-home/iwopenapi/openapi.cfg configuration file on the TeamSite server system is properly configured for the XML search engine. The prompt that you see now is for informational purposes only. To verify proper configuration, open the openapi.cfg file on the TeamSite server system upon completion of the TeamXML installation and check for the presence of the string shown on the last line of text in the following prompt.

The XML Search Engine has been installed on the current system. In order for the XML Search Engine to work with TeamSite server, please make sure the following line is configured correctly in iw-home/iwopenapi/openapi.cfg on the TeamSite server:

Search.xyzfind.url:       http://test.interwoven.com:59000

The TeamXML installation is complete. Proceed to “Post-Installation Tasks” on page 52.

## Post-Installation Tasks

After performing the installation tasks described in the preceding sections, you should perform several additional tasks:

- Run the `iwreset` command-line tool (mandatory).
- Install Tibco XML Authority (on systems used to create or edit category definition files).
- Install an integrated XML editor on client systems (mandatory).
- Set up indexing (mandatory).
- Create a TeamXML directory structure and sample categories (mandatory).
- Verify that the TeamXML system files were installed and configured correctly (optional).
- Verify that the XML search engine is running (optional).
- Install sample content manually (optional).
- Configure the TeamSite server (optional).

### Run `iwreset`

Following installation, you must run the `iwreset -a` command so that the TeamSite server rereads configuration files and restarts services. See *TeamSite Command-Line Tools* for more information.

### Install XML Authority

The Tibco XML Authority schema editor is the recommended tool for creating and editing TeamXML category definition files. You should install XML Authority on systems that administrators will use to create or edit TeamXML category definition files.

#### Windows systems:

1. Insert the TeamXML CD and navigate to the `/Extras` directory.
2. Double-click the XML Authority installation file, `TIB_xmlauth_2.2.2_w32_ts.exe`. Follow the directions provided by the XML Authority installation program. The default installation location for XML Authority is `C:\Program Files\TIBCOExtensibility`.

**UNIX systems:**

1. Insert the TeamXML CD and navigate to the /Extras directory.
2. Copy the XML Authority installation file, `TIB_xmlauth_2.2.2_s4_56_ts.tar.Z`, to a directory of your choice.
3. Unzip and untar the XML Authority installation file and place it in the TeamXML parent directory by entering the following command. The parent directory is a directory of your choice; typical locations are / or /usr.

```
gunzip -c TIB_xmlauth_2.2.2_s4_56_ts.tar.Z | (cd /parent_directory;  
tar xvpf -)
```

This command creates a TIBCOExtensibility directory in *parent\_directory*. The TIBCOExtensibility directory contains subdirectories and the XML Authority program files.

4. Follow the installation instructions in TIBCOExtensibility/INSTALL. When executing the instructions from the “General Instructions” section, begin with Step 4, “Change directory to the 'TIBCOExtensibility/XA' directory...”

Online help provided with XML Authority is also installed in the TIBCOExtensibility directory at this time.

**Note:** There is a limit to the number of XML Authority licenses provided on the TeamXML CD. See the *TeamXML Release Notes* for more information.

**Install an Integrated XML Editor**

You must install an integrated XML editor on each client system that will be used with TeamXML. Contact an Interwoven representative for details about which integrated XML editors support TeamXML. Installation involves two software components: the generally available XML editor application, and the Interwoven adapter that allows the editor to be used with TeamXML. Both software components are available from the integrated editor manufacturers.

## Set Up Indexing

Before you can use any TeamXML search features, you must apply a set of indexing rules to TeamXML components, documents, or simple (that is, non-TeamXML) files. How you apply indexing rules depends on whether the rules will apply to TeamXML or non-TeamXML files. In either case, the indexing rules must conform to the DTD shown in “DTD for Indexing Rule Set” on page 130.

### Indexing TeamXML Files

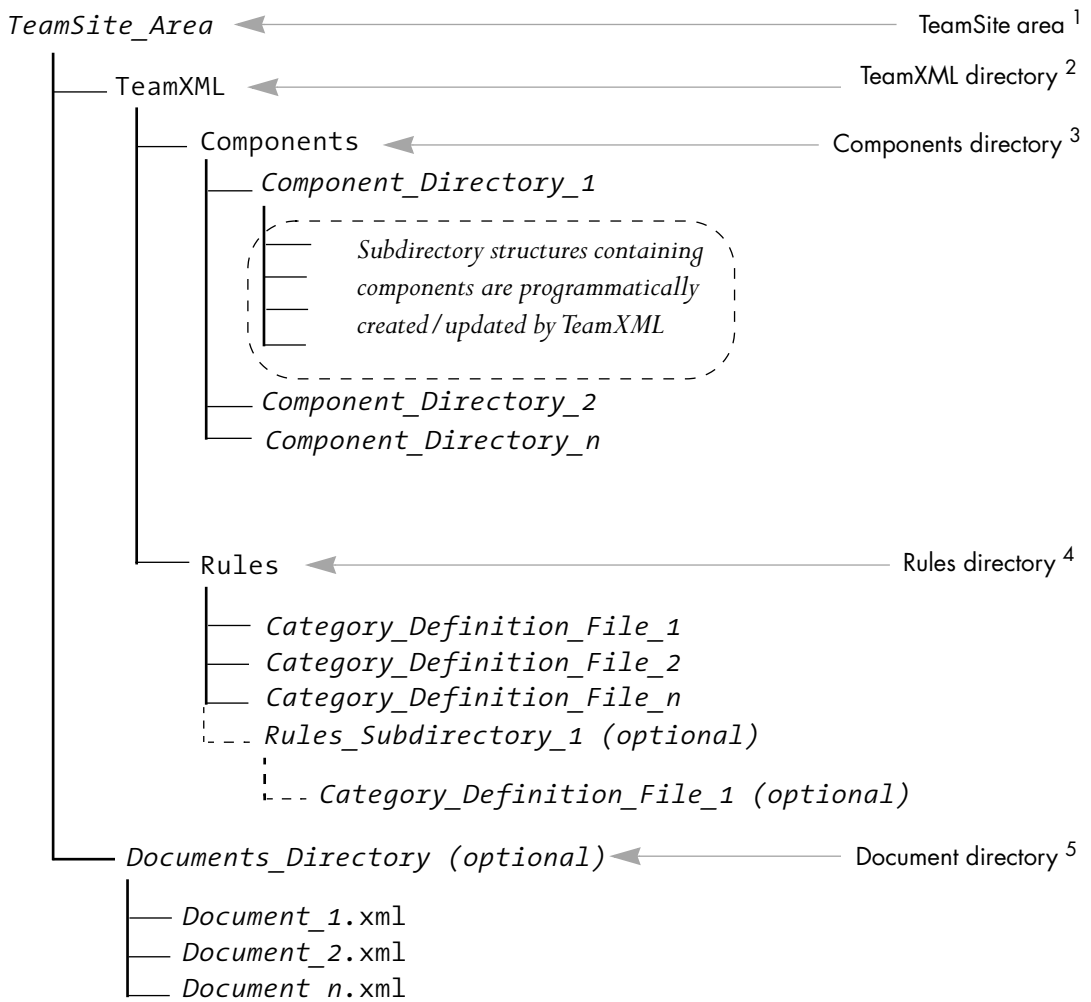
Indexing rules that apply to TeamXML files (that is, files that have been componentized with `iwxmfile`) must reside in the category definition file that you use to componentize the XML file. See “Indexing Rule Set” on page 76 for more information.

### Indexing Non-TeamXML Files

Indexing rules that apply to non-TeamXML files (that is, simple XML files that have not been componentized with `iwxmfile`) must reside in standalone indexing rule files. You must apply these indexing rules to a specified file by running the `iwxmlindex.ipl` command-line tool. See “`iwxmlindex.ipl`” on page 118 for more information.

## Create Directory Structure and Sample Categories

After installing TeamXML, you should create a directory structure for TeamXML files that matches the following layout as closely as possible. This is necessary because category definition files, component directories, and component files cannot be moved or renamed after they are created. Only document files can be moved or renamed after they are initially installed on your system. Main items shown here in the directory structure are keyed to a comment section that explains more details about each item.



## Directory Structure Details

The items in the preceding figure are as follows:

**1: TeamSite Area:** A TeamSite area (typically a user's workarea) that serves as the area root for all TeamXML components, rules, and documents to which the user has access.

**2: TeamXML directory:** A TeamXML directory that holds all components and rules. It is recommended that you name this directory TeamXML as shown in the figure. You must place this directory immediately under the area root because the path to the area root cannot be changed after it is created.

**3: Components directory:** A Components directory that holds subdirectories for TeamXML components and the components themselves. It is recommended that you name this directory Components as shown in the figure. The first-level directories under Components, such as *Component\_Directory\_1*, may have names of your choice, even though these directories are created automatically by TeamXML (their names are derived from entries that you place in category definition files). These are the directories that you specify in category definition files to control where TeamXML places components when a file is componentized. (See Chapter 3, "Category Definition Files," for details about constructing category definition files.) The subdirectory structures under *Component\_Directory\_1* (and under any other component directories, should they exist) are programmatically created and updated by TeamXML when you componentize a document, or when you create or modify a component by some other means.

**4: Rules directory:** A Rules directory that holds all category definition files. It can optionally hold subdirectories if such a layout is necessary to help organize your site's data. It is recommended that you name this directory Rules as shown in the figure. The optional subdirectories (such as *Rules\_Subdirectory\_1*) and category definition files (such as *Category\_Definition\_File\_1*) may have names of your choice.

**5: Document directory:** A directory that holds TeamXML documents. This directory may have a name of your choice, and is shown as *Documents\_Directory* in the figure. The name and location of this directory are less critical than with other parts of the directory structure because document files can be moved, renamed, and deleted. However, in most situations it is highly recommended that you do not place the documents directory within the TeamXML directory; instead, place it as close as possible to the top of the directory structure to maximize performance when files are renamed.



## Populating the Directory Structure

This section describes how to populate the directory structure shown in “Create Directory Structure and Sample Categories” on page 55.

### Rules Directory

You populate the Rules directory structure with category definition files. You should create the files originally in the Rules directory and then specialize them so that TeamXML is aware of them. You can use any editing tool to create category definition files; however, it is highly recommended that you use the XML Authority schema editor that is bundled with TeamXML. XML Authority has integration features specific to TeamXML, giving it the ability to export newly created category definition files directly into TeamSite areas, enforce the category definition file DTD, and automatically specialize category definition files when you create or edit them. If you use any other tool to create a category definition file, you must specialize the file manually with the `iwxmllcat` command-line tool. For example, if you just created a new category definition file `doctype1.dfn` in the directory `/default/main/WORKAREA/andre/TeamXML/Rules`, you would execute the following command to specialize it:

```
iwxmllcat -c /default/main/WORKAREA/andre/TeamXML/Rules/doctype1.dfn
```

See “iwxmllcat” on page 114 or more information about the `iwxmllcat` command.

### Document and Component Directories

Components and XML documents become part of the TeamXML repository (and are recognized by TeamXML) only after you have imported them into TeamXML—that is, after you have converted them by executing the `iwxmllfile -c TeamXML` command-line tool. This process is sometimes referred to as *specialization* elsewhere in Interwoven and third party documents. Simply adding a file to the TeamXML directory structure (for example, using the UNIX `cp` command, or by dragging and dropping a file on a Windows system) does not import the file into TeamXML.

When you run `iwxmllfile`, you specify a category definition file that will not only import the specified ordinary XML file into TeamXML, but will also componentize it. Therefore, you must already have set up your category definition files set up before you can run `iwxmllfile`. See “Guidelines for Creating Categories” on page 64 for more information.

## Importing Existing XML Content

If you already have a large repository of XML content, you should contact an Interwoven representative for details about how to automate the importation of that content into TeamXML.

## Verify Configuration Settings

The `openapi.cfg` file on the TeamSite server system must contain a reference to the XML search engine system and port. Under most circumstances, this reference is added automatically by the TeamXML installation program. You can verify that this reference exists by checking for the following lines in `openapi.cfg`. On UNIX systems, `openapi.cfg` resides in `iw-home/iwopenapi`. On Windows systems, it resides in `C:\Program Files\Interwoven\TeamSite\iwopenapi`.

```
# URL for the xyzfind server
Search.xyzfind.url:  httpd://host_name.domain_name:port_number
```

For example:

```
# URL for the xyzfind server
Search.xyzfind.url:  httpd://test.interwoven.com:59000
```

## Verify XML Search Engine Startup

The TeamXML installation program starts the XML search engine automatically. You can verify that the search engine is running as a daemon (on UNIX systems) or as a service (on Windows systems) by executing a simple TeamXML search. This is a valid test even if you have not yet indexed any documents or components. For example, when you execute the following command, it should return a list (either an empty list or a list containing some objects, depending on how your files are indexed):

```
iwxmlsearch.ipl keyword / dog
```

If the preceding command returns an error message, it is likely that the search engine is not running. Another way to verify that the XML search engine is running is as follows:

**UNIX systems:** Verify that an `xyzfind` process is running by executing the following command:

```
ps -ef | grep xyzfind
```

**Windows systems:** Verify that an Interwoven XML Search Engine service is shown in the services control panel.

## Install Sample Content Manually

If you chose not to install the TeamXML sample content in Step 6 (Windows systems) or Step 8 (UNIX systems), or if the installation program could not install the sample content, you can install it manually as follows. When you execute the steps described in this section, the following actions take place:

- A TeamXMLExample branch is created.
- An edition, staging area, and workarea named `teamxmlworkarea` are created in the TeamXMLExample branch.
- A directory named `ExampleDocuments` is created in `teamxmlworkarea`.
- Within `ExampleDocuments`:
  - The `mktgreq.xml` file is componentized.
  - Annotated copies of the components generated from `mktgreq.xml` are created.
  - The annotated component copies are inserted into `datasheet.xml` and `setupguide.xml`.
  - The `datasheet.xml` and `setupguide.xml` files are componentized.

