



Tridion R5

# Content Delivery Reference Guide

# Content Delivery Reference Guide

Revision CD\_RG\_5.1SP2/5.1SP3/5.1SP4

© 2004-2005 Tridion Development Lab B.V.

**NOTICE:** The accompanying software package is confidential and proprietary to Tridion Development Lab B.V. or its respective licensors. No use or disclosure is permitted other than as set forth by written license with the authorized distributors of Tridion Development Lab BV.

## Trademarks

Tridion and Tridion R5 are trademarks of Tridion Development Lab B.V. or its respective licensors. All other company or product names used herein are trademarks of its respective owners.

## Suggestions

Your suggestions and comments about Tridion R5 functionality, documentation and course material are highly valued and may be used to further enhance our offerings available to you. We will be glad to receive your suggestions at:

Tridion Development Lab B.V.

Product Management

P. O. Box 22709

1100 DE Amsterdam

The Netherlands

fax: +31 (0)20 20 10 501

Email: [PMsuggestions@tridion.com](mailto:PMsuggestions@tridion.com)

## Additional Licenses

Please contact your Tridion sales representative to order additional licenses of Tridion R5 software. [www.tridion.com](http://www.tridion.com) offers you a complete overview of Tridion's sales offices and further contact details.





# Table of contents

Chapter 1 Content Delivery .....	1
Chapter 2 Content Delivery product licenses .....	3
2.1 Installing a license file .....	4
2.1.a Transport Service license file .....	4
2.1.b Content Delivery product license files .....	5

## Part I—Content Distributor

---

Chapter 3 Content Distributor .....	9
Chapter 4 Content Manager Publication Targets .....	11
4.1 Publishing content .....	12
4.2 Publishing protocols .....	13
4.2.a Local File System, FTP and SFTP protocols .....	13
4.2.b HTTP(S) protocol .....	15
4.2.c Creating a Protocol Schema .....	15
4.3 Configuring Publication Targets .....	17
4.3.a Creating a Target Type .....	17
4.3.b Creating a Publication Target .....	19
4.4 Publication Target Code Page values .....	21
4.4.a Overriding mappings for code page values .....	22
Chapter 5 Configuring ASP.NET support for the Content Distributor .....	25
5.1 Configuring assemblies .....	25
5.1.a Configuring assemblies for a web application .....	26
5.1.b Configuring assemblies for a standalone application .....	26
5.2 Configuring WAI profiling and tracking .....	27

## Content Delivery Reference Guide

5.3 Publishing to ASP.NET .....	27
Chapter 6 .....	Senders and Receivers 29
6.1 Senders .....	30
6.2 Receivers .....	30
6.3 Local Copy, FTP and SFTP Senders and Receivers .....	31
6.4 HTTP(S) Senders and Receivers .....	33
6.4.a Troubleshooting transport over HTTP(S) to IIS .....	33
6.4.b Importing a certificate for HTTPS uploads .....	36
6.4.c HTTPS with IIS .....	36
6.4.d HTTPS with a Java Web/Application Server .....	37
6.5 Developing custom Senders and Receivers .....	39
Chapter 7 Transport Service configuration .....	41
7.1 Configuring the Transport Service .....	42
7.1.a WorkFolder .....	43
7.1.b Logging .....	43
7.1.c Senders .....	43
7.1.d Poller .....	44
Chapter 8 Deployer configuration .....	47
8.1 Configuring the Deployer .....	48
8.2 WorkFolder .....	50
8.3 Logging .....	50
8.4 Processors .....	51
8.4.a Developing custom processors .....	51
8.5 Module .....	52
8.5.a Developing custom Deployer modules .....	53
8.5.b Modifying the behavior of default modules .....	55
8.5.c Testing custom modules .....	56
8.6 Transformer .....	57
8.7 ReceiverBindings .....	58
8.8 Receivers .....	58
Chapter 9 Broker configuration .....	61
9.1 Configuring the Content Broker .....	62
9.2 Logging .....	63
9.3 Object caching .....	64
9.3.a ObjectCache .....	64
9.3.b Policy .....	65
9.3.c Features .....	65

## Content Delivery Reference Guide

9.3.d CacheBindings .....	66
9.3.e RemoteSynchronization .....	67
9.3.f Cache Channel Service .....	67
9.4 Storage configuration .....	68
9.5 Bindings .....	70
9.6 Publications .....	71
Chapter 10 Using metadata .....	73
10.1 Mandatory metadata .....	74
10.1.a Mandatory link metadata .....	74
10.1.b Mandatory Component Metadata .....	74
10.1.c Mandatory Page Metadata .....	75
10.1.d Mandatory Component Presentation Metadata .....	76
10.1.e Mandatory binary metadata .....	77
10.2 Custom metadata .....	77
10.2.a Custom metadata examples .....	78
10.2.b Custom metadata fields with multiple values .....	79
10.3 Metadata storage .....	79
10.3.a File system storage .....	79
10.3.b Configuring the storage options .....	80
10.3.c Database storage .....	80
10.3.d Storing custom metadata in SQL databases .....	81
10.4 Metadata examples .....	82
10.4.a ASP example .....	82
10.4.b JSP example .....	83
10.5 Filters .....	84
10.5.a Accessing a single Component Presentation .....	84
10.5.b Search filter .....	85
10.5.c Generating Presentation Filters .....	86
10.5.d Result modifiers .....	86
10.6 Querying custom metadata .....	87
10.6.a Queries against SQL databases .....	87
Chapter 11 Tamino .....	89
11.1 Configuration .....	89
11.2 Using Tamino .....	90
11.2.a Schemas .....	90
11.2.b Component Templates .....	91
11.2.c Components .....	91
11.2.d Updating a Component in Tamino .....	92
11.2.e Unpublishing from Tamino .....	92

## Content Delivery Reference Guide

11.2.f Dynamic Content .....	92
11.2.g Schema mapping .....	92
11.2.h XSLT transformation of W3C schemas .....	93
11.2.i ReferenceCounting .....	93
11.3 Querying Tamino .....	93
11.3.a Generating links from an XSLT using the Linking API .....	95
11.3.b Component link resolution with XSLT .....	96
11.4 TaminoFilter .....	97
11.4.a Using TaminoFilter to query Tamino .....	97
11.4.b Merging Queries .....	97

## Part II—Linking

---

Chapter 11 Linking .....	101
11.1 Link validation and resolution .....	102
11.2 Page links .....	103
11.3 Binary links .....	104
11.4 Component links .....	105
11.4.a Components that link to the same Component .....	105
11.4.b Components that link to a different Component .....	106
11.4.c Creating Component links in templates .....	106
11.4.d Creating Component Link objects in a Page Template ....	107
11.4.e Resolving Component links .....	107
Chapter 12 Linking configuration .....	111
12.1 Linking configuration file .....	112
12.1.a Logging .....	112
12.1.b Publications .....	113
12.2 Linking and Content Broker configuration .....	113
12.2.a Linking bindings .....	114

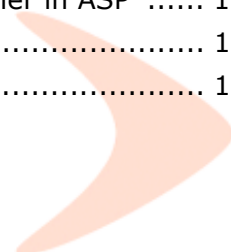
## Part III—Web and Application Server Integration

---

Chapter 13 Web and Application Server Integration .....	117
13.1 Web and Application Server Integration (WAI) .....	118
13.1.a Profiling and personalization .....	118
13.1.b Dynamic content assembly .....	118

## Content Delivery Reference Guide

Chapter 14 Profiling and personalization .....	119
14.1 Setting up Profiling .....	120
14.2 Tracking Keys .....	121
14.2.a Creating Categories and Keywords .....	122
14.2.b Using Categories and Keywords as Schema list field values ..	123
14.2.c Tracking Categories in Component Templates .....	125
14.3 Tracking .....	126
14.4 Configuring the WAI for profiling .....	126
14.4.a Global logging settings .....	127
14.4.b Presentations .....	128
14.4.c Timeframes .....	131
14.4.d Excluding Pages, Components or Paths from being tracked ..	131
14.5 Personalizing content .....	133
14.5.a Creating Target Groups .....	133
14.5.b Associating Target Groups with Component Presentations ....	136
Chapter 15 Dynamic content assembly .....	139
15.1 Using Dynamic Component Presentations .....	140
15.2 Creating Component Templates for dynamic content assembly ....	141
15.2.a Output format .....	142
15.2.b Publishing Dynamic Component Presentations .....	143
15.2.c Publishing Pages that include Dynamic Component Presenta-	143
tions .....	
15.3 Configuring the Content Broker for Dynamic Component assembly	144
15.4 Assembling Dynamic Component Presentations .....	144
15.4.a ComponentPresentationAssembler .....	144
15.4.b ComponentPresentation.getContent() .....	145
15.5 Assembling Dynamic JSP Component Presentations .....	145
15.5.a Using the JSPPProcessor .....	146
15.5.b Using the ComponentPresentationAssembler in JSP .....	146
15.5.c Using DynamicComponentLink .....	147
15.6 Assembling Dynamic ASP Component Presentations .....	149
15.6.a Executing VBScript or JScript Dynamic Component Presenta-	149
tions .....	
15.6.b Using the ComponentPresentationAssembler in ASP .....	150
15.6.c Using DynamicComponentLink .....	150
15.7 Dynamic Assembly XML .....	152



## Appendices

Appendix A Example: An SMTP Custom Sender.....	157
Appendix B Example: Custom POP3 Receiver.....	159
Appendix C Sample Transport Service configuration file .....	163
Appendix D Sample Deployer configuration file .....	164
Appendix E Sample Broker configuration file.....	166
Appendix F Broker bindings for MS SQL, Oracle, Tamino and DB2 databases...	171
Appendix G WAI-Specific bindings.....	173
Appendix H System classpath.....	174
Appendix I Third-party libraries .....	175
Appendix J Configuring the Java Virtual Machine for COM services.....	177







# Preface

## About this guide

This guide is written for template designers and system administrators that configure and use the Tridion Content Distributor, Tridion Linking, and Tridion Web and Application Server Integration (WAI). This guide explains how to configure and use the Content Distributor, Linking, and WAI. The following software and hardware knowledge is assumed:

- JSP or ASP
- Web and Application Servers
- Tridion R5
- XML

## How to use this guide

This guide briefly introduces the different Content Delivery Products and is then divided into the following parts:

### *Part I Content Distributor*

This part describes Content Distributor functionality and describes how to configure and use Publication Targets, the Transport Service, Content Deployer and the Content Broker. In addition, this part describes how to configure Sender-Receiver pairs, and how to use filters and queries.

### *Part II Linking*

This part describes Tridion Linking functionality and describes how to configure Linking for use with the Content Distributor.

### *Part III Web and Application Server Integration*

This part describes the WAI and how to configure the WAI to track and personalize content, and how to create and assemble Dynamic content.

## Related documents

This guide is one document from the documentation set for Tridion products. The full documentation set consists of the following documents:

- The *Tridion 5.1 SP3 Installation Guide* explains how to install Tridion 5.1 SP3 products included in this release. It is intended for system administrators.
- The *Tridion 5.1 SP3 Upgrade Guide* explains how to upgrade and configure Tridion 5.1 SP3 products included in this release. It is intended for system administrators.
- The *Administration Guide* explains how to perform general administrative tasks for the Content Manager. It is intended for system administrators.
- The *Content Manager User Manual* explains which tasks an end user can perform using this product, and what it means to perform those tasks.
- The *Content Manager Publication Management Guide* explains how to manage Publications within the Content Manager.
- The *Content Manager Programmer's Reference Guide* explains how to configure the product in detail and how to extend its functionality through scripting and programming. It is intended for software developers.
- The *WebDAV User Manual* explains which tasks an end user can perform using the Tridion WebDAV Connector.
- The *Business Connector Guide* explains how to use the Tridion Business Connector.
- The *Tridion Release Notes* document lists what has changed from previous versions of the Content Manager and the Content Delivery products. It also describes any open and closed issues.

## Conventions

This section describes the typographical conventions used in this document.

- Code is formatted with the Courier New font, `Set Properties` for example
- Filenames, directory structures and URLs are formatted using Courier Bold font, **`www.tridion.com`** for example
- Book names are italicized, *R5 Administration Guide*, for example
- Cross-references, within the Guide, are italicized and contain the name of the referenced section, table, or figure

# Content Delivery Reference Guide

## Version history

This version history list outlines the changes to the Content Delivery Reference Guide since it was last released.

Product Release number	Document number	Changes
5.1 SP2	1.0	<p>This guide comprises the former Content Distributor Guide, Linking Guide, and Web and Application Server Integration Guide.</p> <p>Most chapters have been rewritten for this release.</p> <p>This book now describes Publication Target configuration as it relates to publishing to the Content Distributor.</p>
5.1 SP3	1.1	Minor changes only. Rerelease.
5.1 SP4	1.2	<p>Fixed faulty code samples in "<i>ComponentPresentation.getContent()</i>" on page 145</p> <p>More information about HTTP(s) transport (including troubleshooting) added in "<i>HTTP(S) Senders and Receivers</i>" on page 33</p> <p>Added configuration info for the LocalCopySender in "<i>Local Copy, FTP and SFTP Senders and Receivers</i>" on page 31</p> <p>Explanation of a new feature, configuring the COM-called JVM from the registry, explained in "<i>Configuring the Java Virtual Machine for COM services</i>" on page 177</p> <p>Minor changes such as typos</p>



# Content Delivery Reference Guide



# Chapter 1 Content Delivery

Tridion Content Delivery products enable you to publish and distribute content to Web and Application servers. This guide describes the following Tridion Content Delivery products:

- Transport Service — The Transport Service is installed as part of a Content Manager installation. The Transport Service contains Senders that send published content to the Content Distributor.
- Content Distributor — The Content Distributor is a Java-based engine designed to deliver Content Manager content to Web and Application Servers. The Content Distributor contains two modules:
  - ▶ Content Deployer — The Content Deployer receives and analyzes content from the Transport Service.
  - ▶ Content Broker — The Content Broker stores and accesses content on a storage medium such as a file system or a database.
- Linking — Linking resolves and validates Component, Binary and Page links on a published Web site.
- Web and Application Server Integration (WAI) — The WAI allows you to track, profile, and personalize content shown to your Web site visitors, and to dynamically assemble content.

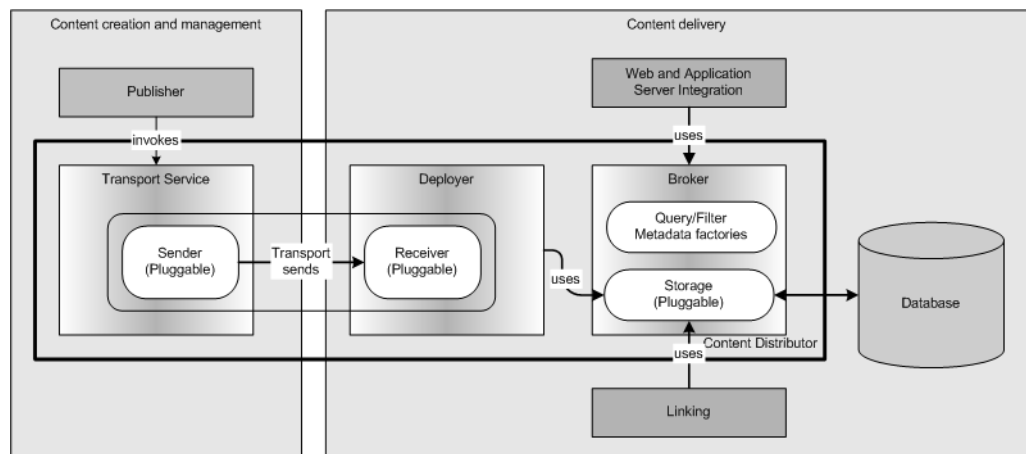


Figure 1-1 Content Delivery architecture





# Chapter 2 Content Delivery product licenses

Tridion products are licensed. Your license agreement with Tridion determines what products and functionality you can use and access. Tridion Customer Support provides you with the license files you require.

For each product, the license may be in one or more files. The following Content Delivery products and features are licensed:

- Transport Service — enables you to publish. By default, includes local file publishing.
  - HTTP(S) — enables you to publish to HTTP(S)
  - FTP — enables you to publish to FTP
  - SFTP — enables you to publish to SFTP
- Content Deployer — enables you to deploy content remotely or to a database
- Content Broker — enables you to store content to a database
- Linking — enables you to use Linking to resolve and validate links on published content
- WAI — enables you to use WAI functionality to profile and track visitor behavior, and to dynamically assemble content

Content Delivery licenses include the following restrictions:

- CPU limit — delimits the number of logical processors in the system
- Node-lock — the computer for which the license applies using the host name of the computer. You must obtain a different license file for each computer on which products are installed. License files are machine specific.
- Time — (Optional) a time limit on your license

### 2.1 Installing a license file

You must obtain the following license files from Tridion Customer support:

- a Transport Service license file
- Content Delivery license(s)

**Note** If features are not licensed, attempts to access or exceed limits of licensed products and features results in the inability to access the requested feature and error or information messages.

#### 2.1.a Transport Service license file

You require a Transport Service license file on your Content Manager server in order to publish. By default, this license file includes local file publishing only. If you wish to publish to HTTP(S), SFTP, or FTP, ensure that you are licensed to publish to the desired protocol.

You must install this license file to publish.

**Prerequisites** To obtain the Transport Service license file, contact Tridion Customer Support.

#### To install the Transport Service license file:

- 1 Do one of the following:
  - If the name of the license file is `cd_licenses.xml`, copy the license file `cd_licenses.xml` to `c:\Program Files\Tridion\config` or the location of your Content Manager installation. (This is the same location as the `cd_transport_conf.xml` file.)
  - If the name of the license file is not `cd_licenses.xml`, you must change the name in the `cd_transport_conf.xml` file after the closing `Senders` element and before the last close tag. For example:

```
<License Location="c:\Tridion\cd_transport_license.xml"/>
</TransportService>
```

- 2 Restart your Transport Service for licensing to take effect.

**Note** If a `LicenseNotFoundException` occurs, ensure that the Transport Service configuration file contains a valid license location.



## 2.1.b Content Delivery product license files

The following Content Delivery products may be licensed:

- Deployer
- Broker
- Linking
- WAI

### To install the Content Distributor license file:

- 1 Do one of the following:
  - If the name of the license file is `cd_licenses.xml`, copy the license file `cd_licenses.xml` to `c:\Program Files\Tridion\config` or the location of your Content Distributor product installation. This is the same location as the related configuration files:
    - ▶ `cd_deploy_conf.xml`
    - ▶ `cd_broker_conf.xml`
    - ▶ `cd_linking_conf.xml`
    - ▶ `cd_wai_conf.xml`
  - You must change the name and/or directory in the related configuration file before the last closing tag in that file if any of the following conditions apply:
    - ▶ The name of the license file is not `cd_licenses.xml`,
    - ▶ You wish to install the license file in a different location
    - ▶ You have separate license files for different products
    - ...
- 2 Restart your system for licensing to take effect.