**Windows systems:** Execute the following programs:

```
C:\Program Files\Interwoven\TeamSite\iwoopenapi\example\teamxml\setup.bat
C:\Program Files\Interwoven\TeamSite\iwoopenapi\example\teamxml\run.bat
C:\Program Files\Interwoven\TeamSite\iwoopenapi\example\teamxml\spec.ipl
```

**UNIX systems:** Execute the following programs:

```
iw-home/iwoopenapi/example/teamxml/setup.sh
iw-home/iwoopenapi/example/teamxml/run.sh
iw-home/iwoopenapi/example/teamxml/spec.ipl
```

## Configure the TeamSite Server

This section describes parameters that you can configure to optimize TeamSite server performance. Server performance is related to the total number of components in a file. To optimize server performance, you can limit the number of components that TeamXML will create. The parameters that specify these limits are in `iw-home/etc/iw.cfg`. You can change these values, although if you increase them you might encounter performance problems. It is recommended that you change these values only after consulting an Interwoven representative.

Parameter	Default Value	Description
xml_max_children	250	The maximum number of first-generation child components that a given XML file can contain.
xml_max_depth	20	The maximum level of component nesting.
xml_max_descendants	1000	The maximum number of components (any generation) that can be contained in an XML file.

## Uninstalling TeamXML

This section describes how to uninstall TeamXML on Windows and UNIX systems.

### Windows Systems

Perform the following steps to uninstall TeamXML:

1. If the XML search engine is installed and running on the system, stop the service (labeled **Interwoven XML Search Service** in the **Control Panel > Administrative Tools > Services** control panel).
2. Open the **Control Panel > Add/Remove Programs** window. This process can differ depending on the version of Windows you are using.
3. Select **Interwoven TeamXML** from the **Currently Installed Programs** list.
4. Click **Change/Remove**.

## UNIX Systems

Perform the following steps to uninstall TeamXML:

1. Navigate to *parent\_dir/teamxml-home/install*.
2. Run the uninstall script by executing the following command:

```
./teamxml_uninstall
```

## Log Files

TeamXML log files are not removed when you uninstall TeamXML.

## Additional Considerations

- Component directories should contain only components. Do not add other types of files to these directories.
- When TeamXML programmatically creates the component subdirectory structure shown in “Create Directory Structure and Sample Categories” on page 55, the resulting component directory and file names are not decipherable through the file system interface. For example, the file and directory names have no obvious meaning to users who navigate through the directory structure and view the file names through Network Neighborhood, a UNIX command line, or other file system interfaces. Instead, you must view the file and directory names through an integrated XML editor, which interprets the programmatically generated names and translates them into names that have relevance to end users.



## Chapter 3

# Category Definition Files

---

Category definition files let you control how TeamXML will interpret and process your site's data. The category definition files that you create determine how your site's data will be categorized, divided into components, and indexed. You can create as few or as many category definition files as you want. The number and type of category definition files will depend on the data organization needs of your specific site.

You should create category definition files only after you have analyzed how your site's data should be organized and determined an overall strategy for dividing the data into documents and components, and for indexing the data. Data organization and indexing vary tremendously from site to site, and detailed instructions for how to organize your site's data are beyond the scope of this chapter. Instead, this chapter contains the following:

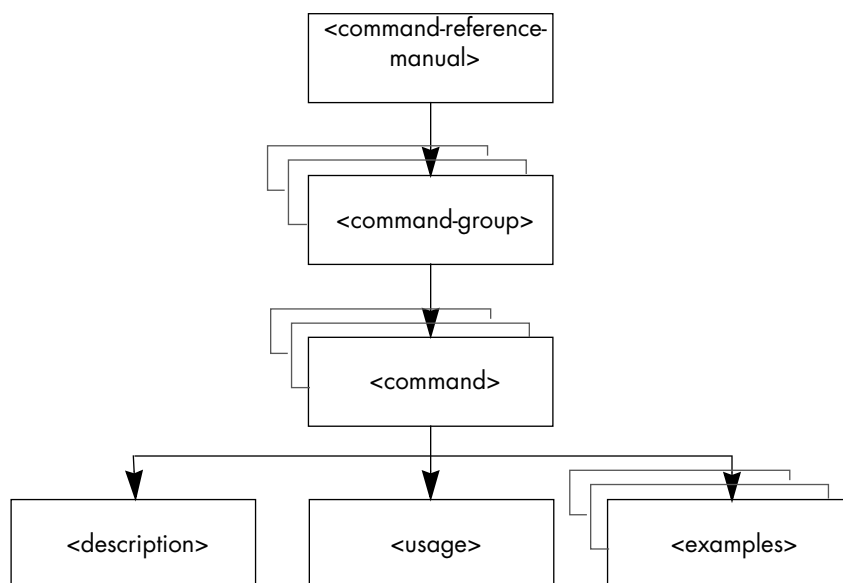
- General guidelines for creating categories.
- Descriptions of category definition file components.
- XPath conventions used in category definition files.
- Pointers to example files that are included with TeamXML.
- Annotated examples showing the relationship between category definition files, their input (simple XML files), and their output (discrete XML-formatted components that can be reused, combined, and rendered to create XML-formatted documents).

You can use the information in this chapter as a starting point for creating category definition files that will organize your site's data in an appropriate way. However, it is very likely that you will also need to consult with an Interwoven representative to analyze your site's data organization needs and to finalize a set of appropriate category definition files.

## Guidelines for Creating Categories

The structure of categories in Team XML reflects the relationship of physical XML elements in documents. A particular type of document—for example, a document that conforms to a schema (DTD)—contains a hierarchical organization of XML element types. TeamXML categories of components derived from this XML structure will have a similar hierarchical relationship through their componentization rules.

The following example illustrates this principle of conformance between XML structure and TeamXML category componentization rules. The example shows the structure of a hypothetical command reference manual. In this example, boxes represent XML element types, and arrows point to elements types that are included within a source element type.





## Creating Document Categories

If you were defining TeamXML categories to hold command reference manuals structured as shown in the preceding example, you would first create a *document* category to hold the main documents. You would create the document category by:

- Constructing a document category definition file.
- Placing the document category definition file in *TeamSite\_Area/TeamXML/Rules* (as described in “Create Directory Structure and Sample Categories” on page 55).
- Making TeamXML aware of the document category definition file by *specializing* it. If you created the category definition file using XML Authority, the file is automatically specialized. If you created the file using another editing tool, you must re-specialize it by running the `iwxm1cat -m` command. After creating the category definition file and specializing it, you could then use it to componentize documents by using the `iwxm1file -c` command.

The document category definition file name can be anything (and it does not need to end in `.dfn`), but it is recommended that you choose a name that describes the document category. For this example, assume that you would create a “command manual” document category, and that the document category definition file name is `command.manual.dfn`.

It is possible (and recommended) to create `command.manual.dfn` such that `<command-reference-manual>` is the root element. Having this root element is useful for two reasons:

- To ensure that only the correct type of documents (command reference manuals) are added to the category.
- To allow XML Authority to identify categories that will accept elements of a particular type.

### Document Characteristics

Documents have the following characteristics that distinguish them from components:

- Document categories should not be named in the componentization rules of other categories.
- Following componentization, document *files* can be moved, renamed, and deleted through file system or command line interfaces. However, document *category definition* files (such as `command.manual.dfn`) cannot be moved, renamed, or deleted.

- Following componentization, documents retain the file system path of the original XML file (their connection with the category is only a “logical” one). For example, you can create a directory in a TeamSite area to hold document files. After you componentize these document files, they remain exactly where you originally put them, and they have their original names. This provides a way of organizing the files in addition to their association with TeamXML. However, without careful file-system planning, users can place document files anywhere in the system and have difficulty identifying their association with a TeamXML category. That is why a document directory similar to that shown in “Create Directory Structure and Sample Categories” on page 55 is recommended as a best practice.

See “Parts of a Category Definition File” on page 71 for details about document category definition file layout and syntax.

## Creating Component Categories

After creating one or more document categories, the next step is to create component categories. You would create the component categories by:

- Constructing a component category definition file for each component type.
- Placing the component category definition files in *TeamSite\_Area/TeamXML/Rules* (as described in “Create Directory Structure and Sample Categories” on page 55). When the component category definition file is specialized, TeamXML creates the component directory structure as specified in the category definition file. If you have set up the TeamXML directory structure shown on page 55, you could specify within the category definition file that the component directory structure be created in *TeamSite\_Area/TeamXML/Components*.
- Making TeamXML aware of the component category definition files by specializing them with the `iwxmllcat -c` command-line tool.

Continuing with the command reference manual example, you could choose to skip creating a category for `<command group>` and go directly to `<command>` for your first level of component category (the `<command-group>` elements could still be included in the `<command-reference-manual>` document). In this case, you would define a “command” category for the `<command>` components by constructing a component category definition file called `command.dfn` (similar to how you created a `command.manual.dfn` file for the a “command manual” document category earlier). As with document category definition files, the component category definition file name can be anything, but it is recommended that you choose a name that describes the component category.

For componentization to work properly, you must also augment the definition of the “command manual” category in `command.manual.dfn` to specify that all content identified by the element type `<command>` should be created as components in the “command” category. It is these componentization rules that define how the categories are related.

**Note:** As an alternative to augmenting the “command manual” definition in an existing `command.manual.dfn` file, you could have created the “command manual” definition when you originally created `command.manual.dfn`.

It is good practice to restrict the root elements of components generated in the “command” category to have the expected `<command>` element type for the same reasons that similar restrictions are placed on document category root elements. See “Parts of a Category Definition File” on page 71 for details about component category definition file layout and syntax.

To complete the command reference manual example, you would define a component category for the `<example>` elements and add componentization rules to the “command” category definition file (`command.dfn`) to decompose commands into example components in this “example” category.

After these document and component categories are created in TeamXML, a command reference manual document in the form of a simple XML file that is added to the “command manual” category will have content defined by `<command>` elements created automatically as components in the “command” category. Additionally, `<example>` elements will be created recursively as components in the “example” category.

## Component Characteristics

Components have the following characteristics that distinguish them from documents:

- They are created by TeamXML automatically under a specified directory (identified when the category is created) regardless of the source of the component.
- They have system-generated names with little meaning to end users.
- If a component is created directly from a user’s file (as opposed to componentizing an enclosing document), the original file continues to exist but is not the component file. The component file is created, as described above, as a new file with a new name in a component directory. The user’s original source file continues to exist as an ordinary file that has no association with TeamXML.

Component display names (specified in component category definition files) and the search methods available in TeamXML are necessary for finding components because the file names have little meaning to end users.

## **Components and Reuse**

This section contains two subsections: “Best Practices” and “Specifics of Content Reuse”. The “Best Practices” section contains general information about organizing components to best facilitate content reuse. The “Specifics of Content Reuse” section contains TeamXML architecture and implementation details that you should take into account as you set up component categories for content reuse.

### ***Best Practices***

Components in the TeamXML repository can be created in two different ways: by decomposing existing XML documents into components or by composing the components directly. Most end users will initially use the first method: componentizing existing XML documents to create components. A potential problem with creating XML content in this way is that the content can reflect too closely the structure of its source documents. In other words, the components generated will be easy to reuse in new documents with the same structure, but it will be more of a challenge to reuse them in documents having a different structure. The relationships of the generated components reflect the physical XML structure of the source documents, which might not be optimal units of content for reuse in other documents.

A solution is to reorganize or “re-tag” existing documents for better content identification before they are used to create components in TeamXML. For example, detailed technical information about the software products produced by a company is of concern to many organizations within the company:

- Education—for customer training and certification.
- Support—for technical accounts, a service center, etc.
- Sales—for information about product configuration, partners, and pricing.
- Marketing—for competitive analysis, product definition, pricing, and distribution.
- QA—for product testing and specifications.
- Development—for product requirements and design specifications.

If the structure of a typical Education “course” is to divide the content into “chapters” and each chapter into “content” or “paragraph” units, this makes perfect sense for the structure of an educational course

composed of chapters. However, this structure will make it difficult to relate these subdivisions of content to other content structures. For example, chapters and paragraphs will have little in common with the product configuration guidelines and data sheets used by Sales or the command manuals developed by technical writers and tested by QA. Chapters developed by Education can be reused in additional courses but the structure of the information will not be suitable for use in a product support document.

An XML structure better designed for reuse would emphasize the content of the units rather than the structure of a course, especially in the more detailed units that are more likely to be reused. In this example, the course could be structured as follows:

- Course
  - introduction, overview, etc.
- Software operation
  - objective
  - command-line reference
  - API reference
  - errors
  - conclusion

By emphasizing the content – dividing the content into software operations – the content of the operations can be more easily reused in documents for Support, QA, Development, and other groups.

### ***Specifics of Content Reuse***

End users can reuse and modify components in the following ways:

- By inserting annotated components in new documents, or nesting them inside of other components, before the containing document or component is specialized. (Make the annotations through an integrated XML editor or directly through the TeamSite file system. See “Types of Annotations for Components” on page 70 for more information.)
- By adding annotated components to existing TeamXML documents or components before the containing document or component is re-specialized.

- By modifying an annotated component as a separate file and re-componentizing that file to modify the original component independently of any containing document or component.

It is important to remember that modification of an annotated copy of a component will cause modification of the reused component and all objects that contain it when the containing document is re-specialized.

If modification of the component is not desired, the annotation can simply be removed from the content and TeamXML will create a new component when the containing document is specialized.

### *Types of Annotations for Components*

There are two forms of component annotation, both of which are processing instructions but have different targets. The targets for the annotations are:

- `iw_component`: a processing instruction inserted into the XML contents of the component immediately following the root tag. This form is useful when a user edits a document or component based on content and needs to identify the source of a component.
- `iw_reference`: a processing instruction that represents the entire component without its contents. This form is useful when it requires too many resources to render the entire document and only a tree structure of the components is desired. This representation is required when the user does not have read access to the component and cannot render the contents.

If an annotated component is inserted into a new document or component, either type of annotated component will be recognized correctly when the new file is componentized in TeamXML. The inserted component will be recognized as a reused component and only a reference to it will be stored in the new file. Any changes to the component will be reflected in the new file when it is rendered.

## Parts of a Category Definition File

This section provides details about various parts of a category definition file. This section discusses the following topics:

- Root Element and Category Display Name
- Root Element Rule Set (optional)
- Componentization Rule Set (optional)
- Component Display Name Rule Set (components only)
- Indexing Rule Set (optional)
- External Validation Specification (optional)
- User-Defined Category Description (optional)
- Component Directory Name (components only)
- Prolog Information (components only)

**Note:** When creating category definition files for documents, do not specify rules for the items in the preceding list that say “components only.”

If you are creating a category definition file for documents, use the DTD for documents (see “DTD for Document Category Definition Files” on page 128). If you are creating a category definition file for components, use the DTD for components (see “DTD for Component Category Definition Files” on page 129).

DTDs for document and component category definition files contain an entity reference to the DTD that you must use for the elements that specify the indexing rule set (the indexing rule set is an optional part of a category definition file). For details, see “Indexing Rule Set” on page 76 and “DTD for Indexing Rule Set” on page 130.

### Root Element and Category Display Name

The name of your category definition file’s root element will be different for documents and components:

- For document category definition files, use `<document_category>`.
- For component category definition files, use `<component_category>`.



To define the category display name, set the name attribute of the `<display_name>` element. The category display name does not need to be unique within any part of TeamSite. (Only area-relative paths must uniquely identify category definition files.) If unique display names are required at your site, it is the responsibility of the person configuring TeamXML to ensure that this requirement is met. As a general guideline, it is highly recommended that category display names be defined to ensure uniqueness within area-relative paths, but TeamXML does not enforce this restriction.

An example of a category type and a category display name follows:

```
<document_category>  
  <display_name name="CATALOG"/>
```

## Root Element Rule Set

To limit the root element of objects placed in a category to certain types, you can define a root element rule set (`<root_elements>`) containing as many elements as desired. If you do not specify a root element, a component with any root element type can be specialized in the category.

An example of a root element rule set follows:

```
<root_elements>  
  <element type="product"/>  
</root_elements>
```

The API allows you to select categories based on the root element rule set.

In most cases, your componentization rule sets should produce homogeneous components in subcomponent categories, that is, components with the same root element. To do otherwise could be confusing. For example, you probably do not want to have one componentization rule set that adds `<computer_models>` elements to category A and a different rule set that adds `<peripherals>` elements to the same category.

If components in a category have different root elements, only some of them will be suitable for reuse in validated documents. An administrator could desire this heterogeneous result, but it should not result accidentally.



## Componentization Rule Set

The componentization rule set (<component\_ruleset>) for a document or component defines how to create the next level of subcomponents from the document or component. This rule set must specify only the next level to ensure that rules for creating subcomponents from a compound component are consistent.

An example of a componentization rule set follows:

```
<component_ruleset>
  <component xpath="/catalog/product" category="TeamXML/Rules/product.dfn"/>
</component_ruleset>
```

The rule set specification is a collection of items, each of which specifies a navigation syntax that is a subset of XPath.

**Note:** Different sections of a category definition file support different levels of XPath conventions. For example, indexing rule sets support a wider range of XPath conventions than do componentization rule sets. See “XPath Conventions” on page 83 for details about which XPath conventions are supported in each category definition file section.

In general, the XPath syntax supported in componentization rule sets specifies navigation to:

- An element in the document below the root element (in this example, the <product> element nested within the <catalog> element). For example:

```
<component component_path="/catalog/product"/>
```

- The corresponding component category where the component should be created. To assign a component to a category, specify the area-relative path of a category definition file. For example:

```
<component category="TeamXML/Rules/product.dfn"/>
```

Unless a document is validated and the elements are mandatory, a document might not contain elements that correspond to componentization rules. TeamXML does not return an error if components are not identified.

Omit the componentization rule set if you do not want files in a particular category to be componentized. TeamSite and TeamXML will treat an XML file in such a category like any other document in the TeamXML repository. For example, the TeamSite server will version such a file, and the TeamXML index and query service can index it for searching. But TeamXML will not create components, and the file is not subject to indirect modification as a result of changes to included components. This feature can be useful at sites requiring highly regulated document environments.

## Entity References

The TeamXML model of componentization does not resolve entity references (internal or external). This means that TeamXML will componentize XML files without attempting to identify and link XML references to a document DTD or external entity. However, any document containing entity references must have a DOCTYPE declaration. Do not use entity references in components unless you can ensure that such components will be included only in documents that specify definitions for those entity references in a DTD.

## Excluding Elements from Componentization

The *exception rule* type within the componentization rule set lets you specify elements that should not have componentization rules applied. An exception rule is identified by an empty value for the category attribute. For example, the following rule causes TeamXML to stop trying to apply componentization rules to a section tag if the tag contains a do-not-componentize attribute:

```
<component xpath="/section[@do-not-componentize]" category=""/>
```

## Adding and Deleting Componentization Rules

TeamXML lets you modify componentization rules in category definition files. When you change a componentization rule, previously-existing members of the category are unaffected by the changes; they will continue to have the same component types that they had prior to the rule changes.

## How Componentization Rules are Applied

For each element tag in the XML file being componentized, the rules for the appropriate category are examined in the order defined in the category definition. The xpath of the rule is evaluated using the root of the current file, which may be different than the original (container) file from which a component has been created. If a componentization rule is true for that tag, a component is created

from that element (including its subelements) and the process continues for subelements using the rules for the new component category. If none of the rules is true, the process continues for subelements and then for sibling elements.

When an element is matched by an exception rule, the effect is to not create a component for that element and to not evaluate any of its subcomponents for componentization. In effect, componentization of that element stops.

### *Example*

The following example shows a hypothetical XML file that will be componentized:

```
<blue_1>
  <blue_2 componentize="true">
    <blue_3>The text content of blue_3 instance 2</blue_3>
  </blue_2>
  <blue_2 componentize="false">
    <blue_3>The text content of blue_3 instance 1</blue_3>
  </blue_2>
</blue_1>
```

The preceding XML file will be componentized using the following componentization rules for the document category:

```
<component xpath="/blue_1/blue_2[@componentize='false']" category=""/>
<component xpath="/blue_1/blue_2" category="blue2"/>
<component xpath="/blue_1/blue_2/blue_3" category="blue3"/>
```

During componentization, the first occurrence of the `<blue_2>` element does not match the exception rule and the second rule is evaluated and found to match. This element will be componentized by creating a `blue2` component and componentization of its subelements will continue using the componentization rules of the `blue2` category definition. The second occurrence of `<blue_2>` matches the exception rule; no components will be created for this element and no further rules will be evaluated. The last rule is included here as a hypothetical example of a rule that will never be applied even though a `blue3` component could be created by the rules in the `blue2` category.



## Component Display Name Rule Set

Following specialization, *documents* in the TeamXML repository (and residing in a TeamSite area) have meaningful file names and do not need a display name. In contrast, the file path to *components* in the TeamXML repository do not have obvious names because TeamXML generates them. TeamXML therefore can assign display names to components by using a display name rule set (`<display_name_ruleset>`). The rules are followed in sequence until a string is generated, although rules are optional and the string can be null. The length of the string is limited to a size defined by TeamSite (1024 bytes). For example, a simple rule might specify the name of an attribute in the root element whose value should be used for the display name. There is no guarantee of uniqueness for this display name and it can be an empty string.

An example of a component display name rule set follows:

```
<display_name_ruleset>
  <display_name_rule xpath="/product/@sku"/>
  <display_name_rule xpath="/product/text()"/>
</display_name_ruleset>
```

To identify the element string value or attribute value used in determining component display names, the display name rule set can use a wider subset of XPath than componentization rule sets. A restriction is that the value should come from within the component (that is, navigation to a parent or sibling component is not allowed).

**Note:** Different sections of a category definition file support different levels of XPath conventions. See “XPath Conventions” on page 83 for details about which XPath conventions are supported in each category definition file section.

## Indexing Rule Set

Indexing every word in every file in a particular category, in addition to indexing all file information and metadata, is not optimal for most users. It may result in large index files that use a large amount of disk space and that slow performance. The indexing rule set’s purpose is to specify what information should be indexed for each file in a category. The indexing rule set is an optional part of the component rule set and is defined by the `<index_ruleset>` element. Without an indexing rule set, nothing in the category can be indexed.

**Note:** Creating an indexing rule set does not, by itself, enable indexing. After defining an indexing rule set, you must also execute the `iwxmindex.ipl` CLT to initially enable the index, and again whenever your content is updated (see “`iwxmindex.ipl`” on page 118). You can execute this CLT from the command line, from within a script, or as part of a workflow job. You can also write applications that perform indexing through the OpenAPI indexing service. You can limit indexing to particular areas or categories. Your indexing rule set must conform to the DTD `xml_index_ruleset1.0.dtd`. See “DTD for Indexing Rule Set” on page 130 for more information about that DTD.

The index rule set divides information about a file into three types of information. Each type is represented as a subelement under the `<index_ruleset>` element and is also referred to as a *facet*. The three supported subelements are as follows; only the `<file_attr>` subelement is required:

- File attributes (`<file_attr>`) – File-system properties, such as `Owner`, `Modified`, and so forth. Each file attribute is identified by a unique name as shown in the DTD. See “DTD for Indexing Rule Set” on page 130 for more information.
- Extended Attributes (`<extended_attr>`) – TeamSite extended attributes are key-value pairs that you can assign to any file. Extended attributes are identified by key name.
- Content (`<content>`) – Information stored inside a file. Content fields are identified by XPath expressions.

**Note:** Different sections of a category definition file support different levels of XPath conventions. See “XPath Conventions” on page 83 for details about which XPath conventions are supported in each category definition file section.

The simplest indexing rule set is:

```
<index_ruleset>
  <file_attr/>
</index_ruleset>
```



The preceding rule set instructs the indexing engine to index the default file-attribute fields. These default fields are:

- area\_path
- area\_rel\_path
- index\_time
- owner
- creator
- creation\_time
- last\_content\_mod\_time
- last\_attr\_mod\_time
- ruleset\_name

The previous example will not index any file content or extended attributes. The following example directs the indexing engine to index file content in addition to a list of file attributes:

```
<index_ruleset>
  <file_attr index_all = "true"/>
  <content is_xml = "true">
    <field name = "DATASHEET PRODUCT" xpath = "/Datasheet/Product/productName/text()">
      <text_type/>
    </field>
    <field name = "PRODUCT SALE PRICE" xpath = "/Datasheet/Product/salePrice">
      <numeric_type/>
    </field>
    <field name = "AVAILABILITY DATE" xpath = "/Datasheet/Product/dateAvailable">
      <date_type format="yyyy-MM-dd"/>
    </field>
    <transform>
      <automatic/>
    </transform>
  </content>
</index_ruleset>
```

The `<content>` subelement contains more options than the other `<index_ruleset>` subelements. For example, it contains a `field` element; each field element specifies a display name that you can use for presentation on a GUI, an XPath expression that identifies the field, and a data type. If you

specify `date_type` data within a `<content>` or `<extended_attr>` element, you must specify a pattern (`java.text.SimpleDateFormat` class) that explains how to interpret the date in the source document. When representing months in a date, be sure to use uppercase `MM` (lower case `mm` represents minutes, not months).

Without a `<transform>` section, the indexing engine indexes all content. The optional `<transform>` section indicates the transformations that TeamXML must perform on the source XML prior to indexing. The transformation does not affect the source document, only the representation that is seen by the indexing engine. The two available transformations are `automatic` and `xslt_stylesheet`. The `automatic` option (used in the previous example) tells the indexing engine to exclude all information that is not identified by the XPath expressions in the list of `<field>` elements. In other words, it tells the indexing engine to exclude as much content as possible to reduce the size of the index files. The `xslt_stylesheet` option allows the creator to include an inline XSLT stylesheet. The XML resulting from the transformation should contain patterns that match the XPath expressions listed in the `<field>` elements.

The `xpath` attribute in the `<field>` element can identify information that is contained in subcomponents. The indexing rules for the Product example (see “Component Category Definition File for Products” on page 96) include a reference to the product name. This allows queries such as “Find all of the products having *product\_name*”. This also implies that if the product name is a searchable field in the product category, the index information is duplicated.

## External Validation Specification

TeamXML does not perform validation of XML files stored within it. However, external applications can provide validation services. TeamXML can store the information necessary for validation when the document or component category is defined by using the `<external_validation_spec>` element.

An example of the XML external validation specification follows:

```
<external_validation_spec type="DTD">
  <schema_ref>IWCatalog-1.dtd</schema_ref>
</external_validation_spec>
```

This specification contains the following two parts (an attribute and a nested element):

- `type` – applicable if validation is required: `dtd`, `schema`, and so forth.
- `<schema_ref>` – applicable if validation is required: a URI to the validation specification.

Place the validation specification file in a location that is accessible to the external application. If it is stored within TeamSite, it should be an ordinary file not associated with a category in TeamXML.

XML Authority allows you to set the `type` attribute to a DTD when you define a category. Integrated XML editors will attempt to retrieve the validation specification and validate the new XML document before it is submitted to the TeamXML repository.

If the `<external_validation_spec>` element is not defined, either:

- An integrated XML editor decides whether to validate and where to find the validation specification.
- Or, an integrated XML editor validates using a validation specification within each document, if a specification is present (for example, in a DOCTYPE declaration).

## User-Defined Document Category Description

Optionally, you may specify a user-defined description for a category. This category description is identified by the element `<category_description>` in the category definition file. You can use an XML editor that has been integrated with TeamXML to retrieve this XML element from the API and show the description to a user who is adding a file to this category.

## Component Directory Name

When creating a new component category, you must specify an area-relative path name of a component directory by setting the `name` attribute of the `<component_directory>` element. TeamXML will create the directory and place all components created for this component category beneath this directory. This means that TeamXML may create subdirectories automatically under this root directory, for efficiency reasons, and populate them with component files as you create them. TeamXML programmatically assigns names to component files.



An example of a specification for a component directory within the recommended TeamXML directory structure is as follows:

```
<component_directory area_rel_path="TeamXML/Components/products"/>
```

You cannot rename component files after their creation. This restriction is necessary to ensure that the area-relative path name of a component remains unchanged. As a result, you also cannot rename the following:

- Any directory containing a component.
- All ancestor directories.

You should not place ordinary (that is, non-TeamXML) files into component directories through the file system interface because the component directories cannot be moved or renamed.

## Prolog Section

TeamXML supports a `<prolog>` section within component category definition files. The following sections describe this feature in detail.

### Prolog Section Purpose

The XML prolog of an XML *document*, as well as miscellaneous trailing matter, is preserved when the document is specialized in TeamXML. The prolog includes an XML declaration and often a DOCTYPE declaration that may be necessary to interpret entity references and other XML constructs within the document. When the document is rendered, you can specify that the XML prolog (and trailing matter) be included.