**Note** If a `LicenseNotFoundException` occurs, ensure that the relevant configuration files contain a valid license location.



## Chapter 2 Content Delivery product licenses



# **Part I**

---

# **Content Distributor**



# Chapter 3 Content Distributor

The Content Distributor is a Java-based engine designed to deliver Tridion Content Manager content to Web servers and Application servers. To publish content from the Content Manager to the Content Distributor, the following process occurs:

- 1 Content is published from the Content Manager to one or more Publication Targets. The Content Manager publisher resolves and renders content using Component Templates and Page Templates.
- 2 The Content Manager publisher creates a Transport Package. The Transport Package contains the rendered content, metadata, and deployment information.
- 3 The Transport Service uses Senders to send the Transport Package to Content Distributor Receivers using the Publication Target protocol to which content was published. Tridion Sender-Receiver pairs support the following protocols:
  - Local Copy
  - (S)FTP
  - HTTP(S)
- 4 In the Content Distributor, the Content Deployer unpackages the Transport Package, analyzes content, and reads the instructions about the publish action. By default, the Content Deployer sends the content to the Content Broker.
- 5 The Content Broker determines where the Pages and metadata associated with the published content are to be stored and stores the content in the content storage medium (a file system, a SQL database, such as MS-SQL, or Oracle, or an XML database, such as Tamino.)

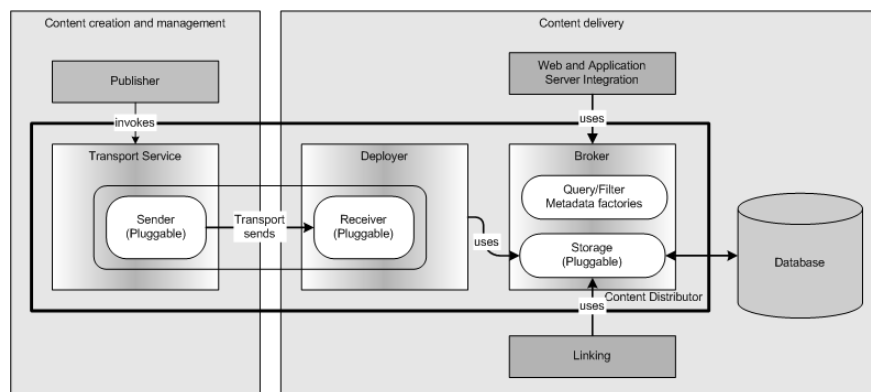


Figure 3-1 Content Distributor architecture

This part describes how to configure and use the following Content Distributor settings and functionality:

- Content Manager Publication Targets — The Publishing Target include the following publishing parameters:
    - ▶ The target language and code page settings of published content.
    - ▶ The protocol details used for the publish action. When you create a Publication Target, you select a Protocol Schema that defines the values required for this configuration. You then fill in the appropriate values in the Publication Target.
    - ▶ The priority of the Publish action (Low, Normal, or High).

See also Chapter 4 "*Content Manager Publication Targets*" on page 11.
  - Transport Service, Deployer, and Broker configuration:
    - ▶ The Transport Service delegates the transport of published content (in Transport Packages) to a protocol-specific Sender. All Sender-related configuration information is specified in a transport service configuration file. See Chapter 7 "*Transport Service configuration*" on page 41.
    - ▶ The Deployer receives Transport Packages, and contains Processors that deploy or undeploy special types of content using modules that carry out the instructions. Content Deployer modules delegate the content to handlers within the Content Broker. The Receivers, Processors and Modules are specified in the Deployer configuration file. See Chapter 8 "*Deployer configuration*" on page 47.
    - ▶ The Content Broker contains handlers that determine where different types of content are stored. Based on the interface that a handler talks to, content can be stored in a SQL database (MS SQL, Oracle, DB2), a file system, or a Tamino database. See Chapter 9 "*Broker configuration*" on page 61.
  - Senders and Receivers — Sender and Receiver pairs are used to publish content using a specific protocol, which is determined by the Publication Target to which content is published.
    - ▶ Senders are configured in the Transport Service configuration file.
    - ▶ Receivers are configured in the Deployer configuration file.

In addition to Tridion supported Senders and Receivers, you can define custom Senders and Receivers.

See Chapter 6 "*Senders and Receivers*" on page 29.
  - Using metadata — Metadata is the descriptive information associated with every Component Presentation, Binary, Component, and Page published to the Content Distributor. Filters allow you to use this metadata to select a subset of Components and to query user-defined metadata.
- See Chapter 10 "*Using metadata*" on page 73.



# Chapter 4 Content Manager Publication Targets

A Publication Target defines the information required to publish content using different protocols and the target language to which content is published.

The publisher retrieves information about the location and other publishing settings from the Publication Target. This information is used by the Transport Service Sender to transport the Transport Package to the Deployer.

The Publication Target specifies the following publishing parameters:

- The target language and code page settings of published content.
- A minimum approval status of published content. For example, if the minimum approval status is "Approved", then only items that have reached this approval status through a Workflow Process can be published
- The protocol details used for the publish action. The values required for this configuration are determined by a Protocol Schema, which defines the type of information that a specific protocol requires for a successful publishing action.

This chapter describes:

- Publishing content
- Publishing protocols
- Configuring Publication Targets
- Publication Target Code Page values

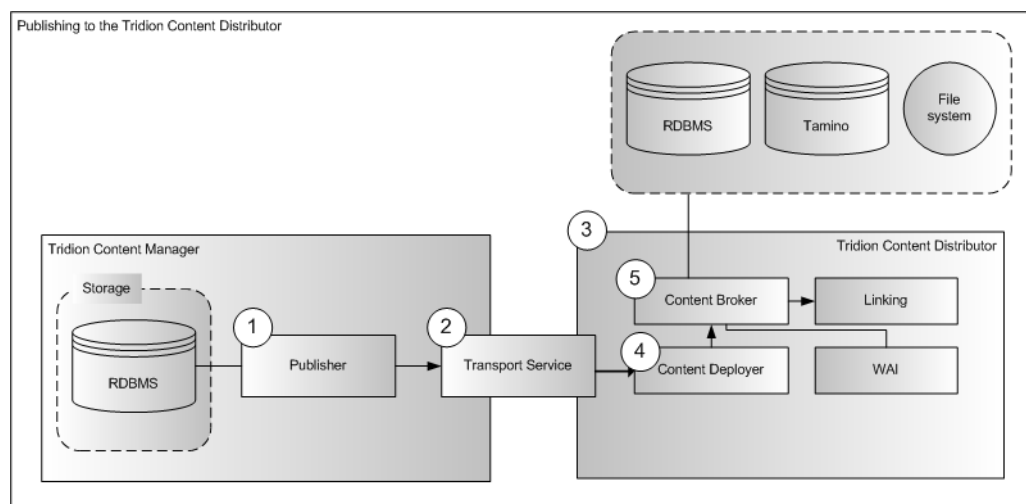
### 4.1 Publishing content

A user publishes content to a Target Type. A Target Type has the following characteristics:

- It is associated with one or more Publishing Target(s)
- It is used to grant users or groups permissions to publish to that Target Type

The Content Manager uses an internal publisher to render and package content and uses an internal Transport Service to deliver this content to the Content Distributor. Figure 4-1 depicts the stages involved in a publish action.

- 1 The Publisher initiates the distribution of content by resolving, rendering, and packaging items:
  - Resolving determines which items need to be rendered
  - Rendering assembles the output
  - Packaging puts all of the rendered output and publishing metadata into a Transport Package
- 2 The Transport Service sends the rendered content and metadata to the Content Distributor. The Transport Service invokes Senders that are responsible for sending content using the protocol information provided by the Publication Target.
- 3 A Receiver accepts the package. The Content Distributor then deploys and brokers content.
- 4 The Content Deployer receives and analyzes the Transport Package and sends content to the Content Broker.
- 5 The Content Broker stores content in a storage medium (such as a file system or database).



**Figure 4-1 Publishing**

*See also* Refer to the relevant chapters for more detailed information about configuring the Transport Service, the Content Deployer, and the Content Broker.



## 4.2 Publishing protocols

Tridion allows you to publish content using the following protocols:

- **Local File System** places content on the file system (local or network).
- **FTP** uses FTP to send a Transport package to a deployment platform.
- **SFTP** uses SFTP to send a Transport package to a deployment platform.
- **HTTP(S)** uses HTTP (and optionally SSL) to send a Transport Package to a deployment platform.

Sender and Receiver pairs are used to publish content using a specific protocol. The protocol used is configured in the Publication Target to which content is published.

Although Local File System, FTP, and SFTP all use different Senders, they use the same Receiver (*FileReceiver*). The *FileReceiver* monitors a specific location for Transport Packages. HTTP(S) uses a dedicated Receiver to receive Transport Packages sent using the HTTP(S) protocol.