*Components* generated by componentization of a document do not have an XML prolog or trailing matter. Similarly, components specialized directly from a file will have any XML prolog and trailing matter discarded by TeamXML. If the component is rendered directly (that is, not included within a document), TeamXML can create an XML declaration for the component but cannot create the other parts of XML prolog that might be needed. The purpose of the prolog section of the component category definition file is to provide this information (usually a DOCTYPE declaration) that can be incorporated into the XML prolog of the component when it is rendered. The prolog material is included only for the component forming the root element of the rendered file (that is, not for embedded subcomponents).

The XML prolog of a component is rendered with the component if one of the following conditions exist:

- The component is rendered through the file system interface.
- The `include prolog` flag of the content specification mask is on when rendering through OpenAPI (see “OpenAPI Support for prolog” on page 82).

### **prolog Section Contents**

The prolog data is text that can be incorporated into the prolog of a component when that component is rendered as the root element of a file. The data is actually not the entire XML prolog, as defined in the XML standard, but rather that part after the XML declaration and before the root element. In general, this will be a DOCTYPE declaration containing entity definitions needed to render components of that category, but can include comments or processing instructions also.

### **prolog Section DTD**

See “DTD for Component Category Definition Files” on page 129 for details about the prolog DTD.

**Note:** Document category definition files do not have a prolog section.

### **OpenAPI Support for prolog**

The methods `IWXMLComponentFile.unspecializeToSimpleFile()` and `IWXMLComponentFile.openWithSpecifiedContent()` take a content specification mask. One flag defined for this mask is `csmIncludeProlog`. When this flag is specified in the mask, the component (or document) will be rendered including:

- An XML declaration created by TeamXML.
- If the component is a document, the rest of the original prolog of the document or, if it is a component, the text defined in the “prolog” section of the category definition.
- Any trailing information of a document (components have none).

### **prolog Example**

Examples included with the OpenAPI installation contain a reference to an external parameter entity that is also included in a prolog section of the component category definition files. See the OpenAPI documentation for more information.

## XPath Conventions

XPath is supported in category definition files and in search (query) statements. In category definition files, XPath is supported in rule sets for indexing, display name, and componentization. In search statements, XPath is supported in predicate and field elements. Which XPath conventions are supported depends on where they are used. The following table describes in detail the XPath conventions supported in each area. An X in the table indicates that the convention is supported. A blank cell indicates that the convention is not supported. N/A indicates that the convention is not applicable to an area.

Category Definition File Rule Set			Search Statement Element		Example	Description
Index	Display Name	Componentization	Predicate	Field		
X	X	X	X	X	/parent/child/grandchild	The grandchild elements.
X	X	N/A	X	X	/parent/child/grandchild/@name	The value of the name attribute of the grandchild element.
X	X	X	X	X	//elem1	The elements elem1 regardless of location in the file.
X	X	N/A	X	X	//@name	The value of the name attribute regardless of parent element or location in the file.
X	X	N/A	X	X	//elem1/@name	The value of the name attribute of elem1, regardless of the location of elem1 in the file.
X	X	X	X	X	/*/elem1	The elements elem1. The parent of elem1 must be a child element of the root node.



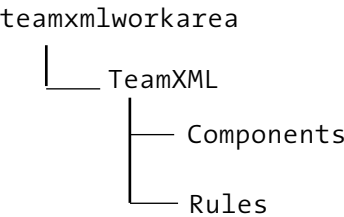
Category Definition File Rule Set			Search Statement Element		Example	Description
Index	Display Name	Componentization	Predicate	Field		
X	X	N/A	X	X	/top/*/elem1/@name	The value of the name attribute of elem1. elem1 can have any parent, but its parent must be at the second level of the document and must have a parent element top.
X	X	X	X	X	//	The root node and all descendants of the root node in a document. In componentization rules, this convention applies only to xpaths that select elements within the XML tree.
X	X	N/A	X	X	//@*	The value of all attributes in the document.
	X				/parent/child[1]	Only the first child element.
X	X	X	X	X	/parent/child[@name="blue"]	The child elements whose name attribute is set to blue.
X	X	X	X	X	/parent/child[@name]	The child elements containing a name attribute (of any value).
X	X	N/A	X	X	/parent/child/text()	Function text() to select the text of a node rather than its complete string value.
X	X	N/A	X	X	/parent/child/processing-instruction()	Function processing-instruction() to select the processing instruction of a node rather than its complete string value.

Category Definition File Rule Set			Search Statement Element		Example	Description
Index	Display Name	Componentization	Predicate	Field		
X	X	X	X	X	/parent//grandchild	All of the parent's grandchild elements (abbreviated location path).
	X	X			/top/descendant-or-self::node()/child::elem1	All elem1 descendants of top elements (unabbreviated location path).

## Sample Category Definition Files

The sample content bundled and installed with TeamXML includes several category definition files. These are installed as described in Chapter 2, “Installation and Initial Configuration.” This section describes the location and content of those files.

When the TeamXML sample content is installed, the end result is a new TeamSite branch called TeamXMLExample. This branch contains an edition, a staging area, and a workarea. The workarea, called `teamxmlworkarea`, contains several subdirectories, including TeamXML. The TeamXML subdirectory contains two additional subdirectories, Components and Rules, as shown below. This is identical to the directory structure shown in “Create Directory Structure and Sample Categories” on page 55.



The following sample category definition files for both documents and components reside in the Rules subdirectory.

## Document Category Definition Files

`datasheet.dfn`

`mktgreq.dfn`

`setupGuide.dfn`

## Component Category Definition Files

`competitors.dfn`

`featurelist.dfn`

`features.dfn`

`productImages.dfn`

`products.dfn`

`setupInstructions.dfn`

`specifications.dfn`

## Sample File Details

This section describes several of the sample files that are bundled and installed with TeamXML. Files include the following:

- A document category definition file (`mktgreq.dfn`) for a marketing requirements document.
- A plain XML source file (`mktgreq.xml`) that could be specialized by `mktgreq.dfn`.
- A component category definition file (`products.dfn`) for componentizing specific sections of `mktgreq.xml`.
- A plain XML file for a product component that was generated when TeamXML componentized `mktgreq.xml`.

**Note:** Your category definition files must use the DTDs provided. See “DTDs” on page 127.

### **Document Category Definition File for Marketing Requirements Example**

The following document category definition file, `mktgreq.dfn`, is installed in the directory `TeamXMLExample/WORKAREA/teamxmlworkarea/TeamXML/Rules`. It represents a typical document category definition file that you would create with XML Authority or with a text editor (if you created it with a text editor you would also need to specialize it manually with the `iwxmcat -c` command). The `mktgreq.dfn` file would then be used to componentize an XML file such as `mktgreq.xml` shown on page 91. Main sections in the sample file are keyed to a comment section following the file that explains more details about those sections.

**Note:** Some lines of this sample file were split into more than one line due to the formatting constraints of this document.



```
<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE document_category SYSTEM "document_category1.0.dtd">
<document_category>
  <display_name name = "MARKETING REQUIREMENTS"/>
  <root_elements>
    <element type = "MarketingRequirements"/>
  </root_elements>
  <component_ruleset>
    <component xpath = "/MarketingRequirements/Product" category = "
      TeamXML/Rules/products.dfn"/>
    <component xpath = "/MarketingRequirements/FeatureList" category
      = "TeamXML/Rules/featurelist.dfn"/>
    <component xpath = "/MarketingRequirements/CompetitorList/Compet
      itor" category = "TeamXML/Rules/c
      ompetitors.dfn"/>
  </component_ruleset>
  <external_validation_spec type = "dtd">
    <schema_ref>mktgreq.dtd</schema_ref>
  </external_validation_spec>
  <index_ruleset>
    <file_attr index_all = "true"/>
    <content is_xml = "true">
      <field name = "MKTG PROD NAME" xpath = "/MarketingRequir
        ements/Product/productName/text()">
        <text_type/>
      </field>
      <field name = "INTERNAL RELEASE DATE" xpath = "/Marketin
        gRequirements/InternalReleaseDate">
        <date_type format = "MM-dd-yyyy"/>
      </field>
      <field name = "MARKETING REQUIREMENTS" xpath = "/Marketi
        ngRequirements">
        <xml_type/>
      </field>
    </content>
  </index_ruleset>
</document_category>
```

Required Interwoven DTD <sup>1</sup>

Category name displayed in editor <sup>2</sup>

Root element designation <sup>3</sup>

Componentization rules <sup>4</sup>

Validation reference <sup>5</sup>

Indexed file attributes to display <sup>6</sup>

Indexed content to display <sup>7</sup>



## Sample File Notes

**1. Required Interwoven DTD:** Several required DTDs are shipped with TeamXML. These DTDs define the structure for document category definition files, component category definition files, indexing rule sets (within a category definition file), and search statements (which are separate from category definition files). Because this sample file is a document category definition file, it must conform to the DTD for document category definition files. This DTD, `document_category1.0.dtd`, is located in *iw-home/iwopenapi/dtd*.

**2. Category name displayed in editor:** The `<display_name>` element defines the name that is displayed for this category in an integrated XML editor and in XML Authority. For example, if someone uses an integrated XML editor to perform a search involving this category, the category is always represented as `MARKETING REQUIREMENTS`.

**3. Root element designation:** The `<root_elements>` element in a document category definition file specifies what root elements are allowed for an XML file to be specialized in this category. This example specifies that only files having the `<MarketingRequirements>` root element can be created in this category.

**4. Componentization rules:** This example instructs TeamXML to identify three different elements and create components for them using three different categories. Specifically, TeamXML will:

- Create a category for products based on the rules defined in `TeamXML/Rules/products.dfn`. Populate the products category with content from the `<Product>` element within the `<MarketingRequirements>` root element of `mktgreq.xml`. Additional (recursive) componentization will occur later if `products.dfn` contains a componentization rule set. See “Component Category Definition File for Products” on page 96 for more information about `products.dfn`.
- Create a category for features based on the rules defined in `TeamXML/Rules/featurelist.dfn`. Populate the features category with content from the `<Featurelist>` element within the `<MarketingRequirements>` root element of `mktgreq.xml`. Additional (recursive) componentization will occur later if `featurelist.dfn` contains a componentization rule set.



- Create a category for competitors based on the rules defined in `TeamXML/Rules/competitors.dfn`. Populate the competitors category with content from the `<Competitor>` element within the `<CompetitorList>` element (which is within the `<MarketingRequirements>` root element of `mktgreq.xml`). Additional (recursive) componentization will occur later if `competitors.dfn` contains a componentization rule set.

The results of applying these rules are shown later in this section. For example, if this sample file (`mktgreq.dfn`) was used to componentize the `mktgreq.xml` file shown in “Plain XML File for Marketing Requirements Example” on page 91, categories for products, features, and competitors would be created. The `products.dfn` file (shown on page 96) would be invoked to componentize the `Product` element in `mktgreq.dfn`. The result would be the plain XML component file shown on page 101. Additional components are also created and reside in `TeamXML/Components`, but they are not shown in this chapter.

**5. Validation reference:** The `<external_validation>` element lets you specify external DTDs or other types of validating schemas (such as XML schemas) with which a third-party tool can validate XML files in this category. (Note that TeamXML itself does not perform any validation.)

In this example, the DTD `mktgreq.dtd` in `/teamxmlworkarea/ExampleDocuments` can be used by a third-party tool to validate the component files (such as the file shown on page 101) generated by the componentization rules specified in the preceding section.

**6. Indexed file attributes to display:** The `<file_attr>` section shown in this example instructs TeamXML to do the following:

- Index all file attributes whenever `mktgreq.xml` is indexed (when `iwxmlindex.ip1` is run).
- Display all file attributes (such as owner, time stamp, and so on) for `mktgreq.xml` as searchable items when an end user performs a search through the integrated XML editor.

**7. Indexed content to display:** The <content> section shown in this example instructs TeamXML to do the following:

- In the integrated XML editor, display MKTG\_PROD\_NAME as a searchable item when an end user performs a search. Also use the text from the source document's <MarketingRequirements/Product/productName> element as match criteria for the search. For example, if an end user constructs a search statement in the integrated XML editor that instructs TeamXML to find all files in which MKTG\_PROD\_NAME equals Camera2001 OM-2i, the XML source file (mktgreq.xml) would be displayed as a match.
- Display INTERNAL\_RELEASE\_DATE as a searchable item in the integrated XML editor. Use the date from the source document's <MarketingRequirements/InternalReleaseDate> element as match criteria for the search. Use the format field to convert date strings into numeric values so searches can use operators such as >, <, >=, and <=.
- Display MARKETING\_REQUIREMENTS as a searchable item in the integrated XML editor. Also use all element text (but no attribute text) from the source document's <MarketingRequirements> element as match criteria for the search. Specifying xml\_type data as shown here indicates that the xpath statement points to a node in an XML tree, and is not limited just to text.

Note that *all* content text is indexed whenever you index a file unless you specify the <transform> element within the <content> element to limit what content gets indexed. The content section shown in this example does not contain a <transform> element, and therefore does not limit what content is indexed (it only limits what content is displayed as searchable in the integrated XML editor).

### Plain XML File for Marketing Requirements Example

The following sample XML file, mktgreq.xml, is installed in the directory TeamXMLExample/WORKAREA/teamxmlworkarea/ExampleDocuments. This file represents a typical XML file that you would componentize with the iwxmlfile -c command after you had created and specialized the necessary category definition files (such as mktgreq.dfn and products.dfn as shown in this chapter).

**Note:** Some lines of this sample file were split into more than one line due to the formatting constraints of this document.

This file is used together with the other sample files described in this section. For example:

- The document category definition file `mktgreq.dfn` shown on page 87 can be used to componentize this file.
- The category definition file `products.dfn` shown on page 96 componentizes the `Product` element shown in this file. (The `products.dfn` file is called when the document category definition file `mktgreq.dfn` is executed.)
- The component file shown on page 101 is created when this file is componentized. (Additional components are also created but are not documented here.)

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE MarketingRequirements SYSTEM "iwts:/ExampleDocuments/DTD/mktgreg.dtd"
[
    <!NOTATION JPEG SYSTEM "iexplore.exe">
    <!ENTITY IntranetLogoPath SYSTEM "iwts:/images/intranetLogo.gif" NDATA JPEG>
]>
<MarketingRequirements>
    <Product>
        <productName>Camera2001 OM-2i</productName>
        <productDescription>The OM-2i combines precision controls and electronic
        bv sophistication to give you the power to create. Multi-spot and off-
        the-film metering, exposure memory and compatibility with various system
        components make the OM-2i a perfect choice for the serious
        photographer. </productDescription>
        <productImage fileref="iwts:/images/camera.jpg">
            <title>Camera2001 OM-2i</title>
            <description>The Brand New Camera2001 OM-2i</description>
            <graphicType>JPEG</graphicType>
        </productImage>
        <availabilityDate>12-01-2001</availabilityDate>
        <salePrice Currency="$">1000</salePrice>
    </Product>
    <FeatureList>
        <Feature>
            <featureTitle>Rugged</featureTitle>
            <featureDescription>Rugged, lightweight titanium
            body</featureDescription>
            <Benefits>
                <benefitTitle>Durability</benefitTitle>
                <benefitDescription>Built to last</benefitDescription>
            </Benefits>
        </Feature>
        <Feature>
            <featureTitle>Synchronization</featureTitle>
            <featureDescription>Synchronization at all speeds</featureDescription>
            <Benefits>
                <benefitTitle>Exact Shutter Speed Selection</benefitTitle>
                <benefitDescription>Allows you to select the exact shutter speed
                and aperture, for action-stopping and depth-of-field
                effects</benefitDescription>
            </Benefits>
        </Feature>
        <Feature>
            <featureTitle>Metering</featureTitle>
            <featureDescription>Center-weighted average

```



```
        metering</featureDescription>
    <Benefits>
        <benefitTitle>Accurate Readings</benefitTitle>
        <benefitDescription>Centre-weighted average metering ensures acc
            urate readings</benefitDescription>
    </Benefits>
</Feature>
<Feature>
    <featureTitle>Exposure</featureTitle>
    <featureDescription>Set exposure values</featureDescription>
    <Benefits>
        <benefitTitle>Suit Composition Elements</benefitTitle>
        <benefitDescription>Set Exposure values to suit the most importa
            nt composition elements of your shot with spot metering
        </benefitDescription>
    </Benefits>
</Feature>
<Feature>
    <featureTitle>Compare Readings</featureTitle>
    <featureDescription>Compare readings in up to eight different spots of
        your frame </featureDescription>
    <Benefits>
        <benefitTitle/>
        <benefitDescription/>
    </Benefits>
</Feature>
<Feature>
    <featureTitle>Shadow Control</featureTitle>
    <featureDescription>Highlight and shadow control</featureDescription>
    <Benefits>
        <benefitTitle>Accentuation </benefitTitle>
        <benefitDescription>Highlight and Shadow control lets you accent
            uate and adjust bright or dark shot elements</benefitDescription>
    </Benefits>
</Feature>
<Feature>
    <featureTitle>Exposure Accuracy</featureTitle>
    <featureDescription>Exposure accuracy to within plus or minus one
        quarter of a stop </featureDescription>
    <Benefits>
        <benefitTitle/>
        <benefitDescription/>
    </Benefits>
</Feature>
<Feature>
```

```

    <featureTitle>OTF Metering</featureTitle>
    <featureDescription>OTF (off-the-film) metering ensures the right
      exposure automatically </featureDescription>
    <Benefits>
      <benefitTitle/>
      <benefitDescription/>
    </Benefits>
  </Feature>
  <Feature>
    <featureTitle>Flash Shots</featureTitle>
    <featureDescription>Avoid blurred action with flash shots at shutter
      speeds up to 1/2000 (with F280 flash) </featureDescription>
    <Benefits>
      <benefitTitle/>
      <benefitDescription/>
    </Benefits>
  </Feature>
</FeatureList>
<Requestors>
  <requestorName>Photographic Concepts Inc</requestorName>
  <requestorEmail>ceo@photoconcepts.com</requestorEmail>
</Requestors>
<CompetitorList>
  <Competitor>
    <competitorName>Pin-hole Camera Inc.</competitorName>
    <competitorDisadvantage>Outdated Technology</competitorDisadvantage>
    <competitorAdvantage>Cheap and Easy to
      Manufacture</competitorAdvantage>
  </Competitor>
</CompetitorList>
<InternalReleaseDate>11-15-2001</InternalReleaseDate>
<IntranetLogo logoPath="IntranetLogoPath"></IntranetLogo>
</MarketingRequirements>

```

### Component Category Definition File for Products

The following component category definition file, `products.dfn`, is installed in the directory `TeamXMLExample/WORKAREA/teamxmlworkarea/TeamXML/Rules`. It represents a typical component category definition file that you would create with Tibco XML Authority or with a text editor (if you created it with a text editor you would also need to specialize it manually with the `iwxmlcat -c` command). The `products.dfn` file would then be used to componentize specific sections of an XML file such as `mktgreq.xml` shown on page 91. Component category definition files such as this are usually called from within document category definition files as shown in `mktgreq.dfn` on page 87.

Main sections in the sample file are keyed to a comment section following the file that explains more details about those sections.

**Note:** Some lines of this sample file were split into more than one line due to the formatting constraints of this document.



```

<?xml version = "1.0" encoding = "UTF-8"?>
<!DOCTYPE component_category SYSTEM "component_category1.0.dtd">
<component_category>
  <display_name name = "PRODUCT"/>
  <root_elements>
    <element type = "Product"/>
  </root_elements>
  <display_name_ruleset>
    <display_name_rule xpath = "/Product/productName/text()"/>
    <display_name_rule xpath = "/Product/productDescription/text()"/>
    <display_name_rule xpath = "/Product"/>
  </display_name_ruleset>
  <component_directory area_rel_path = "TeamXML/Components/products"/>
  <component_ruleset>
    <component xpath = "/Product/productImage" category = "TeamXML/R
      ules/productImages.dfn"/>
  </component_ruleset>
  <index_ruleset>
    <file_attr index_all = "true"/>
    <content is_xml = "true">
      <field name = "PRODUCT NAME" xpath = "/Product/productName/text()">
        <text_type/>
      </field>
      <field name = "PRODUCT DESCRIPTION" xpath = "/Product/pr
        oductDescription/text()">
        <text_type/>
      </field>
      <field name = "PRODUCT AVAILABILITY DATE" xpath = "/Prod
        uct/availabilityDate">
        <date_type format = "MM-dd-yyyy"/>
      </field>
      <field name = "PRODUCT PRICE" xpath = "/Product/salePrice">
        <numeric_type/>
      </field>
    </content>
  </index_ruleset>
</component_category>

```

Required Interwoven DTD <sup>1</sup>

Category name displayed in editor <sup>2</sup>

Root element designation <sup>3</sup>

Component name displayed in editor <sup>4</sup>

Indexed file attributes to display <sup>7</sup>

Destination directory for generated components <sup>5</sup>

Componentization rules <sup>6</sup>

Indexed content to display <sup>8</sup>

## Sample File Details

**1. Required Interwoven DTD:** Several required DTDs are shipped with TeamXML. These DTDs define the structure for document category definition files, component category definition files, indexing rule sets (within a category definition file), and search statements (which are separate from category definition files). Because this sample file is a component category definition file, it must conform to the DTD for component category definition files. This DTD, `component_category1.0.dtd`, is located in `iw-home/iwopenapi/dtd`.

**2. Category name displayed in editor:** The `<display_name>` element defines the name that is displayed for this category in an integrated XML editor. For example, if someone uses an integrated XML editor to perform a search involving this category, the category is always represented as `PRODUCT`.

**3. Root element designation:** The `<root_elements>` element in a component category definition file specifies the root element of the generated component file. This example specifies that all component files generated by `products.dfn` will have the `<Product>` root element.

**4: Component name displayed in editor:** The file path to components in the TeamXML repository do not have obvious names because TeamXML generates them. TeamXML therefore can assign display names to components by using a display name rule set (`<display_name_ruleset>`). The rules are followed in sequence until a string is generated. Display names are used whenever a component name and path are displayed in an integrated XML editor. In this example, the first XPath rule, `"/Product/productName/text()"`, provides a match from the following section in the generated component file shown on page 101:

```
<Product>
  <productName>Camera2001 OM-2i</productName>
```

Because of this match, the display name Camera2001 OM-2i is used whenever this component is listed in an integrated XML editor. If the first XPath rule had not provided a match, TeamXML would have evaluated the second rule, `"/Product/productDescription/text()"`. This rule would have provided a match with the following section in the generated component file:

```
<Product>
  <productDescription>The OM-2i combines precision controls and electronic
    bv sophistication to give you the power to create. Multi-spot and off-
    the-film metering, exposure memory and compatibility with various system
    components make the OM-2i a perfect choice for the serious
    photographer. </productDescription>
```

This match would cause the first 1024 bytes of the text string to be used as the display name for the generated component file. If the second XPath rule had not provided a match, TeamXML would have evaluated the third rule, `"/Product"`. This rule would have provided a match with all element text (but no attribute text) in the generated component file.

**5. Destination directory for generated components:** The section shown here instructs TeamXML to place all components generated by `products.dfn` in the directory `TeamXML/Components/products`.

**6. Componentization rules:** This example instructs TeamXML to create a category for product images based on the rules defined in `TeamXML/Rules/productImages.dfn`. It will then populate the product images category with content from the `<productImage>` element within the component's `<Product>` root element (the component was originally created when `mktgreq.xml` was componentized). Additional (recursive) componentization will occur if `productImages.dfn` contains a componentization rule set.

**7. Indexed file attributes to display:** The `<file_attr>` section shown in this example instructs TeamXML to do the following:

- Index all file attributes whenever the generated component file is indexed (when `iwxmindex.ipl` is run).
- Display all file attributes (such as owner, time stamp, and so on) for the generated component file as searchable items when an end user performs a search through the integrated XML editor.



**8. Indexed content to display:** The content section shown in this example instructs TeamXML to do the following:

- In the integrated XML editor, display `PRODUCT NAME` as a searchable item when an end user performs a search. Also use the text from the generated component's `<Product/productName>` element as match criteria for the search. For example, if an end user constructs a search statement in the integrated XML editor that instructs TeamXML to find all files in which `PRODUCT NAME` equals `Camera2001 OM-2i`, the display name (`Camera2001 OM-2i`) of the generated component file shown on page 101 would be displayed as a match.
- Display `PRODUCT DESCRIPTION` as a searchable item in the integrated XML editor. Also use the text from the generated component's `<Product/productDescription>` element as match criteria for the search.
- Display `PRODUCT AVAILABILITY DATE` as a searchable item in the integrated XML editor. Use the date from the generated component's `<Product/availabilityDate>` element as match criteria for the search. Use the `format` field to convert date strings into numeric values so searches can use operators such as `>`, `<`, `>=`, and `<=`.
- Display `PRODUCT PRICE` as a searchable item in the integrated XML editor. Also use the number from the generated component's `<Product/salePrice>` element as match criteria for the search.

As with document category definition files, unless you specify the `<transform>` element within the `<content>` element, *all* content text is indexed when a file is componentized. The content section shown in this example does not contain a `<transform>` element, and therefore does not limit what content is indexed (it only limits what content is displayed as searchable in the integrated XML editor).

## Generated Product Component File

The following generated component file resides in `TeamXML/Components/products`. It was created (together with several other components) when `mktgreq.xml` was componentized using the rules defined in `products.dfn`. Its file name is programmatically generated by TeamXML and has no obvious meaning. This file is represented in an integrated XML editor as `Camera2001 OM-2i` because of the `<display_name_ruleset>` defined in `products.dfn`.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Product>
  <productName>Camera2001 OM-2i</productName>
  <productDescription>The OM-2i combines precision controls and electronic bv
sophistication to give you the power to create. Multi-spot and off-the-film
metering, exposure memory and compatibility with various system components
make the OM-2i a perfect choice for the serious
photographer.</productDescription>
  <productImage fileref="iwts:/images/camera.jpg">
    <title>Camera2001 OM-2i</title>
    <description>The Brand New Camera2001 OM-2i</description>
    <graphicType>JPEG</graphicType>
  </productImage>
  <availabilityDate>12-01-2001</availabilityDate>
  <salePrice Currency="$">1000</salePrice>
</Product>
```

**Note:** The attribute `fileref="iwts:/images/camera.jpg"` specifies that the `camera.jpg` file resides in a TeamSite area outside of the TeamXML directory structure shown in “Sample Category Definition Files” on page 85.



## Chapter 4

# Search Statements

---

This chapter contains information about how TeamXML performs searches (queries) involving documents and components. It also shows sample search statements that are included and installed with TeamXML.

## Search Overview

Searches can be performed from the command line or through a GUI such as the integrated XML editors that support TeamXML and are provided by third parties.

To perform a search from the command line, you must create an XML formatted search statement such as those shown in “Sample Searches” on page 105 and then pass the search statement file to the `iwxmlsearch.ipl` command-line tool. Search statements must be constructed to conform to the search DTD described in “Search DTD” on page 132. The XPath conventions supported in search statements are described in “XPath Conventions” on page 83.

To perform a search through an integrated XML editor, an end user constructs a search statement by selecting search items, operators, and match criteria from an integrated XML editor’s GUI. The integrated XML editor then uses the OpenAPI search service to perform the search. See the OpenAPI documentation for details about the search service.

Searches will only find content that has already been indexed. The rules for indexing objects in a category are defined in the category definition file. These rules can be explicit about which elements and attributes should have content indexed or can simply specify that all content be indexed. In either case, the indexer identifies the XML structure and does not index the element tags themselves as a full-text indexer would. The indexing rules can also be used to list the display name for searchable fields.

## Search Tips

- Searching an individual category is most meaningful because two factors are always known: the date of last indexing, and which part of the content of objects was indexed.
- A simple search can look for keywords in all content that was indexed regardless of the XML structure. In this case, content refers to excluding XML element tags, attribute names, and other XML structure.
- A structured search can be constructed from the indexing rules, allowing users to search for information in specific XML content using user-friendly field names.
- The indexing rule set in a category definition file can specify three different facets on which to index a component: file properties, extended attributes, and XML content. If `date_type` data is specified within an extended attribute or XML content facet, search statements using the `date_type` data must convert the date string into a numeric value as shown in “available-products.xml” on page 108. The OpenAPI search service used by the integrated XML editors also performs this conversion.
- All virtual path (vpath) names in file attribute fields (specifically, `area_path` and `area_rel_path`) in search statements must use forward slashes and no host names.
- If the `facet` attribute in a search statement is set to `extended_attr`, the `keyname` attribute becomes available for you to use. Use `keyname` if you need to use the `xpath` element to identify a field within the extended attribute’s value when the extended attribute’s value is itself an XML structure. See “simple-ea.xml” on page 109 and “complex-ea.xml” on page 111 for examples of `keyname` usage.