Publication Target specifies destination information for one or more protocols. The Content Manager publisher creates a Transport Package. The Transport Package contains:

- files (binaries, content, and/or Pages)
- XML instructions about the delivery (timing and location)

The Transport Service uses the Transport Packages protocol information to delegate the transport of the package to a protocol-specific Sender (such as HTTP) on the Content Manager. All Sender-related configuration information is specified in a Transport Service configuration file (*cd\_transport\_conf.xml*). Receiver information is configured in the Content Deployer configuration file (*cd\_deployer\_conf.xml*).

For more information about Senders and Receivers, see Chapter 6 "*Senders and Receivers*" on page 29.

This section describes:

- Local File System, FTP and SFTP protocols
- HTTP(S) protocol

### 4.2.a Local File System, FTP and SFTP protocols

Local File System, FTP, and SFTP Senders are configured in the Management System interface as Publication Target Destinations.

A file Receiver polls the specified upload directory at predefined intervals. When a new Transport Package is detected, the file Receiver extracts the package to the specified WorkFolder. You must configure the *FileReceiver* to monitor the folder to which the files are delivered in the Content Deployer configuration file.



## Chapter 4 Content Manager Publication Targets

The Publication Target configuration includes the protocol information required to publish to these protocols. Table 4-1 outlines the values that are used.

**Table 4-1 Local File System, FTP, and SFTP Protocols**

<b>Protocol and description</b>	<b>Schema fields</b>
Local Places content on the local file system.	<b>Location</b> — the Folder to which the Local Copy Sender copies the Transport Package. (This Folder is monitored by the FileReceiver which retrieves and Transport Package).
SFTP Uses SFTP to send a Transport package to a deployment platform.	<b>FTPHost</b> — IP address or host of the FTP server <b>FTPPort</b> — Port number of the FTP server <b>FTPUsername</b> — Username to authenticate against the FTP server <b>FTPPassword</b> — Password to authenticate against the FTP server <b>Location</b> — the Folder to which the SFTP Sender copies the Transport Package. (This Folder is monitored by the Content Deployer which retrieves and Transport Package). <b>SSHHost</b> — IP address or host of the SSH server <b>SSHPort</b> — Port number of the SSH server <b>SSHUsername</b> — Username to authenticate against the SSH server <b>SSHPassword</b> — Password to authenticate against the SSH server
FTP Uses FTP to send a Transport package to a deployment platform.	<b>Host</b> — IP address or host of the FTP server <b>Port</b> — Port number of the FTP server <b>UserName</b> — Username to authenticate against the FTP server <b>Password</b> — Password to authenticate against the FTP server <b>Location</b> — the Folder to which the SFTP Sender copies the Transport Package. (This Folder is monitored by the Content Deployer which retrieves and Transport Package).

### Local file copy security considerations

The Transport Service and the Deployer service need to be able to read and write files for the configured locations to which the Transport Package is sent and retrieved. In a network scenario, the relevant service must run under a user account that has read/write permissions for the file receiver location.

By default, the Transport Service and Deployer service run under the LocalSystem user. This LocalSystem user cannot write to other machines.

## 4.2.b HTTP(S) protocol

HTTP(S) transports Transport Package content to a Content Delivery system using HTTP. This transfer is encoded using a multipart mimeupload request. HTTPS Transport can be encrypted.

The same Sender defines both HTTP and HTTPS transport. To specify one or the other, simply change the protocol scheme of the Publication Target. That is, when creating the Publication target, specify the URL as:

- `http://` if you intend to use HTTP
- `https://` if you intend to use HTTPS

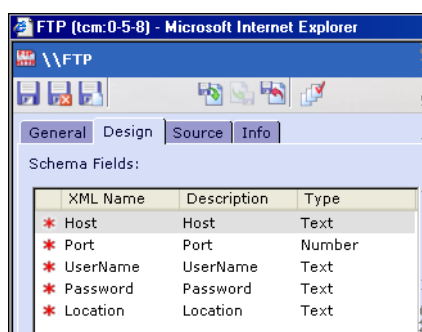
The Publication Target configuration includes the protocol information required to publish to HTTPS:

- **UserName** — User name used to connect to HTTP(S) server
- **Password** — Password used to connect to HTTP(S) server
- **URL** — URL HTTPS server and file upload location

## 4.2.c Creating a Protocol Schema

Protocol Schemas define the protocol information that is needed to transport content to the delivery system. Publication Targets specify which protocol(s) are used for transport and the associated information as defined by the protocol schema(s) for the selected protocol(s).

For example, an FTP Sender requires Host, Port, Username, Password, and Location information to publish content. A protocol schema called FTP is used to specify these fields (figure 4-2). When the protocol schema is used in a Publication Target, the values of these fields can be added (figure 4-3).

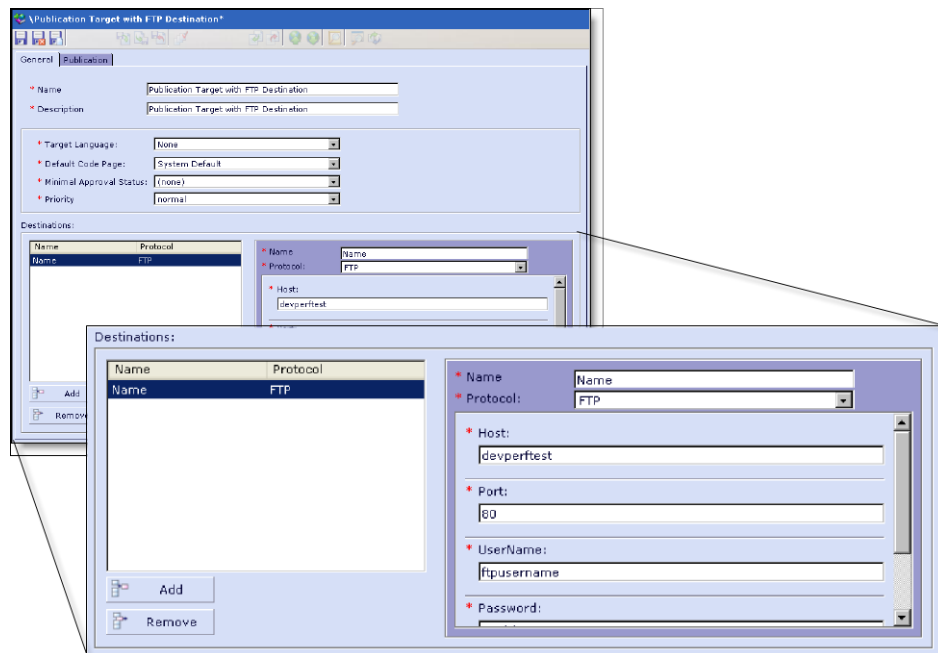


The screenshot shows a Microsoft Internet Explorer window titled "FTP (tcn:0-5-8) - Microsoft Internet Explorer". The address bar shows "\\FTP". The "Source" tab is selected, displaying a table of Schema Fields:

XML Name	Description	Type
* Host	Host	Text
* Port	Port	Number
* UserName	UserName	Text
* Password	Password	Text
* Location	Location	Text

**Figure 4-2 FTP Protocol Schema fields**





**Figure 4-3 A Protocol Schema used in a Publication Target**

When you install the Content Manager, the following default protocol schemas are installed:

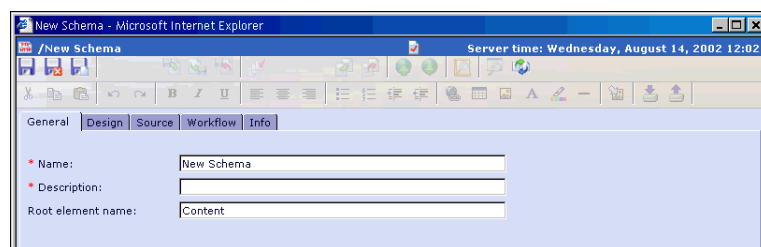
- Local file system
- FTP
- SFTP
- HTTP(S)

**Note** It is not possible to copy/paste a protocol schema directly into the Protocol Schema node. You must create the item through the GUI.

*Prerequisites* A User with system administration privileges can create a protocol schema.

### To create a Protocol Schema:

- 1 In CM Explorer, navigate to **System Administration>Publishing Management>Protocol Schemas**.
- 2 Select **Protocol Schemas** and click the **Add Schema** button on the toolbar. The **New Schema** window appears.



**Figure 4-4 New Schema.**

**To create a Protocol Schema: (Continued)**

- 3 On the **General** tab, fill in the following fields:
  - **Name** – A unique name for this Schema.
  - **Description** – A description of this Protocol Schema.
  - **Root element name** – The Sender protocol name, such as FTP, or HTTPS.
- 4 On the **Design** tab, add fields by clicking the **Add** button. For each added field specify the following field parameters:
  - **XML Name** – The definition of the parameter you want to pass to your Sender, such as Host, Port, or Username, for example. This name is case sensitive.
  - **Description** – A brief description of the parameter specified.
  - **Type** (either Text or Number) – You must specify Number if you are defining a port number.
  - If necessary, define parameters for the type
- 5 Repeat step 5 for each field you need to add.
- 6 Click the **Save & Close** button on the toolbar.

*Results* The Protocol Schema you have created can now be used to define destination details for a Publication Target.

## 4.3 Configuring Publication Targets

This section describes:

- Creating a Target Type
- Creating a Publication Target

### 4.3.a Creating a Target Type

A Target Type specifies a user-friendly name for one or more Publication Targets and specifies permission settings for the target(s). The Content Manager uses Target Types to identify one or more Publication Targets. When a user publishes from the Content Manager, the user publishes to a Target Type. Content is then published to the Publication Targets associated with these Target Types.