## Sample Searches

The sample content bundled and installed with TeamXML includes several search statement files that could be called from the command line through the `iwxmlsearch.ipl` command-line tool. These files are installed in `TeamXMLExample/teamxmlworkarea/ExampleQueries` as described in Chapter 2, “Installation and Initial Configuration.” The files are as follows; the rest of this chapter shows each file in its entirety.

`listwa.xml`

`find-jpegs.xml`

`available-products.xml`

`simple-ea.xml`

`complex-ea.xml`

**listwa.xml**

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE search SYSTEM "xml_search1.0.dtd">

<!--
  List all indexed files in the area named "teamxmlworkarea". The area_path
  will need to be modified if the example is not installed in the default
  workarea.
-->

<search>
  <where>
    <predicate facet="file_attr"
              xpath="area_path"
              operator="EQ"
              value="/default/main/TeamXMLExample/WORKAREA/teamxmlworkarea"/>
  </where>

</search>
```

**find-jpegs.xml**

```

<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE search SYSTEM "xml_search1.0.dtd">

<!--
  Find all TeamXML components in the "TeamXML/Rules/productImages.dfn"
  category that have a jpeg graphicType. Only look in the area named
  "teamxmlworkarea". The area_path will need to be modified if the
  example is not installed in the default workarea.

  Return the "productImage/@fileref" attribute for each TeamXML component
  that matches the query. The fileref attribute contains the location of
  the graphic file as an "area rooted absolute path" using the "iwts:"
  scheme.
-->

-->

<search>
  <select>
    <field facet="content" xpath="/productImage/@fileref"/>
  </select>
  <where>
    <and>
      <predicate facet="file_attr"
        xpath="area_path"
        operator="EQ"
        value="/default/main/TeamXMLExample/WORKAREA/teamxmlworkarea"/>
      <predicate facet="file_attr"
        xpath="ruleset_name"
        operator="EQ"
        value="TeamXML/Rules/productImages.dfn"/>
      <predicate facet="content"
        xpath="/productImage/graphicType/text()"
        operator="EQ"
        value="jpeg"/>
    </and>
  </where>
</search>

```



## available-products.xml

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE search SYSTEM "xml_search1.0.dtd">

<!--
  Find all TeamXML components in the "TeamXML/Rules/products.dfn"
  category that have an availabilityDate <= Jan 31, 2002. Only look in
  the area named "teamxmlworkarea". The area_path will need to be modified
  if the example is not installed in the default workarea.

  Note: The index ruleset within the products.dfn file specifies that
  "availabiltyDate" is a "date type". Therefore, during the indexing
  process, the date string is converted to a numeric value to allow range
  comparisons.
  As a result, we must convert our search date (Jan 31, 2002) into a
  number (milliseconds since Jan 1, 1970) to be used in the query.
  In general, queries are built by graphical applications, so the user
  is not expected to convert the date to a number. The standard approach
  is to have a GUI that prompts the user to enter a date string or pick
  the date from a calendar widget and have the software do the conversion
  when building the query.
-->

<search>
  <where>
    <and>
      <predicate facet="file_attr"
        xpath="area_path"
        operator="EQ"
        value="/default/main/TeamXMLExample/WORKAREA/teamxmlworkarea"/>
      <predicate facet="file_attr"
        xpath="ruleset_name"
        operator="EQ"
        value="TeamXML/Rules/products.dfn"/>
      <predicate facet="content"
        xpath="//availabilityDate/text()"
        operator="LE"
        value="1012464000000"/>
    </and>
  </where>
</search>
```

## simple-ea.xml

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE search SYSTEM "xml_search1.0.dtd">
```

```
<!--
```

TeamSite has a feature that allows users to store extra information about a file with key/value pairs known as extended attributes (EAs). EAs are generally used to store meta information about the file. For example, the setup script for the demo content adds two keys to each graphic file. One of the keys, "Description", is used to store a description of each image. A graphic file does not contain any useful text, so this type of meta information can be very useful when trying to locate a specific image.

The following query searches "teamxmlworkarea" for all files that have the word "OM-2i" in the extended attribute called "Description". At the time of this writing, this query will only find graphic files because those are the only files that have been tagged with extended attributes. However, there is nothing in the query that limits the query to graphic files.

Queries for extended attributes use the optional "keyname" attribute in the predicate. The "keyname" attribute should be set to the keyname that the user is searching ("Description" in this example). For normal EA queries, the "xpath" attribute should be set to an empty string. The xpath attribute is a required attribute, so it cannot be removed from the query. In some advanced scenarios, the "xpath" attribute is used when searching EAs, see the "complex-ea.xml" example for details.



Note: the `area_path` will need to be modified if the example is not installed in the default workarea.

-->

```
<search>
  <where>
    <and>
      <predicate facet="file_attr"
        xpath="area_path"
        operator="EQ"
        value="/default/main/TeamXMLExample/WORKAREA/
          teamxmlworkarea"/>
      <predicate facet="extended_attr"
        keyname="Description"
        xpath=""
        operator="CONTAINS"
        value="OM-2i"/>
    </and>
  </where>
</search>
```

**complex-ea.xml**

```
<?xml version="1.0" encoding="US-ASCII"?>
<!DOCTYPE search SYSTEM "xml_search1.0.dtd">
```

```
<!--
```

It is best to make sure that you understand the simple-ea.xml example before attempting to understand this example.

In some cases, extended attribute values are used to store more than simple string values. It is not uncommon for the "value" part of an extended attribute to be an XML document. For example, Interwoven's Metatagger product is used to classify content against a user-defined taxonomy. Metatagger will set an EA that resembles this example:

EA key = "subjectMetadata"

```
EA Value = <metadata>
    <facet>
        <facetName>sector</facetName>
        <descriptor>
            <vocab>naics</vocab>
            <code>312111</code>
            <label>Soft Drink Manufacturing US</label>
        </descriptor>
    </facet>
    <facet>
        <facetName>locations</facetName>
        <descriptor>
            <vocab>ISO3166-1</vocab>
            (1) <code>US</code>
                <label>United States</label>
            </descriptor>
            <descriptor>
                <vocab>Regions</vocab>
                (2) <code>AF</code>
                    <label>Africa</label>
                </descriptor>
            </facet>
    </metadata>
```

The OpenAPI XML Search Service has the ability to search on specific fields inside of the XML structure, instead of treating it as plain text. For this to work properly, the ruleset used to index the file



must specify that the datatype of the EA (subjectMetadata in this example) is XML.

The following query searches "teamxmlworkarea" for all files that have been classified as having a location of Africa. More specifically, the "subjectMetadata" value must have an XML element in the "locations" facet with a descriptor code of "AF". Given the above XML, the xpath expression that will select the text of all of the correct location code elements is:

```
/metadata/facet[facetName = 'locations']/descriptor/code/text()
```

Given the example, the text of the elements highlighted by (1) and (2) will be selected. In the query below, we specify that the value of one of the selected code elements must equal "AF", so the query will match a file with the example metadata.

From the simple-ea.xml example, you will recall that the "xpath" attribute was not used. In this example, you can see that the "xpath" attribute is used to "reach inside" of the value structure to limit the search to a specific set of nodes in the XML tree. Usage of the "keyname" attribute is the same as in the simple example; it is used to select the key to be searched.

Note: the area\_path will need to be modified if the example is not installed in the default workarea.

-->

```
<search>
  <where>
    <and>
      <predicate facet="file_attr"
        xpath="area_path"
        operator="EQ"
        value="/default/main/TeamXMLExample/WORKAREA/
          teamxmlworkarea"/>
      <predicate facet="extended_attr"
        keyname="subjectMetadata"
        xpath="/metadata/facet[facetName = 'locations']/descrip
          tor/code/text()"
        operator="EQ"
        value="AF"/>
    </and>
  </where>
</search>
```



## Appendix A

# Command-Line Tools

---

This appendix provides a comprehensive summary of TeamXML CLT usage. The following CLTs are described:

- `iwxmcat` – Manipulates categories in TeamXML.
- `iwxmfile` – Manipulates documents and components in TeamXML.
- `iwxmindex.ip1` – Updates indexes for TeamXML files.
- `iwxminfo` – Obtains information about TeamXML, such as known categories or components contained in a category.
- `iwxmsearch.ip1` – Searches for indexed information in TeamXML.

**Note:** Ensure that you have installed and set up TeamXML and that you have created category definition files before proceeding.

## iwxmlcat

iwxmlcat manipulates TeamXML categories.

### Usage

Syntax is as follows:

iwxmlcat [*optional\_argument*] *action*

where *action* is exactly one of the following:

-c <i>vpath</i>	Creates a new category based on a category definition file specified by <i>vpath</i> . (After you create a category definition file, TeamXML will recognize it as a category definition file until you execute the iwxmlcat -c command.)
-h	Displays help information.
-m [-r] <i>old new</i>	Modifies a category whose category definition file has <i>vpath old</i> to use new parameters from a category definition file with <i>vpath new</i> . The optional -r setting removes <i>new</i> after successfully modifying a category.
-v	Displays version information about this software.

- optional\_argument* may be zero or more of the following:

-r	Removes <i>new</i> after successfully modifying a category. Use -r only in conjunction with -m.
----	---

### Examples

- iwxmlcat -c /default/main/WORKAREA/TeamXML/Rules/doctype1.dfn

This command creates a new category, Rules/doctype1.dfn, from the category definition file specified by the *vpath* /default/main/WORKAREA/TeamXML/Rules/doctype1.dfn. After you create a category definition file, TeamXML will not recognize it as a category definition file until you execute the iwxmlcat -c command.

- `iwxmcat -m /default/main/WORKAREA/TeamXML/Rules/doctype1.dfn /default/main/WORKAREA/TeamXML/Rules/MODIFIED.doctype1.dfn`

This command modifies a category, whose category definition file is specified by the vpath `/default/main/WORKAREA/TeamXML/Rules/doctype1.dfn`, to use the settings contained in the file with vpath `/default/main/WORKAREA/TeamXML/Rules/MODIFIED.doctype1.dfn`. To reflect the new settings, the contents of `.../MODIFIED.doctype1.dfn` are also copied into `.../doctype1.dfn`.



## iwxmlfile

iwxmlfile manipulates TeamXML documents and components.

### Usage

Syntax is as follows:

`iwxmlfile [optional_argument] action`

where *action* is exactly one of the following:

<code>-a infile outdir outname</code>	Writes an annotated version of an XML document or component with vpath <i>infile</i> to a file <i>outname</i> in a directory with vpath <i>outdir</i> .
<code>-c infile catname</code>	Creates a new document or component from an ordinary XML file specified by vpath <i>infile</i> and assigns this new document or component to a category specified by <i>catname</i> . (A category name is the area-relative path of the category definition file—for example, TeamXML/Rules/doctype1.dfn.)
<code>-e xmlfile encoding</code>	Sets the encoding for <i>xmlfile</i> .
<code>-h</code>	Displays help information.
<code>-m xmlfile newdata</code>	Modifies a document or component, specified by the vpath <i>xmlfile</i> , by copying data from an ordinary XML file with vpath <i>newdata</i> .
<code>-p xmlfile</code>	Deletes (purges) any cached versions of an XML file specified by vpath <i>xmlfile</i> .
<code>-u infile outdir outname</code>	Writes a referenced version of an XML File with vpath <i>infile</i> to file <i>outname</i> in the directory with vpath <i>outdir</i> .
<code>-v</code>	Displays version information about this software.
<code>-z file catname</code>	Componentizes a file without turning it into a TeamXML file.

*optional\_argument* may be zero or more of the following:

-d <i>dst</i>	Creates a new component in an area with vpath <i>dst</i> . Ignored if <i>catname</i> does not specify a component category.
-catpath	The <i>catname</i> argument will be treated as a category vpath. <i>catname</i> with a leading path separator will always be treated as the category vpath (even without this option).
-i	Prints the resulting ID when creating a new component.
-r	Removes <i>newdata</i> after successfully modifying a file. Use -r only in conjunction with -m.

### Examples

- `iwxmlfile -c /default/main/WORKAREA/wa1/mydoc.xml RULES/doctype1.dfn`  
This command (a) assigns the file specified by vpath `/default/main/WORKAREA/wa1/mydoc.xml` to the category `RULES/doctype1.dfn`; and (b) componentizes this file according to the rules in the specified category.
- `iwxmlfile -a /default/main/WORKAREA/wa1/mydoc.xml /default/main/WORKAREA/wa1 ANNOTATED.mydoc.xml`  
This command creates an ordinary XML file (with vpath `/default/main/WORKAREA/wa1/ANNOTATED.mydoc.xml`) containing an annotated version of the content of an XML document with vpath `/default/main/WORKAREA/wa1/mydoc.xml`.

## iwxmlindex.ipl

`iwxmlindex.ipl` uses the OpenAPI Search and Filesys services to update or delete the indexes for ordinary files and files that are part of TeamXML. To ensure proper user authentication, you must execute `iwxmlindex.ipl` on the same machine as the OpenAPI server.

### Usage

`iwxmlindex.ipl request`

where *request* specifies the requested information. It may be one of the following. Request names are case-insensitive, and most have an abbreviation, shown in parentheses. Multiple vpaths are space separated.

IndexTeamXMLFiles <i>vpath</i> (abbreviation: itf)	Index all the TeamXML files specified by <i>vpath</i> . The vpath can be any of <i>area-vpath</i> , <i>categoryfile-vpath</i> , <i>componentfile-vpath</i> , and <i>documentfile-vpath</i> . Multiple vpaths are allowed.
IndexSimpleFiles <i>ruleset-vpath vpath</i> (abbreviation: isf)	Index the simple file specified by <i>vpath</i> using the index ruleset specified by <i>ruleset-vpath</i> . The vpath can be vpath of IWSimpleFile and its derivatives. If it is a vpath of a TeamXML file such as IXMLComponentFile, the request is treated as IndexTeamXMLFiles. Multiple vpaths are allowed.
IndexSimpleFileCollection <i>ruleset-vpath directory-path</i> (abbreviation: isfc)	Index all the simple files listed in the directory file specified by <i>directory-path</i> using the index ruleset specified by <i>ruleset-vpath</i> . The file specified by <i>directory-vpath</i> contains a list of simple file vpaths which are listed as one vpath per line.
RemoveTeamXMLFiles <i>vpath</i> (abbreviation: rtf)	Remove all the TeamXML files specified by <i>vpath</i> . The vpath can be any of <i>area-vpath</i> , <i>categoryfile-vpath</i> , <i>componentfile-vpath</i> , and <i>documentfile-vpath</i> . Multiple vpaths are allowed.
RemoveSimpleFiles <i>vpath</i> (abbreviation: rsf)	Remove the index for simple file specified by <i>vpath</i> . The vpath can be vpath of IWSimpleFile and its derivatives. If it is a vpath of a TeamXML file such as IXMLComponentFile, the request is treated as RemoveTeamXMLFiles. Multiple vpaths are allowed.
RemoveByQuery <i>directory-path</i> (abbreviation: rq)	Remove index files returned by running a query with <i>directory-path</i> . Only one query file is allowed.

Help [ <i>cmd</i> ] (abbreviation: -h)	Prints help information for <i>cmd</i> or general usage information if <i>cmd</i> is omitted.
Version (abbreviation: -v)	Print version information.

### **Examples**

- `iwxmindex.ipl itf /default/main/STAGING`  
This command updates all indexes for TeamXML repository files in the /default/main/STAGING area.
- `iwxmindex.ipl itf /default/main/WORKAREA/jdoe/COMPONENT/vendor.dfn /default/main/WORKAREA/jdoe/COMPONENT/product.dfn`  
This command updates all indexes for TeamXML repository files in the categories defined by /default/main/WORKAREA/jdoe/COMPONENT/vendor.dfn and /default/main/WORKAREA/jdoe/COMPONENT/product.dfn
- `iwxmindex.ipl rtf /store3/main/STAGING`  
This command removes all TeamXML files that exist in /store3/main/STAGING.

## iwxmlinfo

`iwxmlinfo` obtains information about TeamXML, such as known categories or components contained in a category. To use this CLT, you must specify a temporary directory in a workarea, as described in the “Notes.”

### Usage

Syntax is as follows:

```
iwxmlinfo [optional_argument] request
```

where *request* is exactly one of the following:

AllComponentCategories <i>area</i> (abbreviation: acc)	Lists all of the component categories in an area specified by <i>vpath</i> .
AllDocumentCategories <i>area</i> (abbreviation: adc)	Lists all of the document categories in an area specified by <i>vpath</i> .
CategoryMembers <i>area cat</i> (abbreviations: cm, members)	Lists the members of category <i>cat</i> in an area specified by <i>vpath</i> .
ComponentPath <i>area cat id</i> (abbreviation: cp)	Obtains the area-relative path of the component with identification name <i>id</i> in category <i>cat</i> in an area specified by <i>vpath</i> .
Components <i>vpath</i> (abbreviation: comp)	Lists the first-generation child components of the document or compound component with the specified <i>vpath</i> .
Containers <i>vpath</i> (abbreviation: cont)	Lists the documents and components that contain a first-generation child component specified by <i>vpath</i> .
FileInfo <i>vpath</i> (abbreviations: file, fi)	Obtains details about a file in TeamXML, such as its display name and mutation information.
Help [ <i>cmd...</i> ] (abbreviation: -h)	Displays help information. Optionally, you may request help information about a specific command or commands by typing <code>iwxmlinfo -h <i>cmd</i></code> , where <i>cmd</i> is one or more commands.



IsChild <i>parent vpath</i> (abbreviation: <i>isc</i> )	Determines if the file with vpath <i>vpath</i> is a component contained directly by the file with vpath <i>parent</i> .
IsDescendant <i>parent vpath</i> (abbreviation: <i>isd</i> )	Determines if the file with vpath <i>vpath</i> is a component contained by the file with vpath <i>parent</i> , either as a first-generation child or a more remote descendant.
Version (abbreviation: <i>-v</i> )	Displays version information about this software.

*optional\_argument* is zero or more of the following:

<i>-f fmt</i>	Specifies output format, where <i>fmt</i> can have one of two values: – <i>text</i> : Human-readable text (default) – <i>script</i> : Terse text, suitable for parsing
<i>-t vpath</i>	Specifies using a workarea directory <i>vpath</i> for temporary files.

## Notes

- These request names are not case sensitive; most have an abbreviation, shown in parentheses.
- Many requests require temporary files in a workarea (though not necessarily the workarea that is the target of the request). TeamXML will search, in the following order, for a directory in which to create these files:
  - a. The directory specified by the *-t* option, if present.
  - b. The directory specified by the IWCLT\_TMPDIR environment variable, if present.
  - c. The root directory of the target area, if it is a workarea.
- The script format (*-f script*) for FileInfo outputs eight fields, suitable for interpretation by a parser. See the following “Examples” section for details.



### Examples

- `iwxmllinfo AllComponentCategories /default/main/WORKAREA/wa1`

This command lists component categories in area `/default/main/WORKAREA/wa1`:

Name: One	Definition: CATEGORY/COMPONENTS/one.rules.dfn
Name: Three	Definition: CATEGORY/COMPONENTS/three.rules.dfn
Name: Models	Definition: CATEGORY/COMPONENTS/two.rules.dfn

- `iwxmllinfo -f script acc /default/main/WORKAREA/wa1`

This command uses the `acc` abbreviation instead of the longer argument `AllComponentCategories`. This command also lists component categories in area `/default/main/WORKAREA/wa1`, as in the previous example, but the output is more terse due to the `-f script` option:

```
One CATEGORY/COMPONENTS/one.rules.dfn
Three CATEGORY/COMPONENTS/three.rules.dfn
Models CATEGORY/COMPONENTS/two.rules.dfn
```

- `iwxmllinfo fileinfo /default/main/WORKAREA/wa1/mydoc.xml`

This command lists details about the specified file:

```
Kind:          DOCUMENT
Category:      CATEGORY/COMPONENTS/doctype1
Mutated:       Tue May 8 11:32:53 2001 by user1
# components:  4
```

- `iwxmllinfo -f script fi /default/main/WORKAREA/wa2`

This command lists eight information fields, suitable for parsing, for category definition files and XML files in the specified area:

- For the category definition file `doc_1.dfn`:  
"doc\_1" 0 <n/a> <n/a> <n/a> <n/a> <n/a> <n/a>
- For the XML file `doc_1`:  
"" 1 "doc\_1.dfn" 1 988922533 XYZCorp\mjones 2 <n/a>

The meaning of these eight fields is as follows:

1. Display name (or "" if display name is not defined)
2. The type of XML file. The possible values are:
  - 0: category definition
  - 1: document
  - 2: compound component
  - 3: leaf component
3. Possible values are:
  - Category name (for documents and components; category name is the area-relative path of the category definition file)
  - <n/a> (for category definition files; <n/a> is an abbreviation for “not applicable”)
4. Possible values are:
  - 0 (when a file is *not* a container file)
  - 1 (for documents and components; 1 indicates that a file is a container file)
  - <n/a> (for category definition files and info files; <n/a> is an abbreviation for “not applicable”)
5. Possible values are:
  - Mutation time in Portable Operating System Interface (POSIX) format (for documents and components)
  - <n/a> (for category definition files, info files, and components; <n/a> is an abbreviation for “not applicable”)
6. Possible values are:
  - Mutator; that is, the user responsible for the modification corresponding to the mutation time (for documents and components)
  - <n/a> (for category definition files, info files, and components; <n/a> is an abbreviation for “not applicable”)



7. Possible values are:

- Number of child components (for documents and components)
- 0 (for a leaf component)
- `<n/a>` (for category definition files and info files; `<n/a>` is an abbreviation for “not applicable”)

8. Possible values are:

- Number of containers for components
- `<n/a>` (for documents, category definition files, and info files; `<n/a>` is an abbreviation for “not applicable”)

## iwxmlsearch.ipl

`iwxmlsearch.ipl` uses the OpenAPI Search and Filesys services to search for information in the TeamXML repository and return a list of vpaths to files that match the query. A search can be a simple keyword search (`Keyword` option) or a complex search (`Query` option) that executes a query contained in an XML file that complies to a required DTD, `xml_search1.0.dtd`. If you use the `Query` option, the results will also contain fields listed in the `select` clause. To ensure proper user authentication, you must execute `iwxmlsearch.ipl` on the same machine as the OpenAPI server.

### Usage

`iwxmlsearch.ipl option request`

*option* is zero or one of the following:

<code>-m maxResult</code>	Specify the maximum number of results ( <i>maxResult</i> ) that should be returned from the search. If no number is specified, the default maximum number of 500 is used.
---------------------------	---

*request* specifies the requested information. It may be one of the following. Request names are case-insensitive, and most have an abbreviation, shown in parentheses.

Keyword <i>vpath</i> <i>keyword-string</i>	Find files containing <i>keyword-string</i> . The search results are limited to files at or below <i>vpath</i> . To search across TeamSite, use "/" as <i>vpath</i> .
Query <i>directory-path</i>	Find files based on a query with <i>directory-path</i> .
Help [ <i>cmd</i> ] (abbreviation: -h)	Prints help information for <i>cmd</i> or general usage information if <i>cmd</i> is omitted.
Version (abbreviation: -v)	Print version information.

*Examples*

- `iwxmlsearch.ipl Keyword / "dog food"`

This command searches the entire TeamXML repository for files that contain the phrase dog food.

- `iwxmlsearch.ipl Query /tmp/myquery.xml`

This command executes the query contained in the file `/tmp/myquery.xml`.

- `iwxmlsearch.ipl -m 100 keyword /default/main/WORKAREA/jdoe/TeamXML/component/vendor "Interwoven"`

This command searches the `vendor` category in the workarea for files that contain the word Interwoven and limits the results to 100 files.

## Appendix B

# DTDs

---

This appendix contains DTDs for TeamXML-specific (that is, non-content) files. The DTDs are for:

- document category definition files
- component category definition files
- indexing rule set
- search statements

The DTDs shown in this appendix reside in `iw-home/iwopenapi/dtd`.

Additional DTDs for sample content files are installed with TeamXML and reside in `teamxmlworkarea/ExampleDocuments/DTD`. Those DTDs are not shown in this appendix.

## Category Definition File DTDs

You must use TeamXML DTDs when creating category definition files. This section contains the following required DTDs:

- DTD for Document Category Definition Files
- DTD for Component Category Definition Files
- DTD for Indexing Rule Set

## DTD for Document Category Definition Files

Use the following DTD, `document_category1.0.dtd`, when creating document category definition files.

```
<?xml encoding="UTF-8"?>

<!--
    Copyright 2001 Interwoven Inc.
    All rights reserved.

    DTD for validating definition files for a document categories.
-->

<!ELEMENT document_category (display_name, root_elements?,
                             component_ruleset?,
                             external_validation_spec?, index_ruleset?,
                             category_description?)>

<!ELEMENT display_name EMPTY >
<!ATTLIST display_name name CDATA #REQUIRED >

<!ELEMENT root_elements (element+) >

<!ELEMENT element EMPTY >
<!ATTLIST element type NMTOKEN #REQUIRED >

<!ELEMENT component_ruleset (component+)>

<!ELEMENT component EMPTY >
<!ATTLIST component xpath CDATA #REQUIRED >
<!ATTLIST component category CDATA #REQUIRED >

<!ELEMENT external_validation_spec (schema_ref)>
<!ATTLIST external_validation_spec type CDATA #IMPLIED >

<!ELEMENT schema_ref (#PCDATA)>

<!ENTITY % index_ruleset SYSTEM "xml_index_ruleset1.0.dtd">
%index_ruleset;

<!ELEMENT category_description ANY>
```



## DTD for Component Category Definition Files

Use the following DTD, `component_category1.0.dtd`, when creating component category definition files.

```
<?xml encoding="UTF-8"?>

<!--
    Copyright 2001 Interwoven Inc.
    All rights reserved.

    DTD for validating definition files for a component categories.
-->

<!ELEMENT component_category (display_name, root_elements?, display_name_ruleset
?,
                                component_directory, component_ruleset?,
                                external_validation_spec?, index_ruleset?,
                                category_description?, prolog?)>

<!ENTITY % document_category SYSTEM "document_category1.0.dtd">
%document_category;

<!ELEMENT display_name_ruleset (display_name_rule+) >

<!ELEMENT display_name_rule EMPTY >
<!ATTLIST display_name_rule xpath CDATA #REQUIRED >

<!ELEMENT component_directory EMPTY >
<!ATTLIST component_directory area_rel_path CDATA #REQUIRED >

<!ELEMENT prolog (prolog_data | prolog_file)>

<!ELEMENT prolog_data (#PCDATA)>

<!ELEMENT prolog_file EMPTY>
<!ATTLIST prolog_file area_rel_path CDATA #REQUIRED >
```

## DTD for Indexing Rule Set

The DTDs for document and component category definition files refer to `xml_index_ruleset1.0.dtd` as an external parameter entity. Use this DTD when creating elements that specify an indexing rule set inside your category definition file or when creating a standalone file for an indexing rule set (to use when indexing non-TeamXML files).

```
<?xml encoding="UTF-8"?>

<!--
    Copyright 2001 Interwoven Inc.
    All rights reserved.