For example, your organization may have two different Target Types:

- A Target Type called Staging to which authors can publish content.
- A Target Type called Live to which editors can publish content.

In this example, the Target Types are associated with Publication Target types. The Staging Target Type may publish to a local file system, while the Live Target Type may publish to an HTTP server that is accessible by external visitors.

## Chapter 4 Content Manager Publication Targets

You can create Target Types in the Tridion Content Manager (CM) Explorer.

### To create a Target Type:

- 1 In CM Explorer, navigate to **System Administration>Publishing Management>Target Types**.
- 2 Click the **Add Target Type** button on the toolbar.
- 3 In the **New Target Type** window that appears, fill in the following fields:
  - **Name** — The name of the Target Type (For example, Live)
  - **Description** — A Description of the Target Type (For example, "Staging" or "Live")
- 4 Click the **Save & Close** button on the toolbar.

**Results** For Users to be able to publish to this Target Type you must now complete the following actions:

- Add this Target Type to one or more Publication Targets. After it is added to a Publication Target, Users can select the name of a Target Type from a list. For more information, refer to 4.3.b "*Creating a Publication Target*" on page 19.
- Modify the security settings of the Target Type by editing the Target Type.

Edit a Target Type to change the Name and/or Description of the Target Type and to give Users permission to publish to the Target Type.

### To edit a Target Type:

- 1 In CM Explorer, navigate to **System Administration>Publishing Management>Target Types**.
- 2 In the list view, select the Target Type you want to edit and click the **Open** button on the toolbar.
- 3 To edit information on the **General** tab, modify field values as required.
- 4 To grant or remove permission to publish to the Target, select the **Security** tab.

On the Security tab, select the **Show All** check-box and select the Users or Groups that should have permission to publish to this Target Type. In the Permissions field, select the Use Target Type check-box.

- 5 Click **Save & Close** on the toolbar.

**Results** You can now add this Target Type to one or more Publication Targets. After it is added to a Publication Target, Users can select the name of a Target Type from a list when publishing items.

### 4.3.b Creating a Publication Target

Create a Publication Target to specify information about a publish action (such as the location and protocol of a publishing action).

When an item is published to a Target Type associated with this Publication Target, the Publication Target information is passed on to the Transport Service. This information is used by the Content Distributor when the publishing action takes place.

#### Prerequisites

Before you can create a Publication Target and configure its settings, you may need to create Protocol Schemas and Target Types if they do not already exist.

Users with system administration privileges can create Publication Targets.

#### To create a Publication Target:

- 1 In CM Explorer, navigate to System **Administration>Publishing Management>Publication Targets**.
- 2 Click the **Add Publication Target** button on the toolbar. A **New Publication Target** window opens.

**Figure 4-5 Publication Target – General tab**

- 3 On the **General** tab fill in the following fields:
  - **Name:** A unique name for this Publication Target.
  - **Description:** A description of the Publication Target.
  - **Target Language:** The script language for generating script code
  - **Default Code Page:** The code page used for the published content. (See also: "*Publication Target Code Page values*" on page 21)



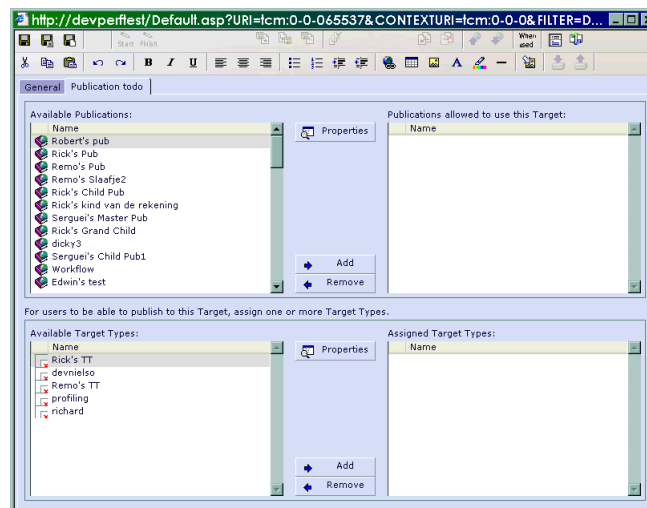
### To create a Publication Target: (Continued)

**Note** You may need to change the Default Code Page for Pages that contain other character types (such as Japanese). Usually the setting for languages is UTF-8. Templates may override this code page setting.

- **Minimum Approval Status:** The minimum Workflow Approval Status required for an item in Workflow for the item to be published. If the Workflow version of an item does not have the required status, the latest checked-in version is rendered instead.
  - **Priority:** The priority to which publish actions to this Publication Target will have. High, Medium or Low.
- 4** In the **Destinations** area, you can add one or more destinations to which the Transport Package will be sent. Each destination has a protocol. After you select a protocol, you can specify the relevant protocol information.

Click the **Add** button and fill in the following fields:

- **Name:** the name of the destination location
  - **Protocol:** protocol information for this destination. The selected protocol is based on existing Protocol Schemas. Fill in protocol information. "*Publishing protocols*" on page 13 provides a description of the information required for each protocol.
- 5** Select the **Publication** tab and fill in the following fields:
- To select Publications that can Publish to this Publication Target, select **Publications** from the list of Available Publications and click the **Add** button.
  - To add Target Types to this Publication Target, select **Available Target** and click the **Add** button.



**Figure 4-6 Publication Target – Publication tab**

- 6** Click the **Save & Close** button on the toolbar.

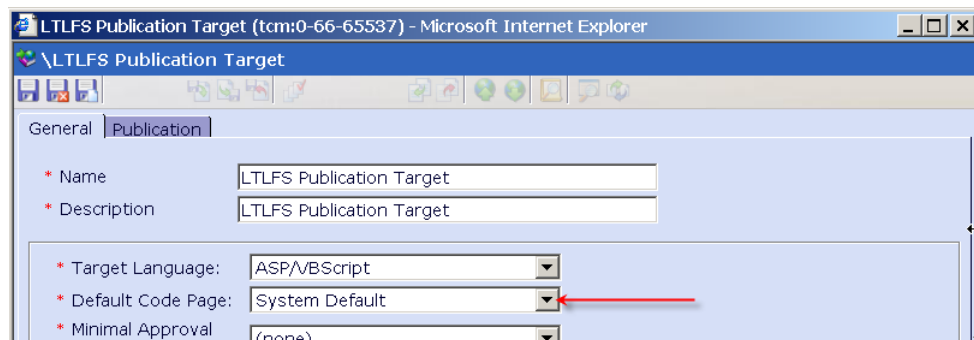
**Results** Specified Publications can now publish to the Publication Target. When a User performs a publish action in a Publication, the User can select the Target Type for which they have permission. This Target Type publishes to the associated Publication Target(s).



## 4.4 Publication Target Code Page values

Component Presentations and Pages that are sent to the Content Manager Publisher as part of a Transport Package are sent with their "code page" values. These values are set by the Publication Target "Default Code Page" value.

When these Component Presentations and Pages are retrieved by the Content Distributor, these character encodings need to be translated to Java *character set names*. This is done using a mapping file that is located in the Content Broker jar file.



**Figure 4-7 Default Code Page: System Default**

If you do not use the System Default settings for the Default Code Page, you must make the following changes to your Page Templates and Component Templates to ensure that web browsers correctly interpret content rendered by these templates.

**Note** Add the code described in table 4-1 to Page Templates before any `<html>` tags. For Dynamic JSP Component Templates, add the code to the top of the Component Template.

**Table 4-1 Additions to Page Templates and Component Templates when the default code page value is not System Default**

Item type	Example <sup>1</sup>
ASP Page Template	<pre>&lt;% Response.CodePage="65001" Response.ContentType="text/html; charset=UTF-8" %&gt;</pre>
ASP.NET Page Template	<pre>&lt;% Response.ContentType="text/html; charset=UTF-8" %&gt;</pre> <p>See also "<i>ASP.NET IIS configuration</i>" on page 22 (below).</p>
JSP Page Template	<pre>&lt;%@ page contentType="text/html; charset=UTF-8" %&gt;</pre>
Dynamic ASP Page Template	<pre>&lt;% Response.ContentType="text/html; charset=UTF-8" %&gt;</pre>
Dynamic JSP Page Template and dynamic Component Template	<pre>&lt;%@ page contentType="text/html; charset=UTF-8" %&gt;</pre> <p><b>Important:</b> For dynamic JSP, this line must be added to both the Page Template and the dynamic Component Template.</p>

1. These examples are for UTF-8. Modify the values according to the encoding that you intend to use.

If necessary, you can override mappings or add mappings. For more information, see "*Overriding mappings for code page values*" on page 22.

### ASP.NET IIS configuration

For ASP.NET, in addition to the template changes indicated in table 4-1, you also need to configure IIS with a file encoding value for these ASP.NET pages. The relevant `web.config` file should contain the following minimum content:

```
<configuration>
  <system.web>
    <globalization fileEncoding="utf-8"/>
  </system.web>
</configuration>
```

---

**Important:**

This is an example only, and is for UTF-8. Change the values according to the encoding that you intend to use.

---

### 4.4.a Overriding mappings for code page values

**Note** This section applies to 5.1 SP3 and higher.

Component Presentations and Pages that are sent to the Content Manager Publisher as part of a Transport Package are sent with their "code page" values. These values are set by the Publication Target "Default Code Page" value.

When these Component Presentations and Pages are retrieved by the Content Distributor, these character encodings need to be translated to Java *character set names*. This is done using a mapping file that is located in the Content Broker jar file.

If the existing mappings are not sufficient for your requirements, you can override mappings or add additional mappings.

---

**Important:**

Before you override any mappings, ensure that you have specified correct HTTP character set header in your templates. For more information, refer to "*Publication Target Code Page values*" on page 21.

---

To override mappings or add mappings, you create a file (`codepage_encoding.properties`) in which you specify name=value pairs, where:

- name is the codepage number
- value is the java character set name

For example, to map to Microsoft code page 1200 (which represents UTF-16) to the Java equivalent (which is the character set UTF-16) you add the following line to the properties file:

```
1200=UTF-16
```

**Note** This mapping is for illustration purposes only. This mapping is already part of default mappings that are shipped with the Tridion Content Broker.

**To override or add additional mappings:**

- 1** Create or open the following Java properties file:  
`codepage_encoding.properties`
- 2** Put this file on the classpath of the Deployer or Broker. (For example, in the same location as the configuration files.)
- 3** In this file, add name=value pair mappings with the following syntax:
  - Separate all items with "="
  - Start a new line for each new mapping.
  - Add comments with lines starting with "#".







# Chapter 5 **Configuring ASP.NET support for the Content Distributor**

This section is for system administrators that have installed ASP.NET support as part of the Content Delivery product installation.

If you selected .NET support as part of your installation the installer has copied to the required .NET wrappers to the bin directory of your selected installation directory. By default, this location is:

```
C:\Program Files\Tridion\bin
```

To configure .NET support you must have fully functional COM support. COM support requires the following:

- a correct setup of the Java libraries
- valid configuration files
- installed and running services

This chapter describes:

- Configuring assemblies
- Configuring assemblies for a web application
- Configuring assemblies for a standalone application
- Configuring WAI profiling and tracking
- Publishing to ASP.NET

## 5.1 **Configuring assemblies**

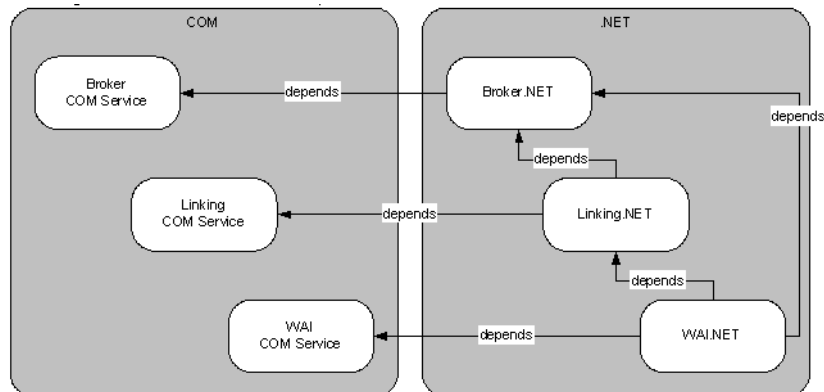
The following assemblies are installed in the bin folder of your installation directory (default location is C:\Program Files\Tridion\bin):

- Tridion.ContentDelivery.Broker.dll
- Tridion.ContentDelivery.Broker.Interop.dll
- Tridion.ContentDelivery.Linking.dll

## Chapter 5 Configuring ASP.NET support for the Content Distributor

- Tridion.ContentDelivery.Linking.Interop.dll
- Tridion.ContentDelivery.WAI.dll
- Tridion.ContentDelivery.WAI.Interop.dll
- Tridion.ContentDelivery.WAI.Interop.ASP.dll

Figure 5-1 depicts the dependencies between .NET assemblies and their COM counterparts.



**Figure 5-1 Dependencies between .NET assemblies and COM counterparts**

### 5.1.a Configuring assemblies for a web application

To configure assemblies for a web application, you can do one of the following:

- add the assemblies to the bin folder of your Website (recommended)
- register the DLLs in the Global Assembly Cache

**Note** Consult the Microsoft Development Network Library online at <http://msdn.microsoft.com/library/default.asp> to learn about other ways of loading assemblies.

### 5.1.b Configuring assemblies for a standalone application

To configure assemblies for a standalone application, do one of the following:

- reference the individual DLLs
- copy the DLLs to the project folder
- add the DLLs to the global assembly cache

## 5.2 Configuring WAI profiling and tracking

The Web and Application Integration (WAI) uses a .NET web module. At the beginning of a request, the module handles the authentication and persistence of users. At the end of a request, the module invokes tracking of the visited content.

### To configure profiling and tracking for use with .NET:

- 1 Create a `web.config` configuration file in the root of your web site. If a `web.config` file already exists, you can use the existing file.
- 2 Configure the WAI HTTPModule:  
`Tridion.ContentDelivery.WAI.WAIModule.`

The `web.config` should appear as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.web>
  <!-- Other configuration such as session persistence and
  authentication -->
<httpModules>
  <add
    type="Tridion.ContentDelivery.WAI.WAIModule,
    Tridion.ContentDelivery.WAI"
    name="TridionWAIHttpModule" />
</httpModules>
  <!-- Other configuration such as session persistence and
  authentication -->
</system.web>
</configuration>
```

## 5.3 Publishing to ASP.NET

As of 5.1 SP3, it is possible to publish content to ASP.NET. This is performed using an intermediate deployment language.

ASP.NET output uses:

- a Publication Target that specifies the intermediate language format (Tridion Content Deployer Language — TCDL) as an output format
- the Content Deployer ASP. NET configuration, which transforms the intermediate language format to ASP. NET

**Note** The addition of this functionality has no impact on your installation. This section is intended to outline how you can implement this functionality.

Unlike other output formats, the Content Manager does not transform the content to the published format. Instead, the Content Manager transports the content in TCDL. The Content Deployer then transforms this format into ASP. NET.



## Chapter 5 Configuring ASP.NET support for the Content Distributor

In the Deployer configuration file (`cd_deployer_conf.xml`), the optional `Transformer` element can be used if you want to publish content to ASP.NET. Add the `Transformer` element as a sub-element to the `PageDeploy` module. The `Transformer` element specifies the following `Class` attribute:

```
Class="com.tridion.transformer.ASPDotNETTransformer"
```

The following example depicts this element used for `PageDeploy`:

```
<Processor Action="Deploy" Class="com.tridion.deployer.Processor">
  <Module Type="PageDeploy"
    Class="com.tridion.deployer.modules.PageDeploy">
    <Transformer Class="com.tridion.transformer.ASPDotNETTransformer" />
  </Module>
  <Module Type="SchemaDeploy"
    Class="com.tridion.deployer.modules.SchemaDeploy"/>
  <Module Type="ComponentPresentationDeploy"
    Class="com.tridion.deployer.modules.ComponentPresentationDeploy">
  </Module>
  <Module Type="BinaryDeploy"
    Class="com.tridion.deployer.modules.BinaryDeploy"/>
  <Module Type="ComponentDeploy"
    Class="com.tridion.deployer.modules.ComponentDeploy"/>
  <Module Type="TemplateDeploy"
    Class="com.tridion.deployer.modules.TemplateDeploy"/>
</Processor>
```

When publishing to ASP.NET, the following conditions apply:

- The Deployer is configured to use the `ASPDotNETTransformer`
- The Publication Target to which content is published specifies the TCDL (Tridion Content Deployer Language) as the output language
- The Page Template file extension is `.aspx`
- The ASP.NET libraries are installed on the presentation server

ASP.NET output requires:

- a Publication Target that specifies the intermediate language format (Tridion Content Deployer Language — TCDL) as an output format
- the Content Deployer ASP.NET configuration, which transforms the intermediate language format to ASP.NET
- a `web.config` file in the root of the web site to which you want to deploy Page. This file requires the following lines:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation defaultLanguage="c#" debug="false" />
  </system.web>
</configuration>
```

Unlike other output formats, the Content Manager does not transform the content to the published format. Instead, the Content Manager transports the content in TCDL. The Content Deployer then transforms this format into ASP.NET.

**Important:** If you want to publish to ASP.NET using a Code Page Value that is not the default value, see also 4.4 "Publication Target Code Page values" on page 21.



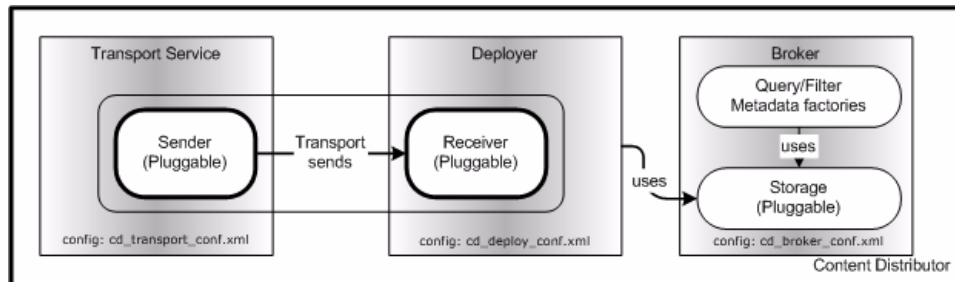
# Chapter 6 Senders and Receivers

Sender and Receiver pairs are used to publish content using a specific protocol. The Publication Target protocol determines how content is published. The Transport Service receives destination information using Publication Target information and the relevant Sender is created. You configure Senders in the Transport Service configuration file.