    DTD for validating index rulesets for the OpenAPI XML Search Service.
-->

<!ELEMENT index_ruleset (file_attr, extended_attr?, content?)>

<!ELEMENT file_attr (attr_name*)>
<!ATTLIST file_attr
    index_all (true | false) "false">

<!ELEMENT attr_name EMPTY>
<!ATTLIST attr_name
    name (area_path | area_rel_path | index_time | owner | creator |
        creation_time | last_content_mod_time | last_attr_mod_time |
        ruleset_name) #REQUIRED>

<!ELEMENT extended_attr (all_attrs | partial_attrs)>

<!ELEMENT all_attrs (key*)>

<!ELEMENT partial_attrs (key+)>

<!ELEMENT key (text_type | numeric_type | date_type | xml_type)>
<!ATTLIST key
    name CDATA #REQUIRED>

<!-- Preprocessing is only applicable for XML content (is_xml = "true") -->

<!ELEMENT content (transform?, field*)>
<!ATTLIST content
    is_xml (true | false) "true">
```

```
<!ELEMENT transform (automatic | xslt_stylesheet)>

<!ELEMENT field (text_type | numeric_type | date_type | xml_type)>
<!ATTLIST field
    name CDATA #REQUIRED
    xpath CDATA #REQUIRED>

<!ELEMENT xslt_stylesheet (#PCDATA)>

<!ELEMENT automatic EMPTY>

<!ELEMENT text_type EMPTY>

<!ELEMENT numeric_type EMPTY>

<!ELEMENT date_type EMPTY>
<!ATTLIST date_type
    format CDATA #REQUIRED>

<!ELEMENT xml_type EMPTY>
<!ATTLIST xml_type
    validate (true | false) "false">
```



## Search DTD

Use the following DTD, `xml_search1.0.dtd`, when creating search statement files.

```
<?xml encoding="UTF-8"?>

<!--
    Copyright 2001 Interwoven Inc.
    All rights reserved.
    DTD for validating queries for the OpenAPI XML Search Service.
-->

<!ELEMENT search (select?, where)>

<!ELEMENT select (field+)>

<!ELEMENT field EMPTY>
<!ATTLIST field
    facet (file_attr | extended_attr | metadata | content) #REQUIRED
    xpath CDATA #REQUIRED
    keyname CDATA #IMPLIED>

<!ENTITY % searchitem "and | or | not | predicate" >

<!ELEMENT where (%searchitem;)>

<!ELEMENT and ( %searchitem;)* >

<!ELEMENT or (%searchitem;)* >

<!ELEMENT not (%searchitem;)* >

<!ELEMENT predicate EMPTY>
<!ATTLIST predicate
    facet (file_attr | extended_attr | metadata | content) #REQUIRED
    xpath CDATA #REQUIRED
    keyname CDATA #IMPLIED
    operator (GT | LT | GE | LE | EQ | NOTEQ | CONTAINS | EXIST) #REQUIRED
    value CDATA #REQUIRED>
```

# Glossary

---

## *Annotation*

Processing instructions added by the TeamSite server to TeamXML files so that it can identify the file's subcomponents. Normally these annotations are not shown; however, when a user is editing XML-format data, the annotations display in the form of XML processing instructions. The annotation:

- Identifies a component by placing annotation information at the beginning of the component.
- Describes the location of the component.
- Provides access control information.

## *Category*

A set of related documents or components in TeamXML. The members of a category all share a common category definition. A given document or component is a member of exactly one category. There are two types of categories: document categories and component categories.

If a category is populated with one or more members, the category definition file itself, along with many of the properties described within it, becomes immutable. The mutable properties are modified using a programming interface (OpenAPI) or the `iwxmlcat` CLT, and the category definition file is updated automatically. The category definition file cannot be deleted until after the category itself has been deleted.

## *Category Definition File*

The properties of a category are defined in a special file known as the category definition file, which is provided to TeamXML by the system administrator before any documents or components are added to the category. Information contained in a category definition file describes details of a category, such as where members of the category are located, how they are indexed, their componentization ruleset, and how to determine their display name.

### *Component*

A single XML object and its children that is treated as a unit by TeamXML. Unlike documents, components can be located inside another object. Components may contain other components; a nested component is called a subcomponent. A given component can be contained by any number of documents and/or components. Because components are contained by reference and are rendered dynamically at read time, any change made to a component will be reflected immediately in all of the documents and components that contain it. However, modifying a component does not cause any containing documents or components to be considered modified for versioning purposes. Components are represented with mutated icons until the changes are submitted.

A component is normally made up of a single root XML element, which may in turn contain any number of child elements. Because XML-formatted data is tree-like in structure, a component can be considered to form a distinct subtree of the document or compound component that contains it. Even though every component is an XML element, it is not necessarily true that every element in a stream of XML-formatted data is a component; the distinctions between elements that are components and those that are not components are defined by a componentization ruleset. See also *document*.

### *Component Category*

A category containing components. In general, a member of a component category can always be a subcomponent in another document or component. See also *category*.

### *Componentization Rule Set*

The componentization rule set for a document or compound component defines how to create the next level of subcomponents from this document or component. The rule set specifies only the next level to ensure that the rules for creating subcomponents from a compound component are consistent regardless of the source of the component (that is, if the component is derived from different categories of documents).

The rule set specification is a collection of items, each of which specifies an XPath navigation to an element in the document below the root element and the corresponding category name of the component that should be created using this element as the root. There is no guarantee that a particular document instance will correspond to the rule set and generate components unless the document is validated and the elements are mandatory; there is no requirement that components must be generated.

There is no relationship between a componentization ruleset for a component and a validation specification (such as a DTD or schema) that may be used to validate that component. You may apply a single rule set to components having many different DTDs or schemas, or you may apply many different componentization rule sets to components with a single given DTD or schema.

### *Componentize*

The process of converting an ordinary XML file to a document or component by associating it with a particular category to create components. This makes components available for reuse in other documents. Subcomponents are identified and categorized according to the componentization rule set specified in the category definition. Each exponent is extracted from the original document, stored in a separate file, and componentized recursively. References to the subcomponents are associated with the containing file so that they can be reincorporated later when the containing file is rendered. The resulting document has the same area-relative path as the original ordinary file.

The generated components are stored with an arbitrary, unique numeric name rooted in directories specified by their category definitions. Subfolders below the category directory are created to aid server performance and are not intended for end user browsing. See also *uncomponentize*.

### *Composition*

The process of creating a new document or component from existing components.

### *Display Name*

An name chosen by a system administrator to identify TeamXML files in user interfaces. Because component files have unique numeric names assigned by TeamXML, a particular element or attribute within the component can be chosen to be a user-readable way to identify the components. Usually you will want display names to be unique, although this is not a requirement. The display name is set in the category definition file for either a document or a component. The display name for a document is its area-relative path. The display name for a category definition file is specified explicitly in the category definition itself.

### *Document*

A TeamXML document is an XML file that has been componentized. The main distinction between a document and an ordinary file is that portions of the document, known as components, may in fact be stored in other, separate files and are included in the document when it is read. A document represents a complete, standalone unit while other components typically represent portions of a

document. A document cannot be contained by another element. A document can contain any number of components.

Components are included in a document as references. When a document is read, those components are automatically reincorporated (“rendered”) into the resulting byte stream; consequently, a casual observer would not know that the components were in fact stored separately from the containing document.

Unlike components, a document is allowed to contain non-element nodes such as processing instructions, comments, and document types outside of its root element. These are sometimes referred to as “front matter” and “back matter,” depending on whether they occur before or after the element.

#### *Document Category*

A category containing documents. A document can be a member of one document category.

#### *Mutated*

When one or more of the components of a document or component have been modified, the document or component has been mutated. For example, if document D contains component C and only C is changed, D is considered mutated. The term *mutated* does not indicate that a document or component has been directly modified; it only indicates that components contained within it have been modified.

TeamSite will calculate a “last-mutation” time for every document and compound component by choosing the most recent of the various last-mutation times for any components. This last-mutation time will not be versioned by TeamSite, so updates to it will not cause the containing documents and components to appear modified. When viewed through the file system, the last-modification time that is reported for a document or component will in fact be chosen from the most recent of its last-modification and last-mutation times in order to keep file caching systems working properly.

#### *Non-Componentized*

A file that contains XML-formatted data that has not been componentized for TeamXML and thus is only recognized by TeamXML as an ordinary XML file. May also be referred to as a *simple file*, *ordinary file*, or *flat file*.



### *Render*

The process of creating a textual representation of a document or component that includes the contents of all of its subcomponents. This operation might be performed, for example, when you access a document. Rendered documents are not specifically formatted or processed in any way; they return the contents of their components as-is. A rendered, annotated file contains in-line copies of all of its subcomponents, just as if it were read via the file system; additionally, each of those subcomponents will contain an annotation describing the location in the TeamXML repository of that subcomponent.

### *Specialize*

The process of converting an ordinary XML file into a TeamXML file. Specializing can optionally break a file into components (see “Componentize” on page 135). However, it is sometimes appropriate to specialize a file without breaking the file into individual components. Therefore, componentizing is considered to be a subset of specialization. For example, category definition files are typically specialized, not componentized, because they are never intended to be broken into individual components. All files that have been specialized appear with the TeamXML icon in the TeamSite GUI, even if they have not been componentized.

### *Subcomponent*

A component that is contained within another component or within a document.

### *TeamSite Area*

Any TeamSite workarea, edition, or staging area.

### *TeamXML Repository*

The collection of TeamXML files (documents, components, and category definitions) that reside in a particular area of a TeamSite server.

### *Uncomponentize*

The process of converting a document or component back into an ordinary XML file (essentially removing it from the TeamXML repository). The file will no longer have any of the special properties that distinguish documents or components. See also *componentize*.

### *Unrendered*

A document or component whose internal references to components have not been replaced with the actual contents of those components. An unrendered, annotated file does not contain the contents of

any of its subcomponents. Instead, the element corresponding to each subcomponent will be replaced with a reference annotation describing the component that would be present if the file were rendered normally.

#### *Validation Specification*

A set of rules used to determine if a component is valid. The format of these rules depends on the type of validation that is performed. For example, if validation is performed using a DTD, the validation specification is an external DTD; if validation is performed using an XML schema, the validation specification is a schema definition.

# Index

---

## Symbols

@do-not-componentize 74

## B

backing store  
disk space 31

## C

categories  
creating component 66  
creating document 65  
defined 19  
manipulating 114, 116  
obtaining information 120  
overview 15  
purpose 24  
category definition files 15, 63, 80  
component example 66  
componentization rule set 73  
creating 64  
display name 72, 76  
document example 65  
DTDs 127  
editing 25  
examples 85  
indexing rule set 76  
parts of 71  
prolog section 81  
root element 71  
root element rule set 72

specialize 16, 66  
user description 80  
validate 79  
XPath conventions 83  
category\_description 80  
client hardware 33  
client software 34  
command-line tools 113  
iwxmllcat 114  
iwxmllfile 116  
iwxmllindex 118  
iwxmllinfo 120  
iwxmllsearch 125  
component\_category 71  
component\_directory 80  
component\_ruleset 73  
componentization 16  
advantages 20  
example 16  
example rules 99  
excluding from 74  
componentization rule set  
XPath 73  
componentization rules  
applying 74  
changing 74  
components 81  
best practices 68  
category definition file  
DTD 129  
characteristics of 67

create reusable 69  
creating 57  
defined 18  
display name 76  
entity references 74  
example 101  
names of 67  
obtaining information 120  
overview 15  
reuse 20  
usage 24  
configuration settings 58  
configuring  
disk space 31  
content 77  
tag for reuse 69  
convert XML file 16, 17  
CPU requirements 29  
CPUs 30  
creating 73

## D

delete indexes 118  
directory structure  
creating 55  
example 85  
explained 56  
disk partitions 31  
disk space  
requirements 29

- display name 72, 76
- display\_name 72
  - example 98
- display\_name\_ruleset 76
  - example 98
- document\_category 71
- documents

- advantages 23
  - category definition file
    - DTD 128
  - characteristics of 65
  - defined 18
  - manipulating 116
  - rendering 70
  - retag content 68
- DTDs 71, 127
  - category definition files 127, 129
  - indexing rule set 130
  - search 132

**E**

- entity references 74
- exception rule 74
- extended attributes 77
- extended\_attr 77
- external\_validation\_spec 79

**F**

- features of TeamXML 21
- file attributes
  - indexing 77
- file\_attr 77
  - example 99
- files
  - creating TeamXML 16
  - default locations 36

**H**

- hardware
  - client 33
  - CPUs 30
  - requirements 29
  - TeamSite server 29

**I**

- index\_ruleset 76
- indexes
  - update 118
- indexing 54, 76, 99
  - content 77
  - example 78
  - extended attributes 77
  - file attributes 77
  - for searches 100, 103
  - rule set DTD 130
  - transform 79
- inode requirements 31
- install
  - adapters 37
  - final tasks 52
  - integrated XML editors 37, 53
  - preliminary tasks 37
  - sample content 59
  - schema editor 37
  - search engine 35
  - task overview 38
  - TeamXML system files 35
  - Tibco XML Authority 37, 52
  - UNIX systems 47
  - Windows systems 39
- integrated XML editors
  - install 37, 53
- integration with TeamSite 22
- iw\_component 70
- iw\_reference 70

- iwxmlcat 114
- iwxmlfile 116
- iwxmlindex 118
- iwxmlinfo 120
- iwxmlsearch 125

**M**

- manipulate
  - categories 114, 116
  - documents 116
- memory requirements 31
- mutation
  - defined 26

**N**

- network connections 36

**P**

- path name 80
- processing instructions
  - iw\_component 70
  - iw\_reference 70
- prolog 81
  - contents 82
  - OpenAPI support 82
  - purpose 81

**Q**

- queries
  - extended attributes 111
  - extended attributes
    - example 109
  - find by date example 108
  - find graphics example 107
  - search examples 105
  - workarea example 106

## **R**

- RAID 0+1 32
- render document 70
- requirements
  - backing store 31
  - CPU 29
  - disk space 29, 32
  - hardware 29
  - inode 31
  - memory 31
- root element rule set 72
- root\_elements 72
  - example 98
- Rules subdirectory
  - populating 57

## **S**

- sample content
  - install 59
- schema editor 52
  - install 37
- SCSI controllers and drives 32
- search 100, 125
  - command line 103
  - DTD 132
  - examples 105
  - guidelines 104
  - integrated XML editor 103
  - overview 25, 103
- search engine
  - install 32
  - network connection 36
  - verify startup 58
- server
  - hardware requirements 29
  - optimize performance 60
  - software 32

## **software**

- client 34
- default file locations 36
  - TeamSite server 32
- software components 28
- specialization 16
- startup
  - verify search engine 58
- subcomponents 17, 73
  - defined 19

## **T**

- TeamXML
  - advantages 20
  - features 21
  - software 28
  - TeamSite integration 22
- TeamXML directory 55
- Tibco XML Authority 37
  - install 52
- transformations
  - for indexing 79

## **U**

- uninstall 60
- update indexes 118

## **V**

- validation
  - XML files 79

## **X**

- XPath 73
  - conventions 83
  - indexing 79