Using the specified configuration, the Sender then sends the Transport Package to a Receiver that is invoked by the Content Distributor.

Tridion supports the following Sender-Receiver pairs:

- Local file
- FTP
- SFTP
- HTTP(S)



**Figure 6-1 Sender and Receivers.**

This chapter describes:

- Senders
- Receivers
- Local Copy, FTP and SFTP Senders and Receivers
- HTTP(S) Senders and Receivers
- Developing custom Senders and Receivers

## 6.1 Senders

Configure Senders in the Transport Service configuration file:

`cd_transport_conf.xml`.

The `Senders` element defines the Sender types you intend to use. Senders are configured in the Management System interface as Publication Target Destinations.

Specify a Sender by providing a 'Type' that matches the root element name of a Content Manager Protocol Schema. The 'Class' attribute specifies the Java class that implements the functionality for a Sender.

For example:

```
<Sender Type="FTP" Class="com.tridion.transport.FTPSender"/>
```

---

### Important:

---

The value for Sender Type in the sender configuration and the value of the Protocol Schema name must be identical including case.

---

The following are the available Sender types, and the classes responsible for implementing the respective protocols:

- Local — `com.tridion.transport.LocalCopySender`
- FTP — `com.tridion.transport.FTPSender`
- SFTP — `com.tridion.transport.SFTPSender`
- HTTP(S) — `com.tridion.transport.HTTPsSender`

For more information about configuring Senders, refer to "*Transport Service configuration*" on page 41.

## 6.2 Receivers

Configure Receivers in the Content Deployer configuration file:

`cd_deployer_conf.xml`.

The `ReceiverBindings` element defines the types of Receivers that the Deployer can use. The `ReceiverBindings` element can have the following attributes:

- The `Type` attribute identifies the Receiver. For example, "`HTTPSReceiver`"
- The `Class` attribute defines the implementation used for receiving the Transport Packages. For example, `com.tridion.transport.HTTPSReceiver`

The `Receivers` element specifies which Receiver instances the Deployer starts. The element names for these Receivers must match the `Type` attribute defined in the `ReceiverBindings` section.

For example

```
<ReceiverBindings>
  <Receiver Type="FileReceiver"
    Class="com.tridion.transport.FileReceiver"/>
</ReceiverBindings>
```

The following Receiver classes are available:

- Local Copy — `com.tridion.transport.FileReceiver`
- FTP — `com.tridion.transport.FileReceiver`
- SFTP — `com.tridion.transport.FileReceiver`
- HTTP(S) — `com.tridion.transport.HTTPSReceiver`

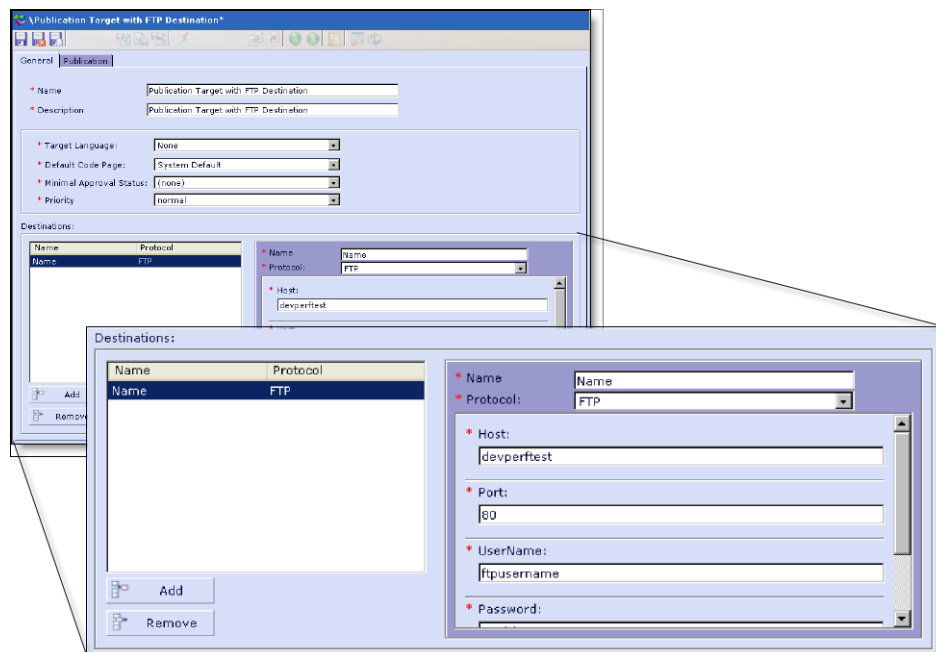
For more information about configuring Receivers, refer to "*Deployer configuration*" on page 47.

### 6.3 Local Copy, FTP and SFTP Senders and Receivers

The Transport Service configuration on the management system configures the following Senders to send a Transport Package to a specified location:

- LocalCopy — a location on the file system
- FTP — a location on an FTP server. The FTP Sender uses the `ftp.jar` library to transfer content to the Deployer.
- SFTP — a location on an SFTP server. The SFTP Sender uses the `sftp.jar` library to transfer content to the Deployer.

Senders are configured as Publication Target Destinations (figure 6-2).



**Figure 6-2 A Publication Target using FTP**

The file Receiver polls the specified upload directory (the Location value of the Publication Target) at predefined intervals. When a new Transport Package is detected, the file Receiver extracts the package to the specified WorkFolder.

In the Content Deployer configuration file, you configure the file Receiver to monitor the folder to which the files are delivered using the following attributes:

- `Location` — specifies folder that the Receiver should monitor.
- `Interval` — specifies the frequency (in seconds) with which the Receiver checks the location for new files.

```
<FileReceiver Location="C:\Tridion\incoming" Interval="1"/>
```

If the Deployer is not running when the Transport Packages are uploaded to the file system, they are processed when the Deployer is restarted.

The Deployer must run as an independent process:

- On Windows-based systems, start the Tridion Content Deployer service
- On UNIX-based systems, add a script file to the boot procedure that starts the process. For example:

```
#!/bin/sh
java -cp
./classes:./lib/cd_core.jar:./lib/cd_transport.jar:./lib/cd_deployer.jar:./lib/xalan.jar:./lib/xercesImpl.jar:./lib/xmlParserAPIs.jar:./lib/jndi.jar:./lib/ezlicrun.jar com.tridion.deployer.Deployer
```

### Local copy Sender and Receiver example

The Local Copy Sender configured in `cd_transport_conf.xml`:

```
<Sender Type="LocalCopy"
Class="com.tridion.transport.LocalCopySender"/>
```

The FileReceiver configuration in `cd_deployer_conf.xml`:

```
<FileReceiver Location="C:\Program Files\Tridion\FileReceiver"
Interval="1"/>
```

### FTP Sender and Receiver example

The FTP Sender configured in `cd_transport_conf.xml`:

```
<Sender Type="FTP" Class="com.tridion.transport.FTPSender"/>
```

The FTP FileReceiver configured in `cd_deployer_conf.xml`:

```
<FileReceiver Location="C:\Tridion\incoming" Interval="1"/>
```

### SFTP Sender and Receiver example

The SFTP Sender configured in `cd_transport_conf.xml`:

```
<Sender Type="SFTP" Class="com.tridion.transport.SFTPSender"/>
```

The SFTP FileReceiver configured in `cd_deployer_conf.xml`:

```
<FileReceiver Location="C:\Tridion\incoming" Interval="1"/>
```

## 6.4 HTTP(S) Senders and Receivers

HTTP(S) transports Transport Package content to a content delivery system using an HTTP POST request.

HTTPS Transport can be encrypted. Although the transport method is called HTTPS, the use of SSL and of username/password is optional. This document does not describe how to configure the Web server to use SSL certificates.

The URL of the HTTPS transport method depends on the delivery system configuration. For example:

- ASP — `https://www.visitorsWeb.com/upload/httpupload.asp`
- J2EE — `https://www.visitorsWeb.com/httpupload`

The same Sender defines both HTTP and HTTPS transport. In order to specify one or the other, simply change the URL in the Publication Target. That is, when creating the Publication target, specify the URL as:

- `http://` if you intend to use HTTP
- `https://` if you intend to use HTTPS

The remainder of the URL must be the address of a correctly configured Web server, which contains the corresponding HTTPS Receiver.

To create an HTTPS transport Sender, add the following line to the Transport Service configuration file:

```
<Sender Type="HTTPS" Class="com.tridion.transport.HTTPSSender"/>
```

You must also specify an HTTP(S) Receiver in the Deployer configuration file:

```
<Receiver Type="HTTPSReceiver"  
Class="com.tridion.transport.HTTPSReceiver"/>
```

For information about the necessary jar files to enable HTTPS, refer to Appendix I "Third-party libraries" on page 175.

### 6.4.a Troubleshooting transport over HTTP(S) to IIS

When you transport large Transport Packages over HTTP or HTTPS to an IIS server, the transport may fail due to one or both of the following reasons:

- The receiving IIS server refuses to accept the Transport Package because the Transport Package is too large. You can find out if this is the case by checking the IIS log file. It should contain a 403 HTTP error code.
- The ASP page that receives the Transport Package times out because the Transport Package is too large. This will be reported as a failed transport in the Transport Log file (see "Logging" on page 43 to locate this log file).

#### Configuring IIS

To prevent 403 HTTP errors from IIS, you can configure IIS so that it accepts larger pieces of data offered through the HTTP POST command (the command used by Transport Packages). By default, IIS sets a size limit of 200 Kilobytes on

such data. In IIS 6.0, you can increase this maximum size by editing the `AspMaxRequestEntityAllowed` property in the IIS Metabase XML file. Refer to the MSDN Library at <http://msdn.microsoft.com/library/> for more details.

### Changing the timeout limit of the ASP page

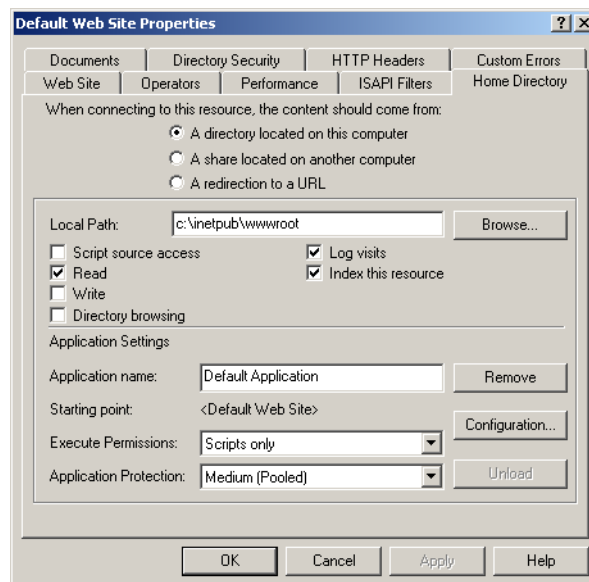
To prevent the ASP page from timing out, you can configure the Web site to allow ASP scripts to wait longer for incoming data.

#### To change the timeout settings for ASP scripts:

- 1 On the machine to which you are transporting, access the Windows Control Panel, access Administrative Tools, then access the Internet Services Manager.

The Internet Services Manager displays the Web sites running on the machine.

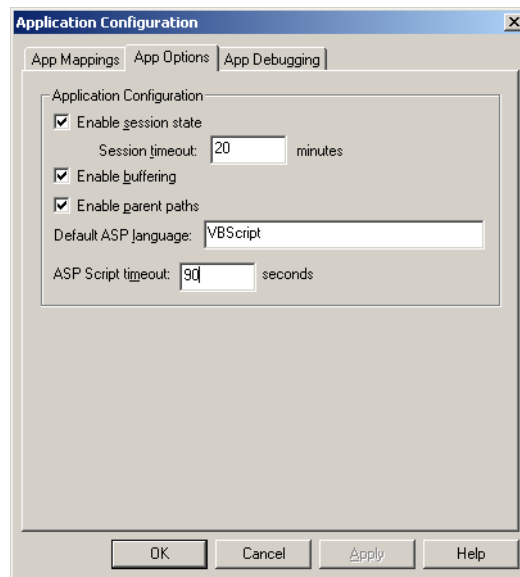
- 2 Right-click the Web site that contains the ASP page and select **Properties** from the context menu. A Properties window opens for this Web site.



**Figure 6-3 Web site properties in Internet Services Manager**

- 3 Click the **Home Directory** tab and click **Configuration** in the **Application Settings** area. The **Application Configuration** window opens. Select the **App Options** tab.

To change the timeout settings for ASP scripts:



**Figure 6-4 App Options tab in Application Configuration**

- 4 Set the value (in seconds) for the **ASP Script timeout**. The default value is 90 seconds. Increase the value to allow the ASP page to wait longer.
- 5 Click **OK** to close the **Application Configuration** window. Then click **OK** close the **Web site Properties** window.
- 6 Close the Internet Services Manager.

**Note** If your Web site contains other ASP pages that should keep the original timeout value, you can set the timeout on a Web folder level from the Internet Services Manager, or even set it in the source code of the specific ASP page. Refer to the MSDN Library for details.

### Limiting Transport Package size

Regardless of how you configure the Web site, it is always possible that a Transport Package will be too big to transport. It is best to find this out before you actually start sending the Transport Package, because attempting this transport will slow down the Web site.

So rather than waiting for the HTTP 403 error to return or for the ASP page to time out, it is better to catch these large Transport Packages at the source.

You can detect large Transport Packages by setting an element in the Content Deployer configuration file called `HTTPSReceiver`. This element, which is commented out by default, has an attribute `MaxSize`. By uncommenting it and setting the value of this attribute to a certain value, any Transport Package larger than this size will automatically fail to transport, and such failure will be logged.

For more information about configuring the Content Deployer, see "Configuring the Deployer" on page 48.



### 6.4.b Importing a certificate for HTTPS uploads

The Transport Service can only perform HTTPS uploads (necessary for HTTPS transport) if it can find a certificate that permits it to access the destination. If no such certificate is available, transport will fail.

#### To make a certificate available to the Transport Service:

- 1 Find the Transport Service log file. The location and name of this file are configured in the Transport Service configuration file (`cd_transport_conf.xml`). By default, the log path and file is `c:\Program Files\Tridion\log\cd_transport.log`.
- 2 If you cannot find the log file at the configured location, you may have to start the Transport Services (under **Services** in the Windows Control Panel). As soon as the service starts, it creates a log file in the configured location.
- 3 Open the log file in a text editor. At the top of the file is a list of the system properties. Search for a property called `java.home`. The value of this property is the location of your Java Runtime Environment.
- 4 Import the certificate from the destination into the `lib\security` subdirectory of the Java home location you found.

To perform this import, you can use either a command-line tool called `keytool`, or the GUI-based `Policy Tool`. Both are shipped with the Java SDK. The following URLs explain how to use these tools:

<http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html>

<http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/policytool.html>

### 6.4.c HTTPS with IIS

The HTTPS transport method for IIS uses an ASP page to handle incoming packages.

Your installation CD-ROM contains the following files in the `Content Delivery/windows/httpupload` directory:

- A single ASP page called `httpupload.asp`  
Create a separate Web site within the IIS management console and add this page to the root of this site.
- The ASP page uses a COM module that handles the upload of files:  
`cd_upload.dll`.

Copy this file to the machine running the IIS server and register this module using the following command line:

```
regsvr32 cd_upload.dll
```

The ASP upload page sends incoming packages to a running Deployer through DCOM. The Deployer service must be running on the same machine as IIS.

You can test the installation by requesting the ASP Page using a Web browser. The following content should be returned to the browser:

```
TCDFileUpload.FileUpload.1.1 error '80004005'  
TCDFileUpload::SaveToDisk(): no data to save (due to 'GET' instead of
```



```
'POST', maybe?)  
/httpupload.asp, line 9
```

The URL that you use to get this message can be used as the value of the HTTPS protocol URL field in your Content Manager Publication Target.

#### 6.4.d HTTPS with a Java Web/Application Server

When you use HTTP upload in a Java Web environment, you do not need to run the Deployer as a separate process, it will run in the context of the Java Web/application server.

To use HTTP upload, use the Web Archive (WAR) (`cd_upload.war`), which is delivered in the `Content Distributor/java/httpupload` folder of your installation CD-ROM. This file contains the required libraries and configuration needed to handle incoming file requests and is deployed using the deployment tools in your Web/application server.

The HTTP upload servlet connects with a running Deployer instance within the application server (the Deployer is started if necessary).

The JAR and configuration files used by the Deployer must therefore be available to the `httpupload` Web application. The JAR files should be placed in the library folder of the Web application and a `cd_deployer_conf.xml` file should be placed on the CLASSPATH of the Web application (e.g. in the "classes" folder). Please refer to the documentation of your application server for more information.

After you deploy the WAR file, a configured servlet called `httpupload` is installed relative to the location of the root of the Web server. You can test the installation by opening the URL of this servlet in a Browser. The following content should be returned to the browser:

```
Tridion HTTP Upload and Responder Servlet
```

The URL that you use to get this message can be used as the value of the HTTPS protocol URL field in your Content Manager Publication Target. If the Deployer is not running when the Transport Packages are uploaded to the HTTPS Server, it will automatically be started by the servlet.



### Tomcat example

The following Tomcat example specifies the JAR files that you need to copy over.

This section describes how to configure HTTP(s) using the Apache Tomcat application server.

#### To configure HTTP(S) using the Apache Tomcat application server:

- 1 Copy `cd_upload.war` to Tomcat's `webapps` directory so that it is deployed. This will create a Web-app under the folder `cd_upload`.
- 2 Copy the following JAR files from the `Content Distributor/java/lib` folder on the CD into the `cd_upload/WEB-INF/lib` folder:

- `cd_broker.jar`
- `cd_core.jar`
- `cd_deployer.jar`
- `cd_transport.jar`
- `ezlicrun.jar`
- `xalan.jar`
- `xercesImpl.jar`
- `xmlParserAPIs.jar`

- 3 Create a folder called "classes" under the `WEB-INF` folder and copy the `cd_deployer_conf.xml`, `cd_broker.conf.xml` and `cd_licenses.xml` files to the root of this folder.
- 4 Ensure the configuration files are configured appropriately by ensuring that the following configuration elements are present in the `cd_deployer_conf.xml` configuration file:

```
<ReceiverBindings>
  <Receiver Type="HTTPSReceiver"
    Class="com.tridion.transport.HTTPSReceiver"/>
</ReceiverBindings>
<Receivers>
  <HTTPSReceiver MaxSize="10000000"/>
</Receivers>
```

- 5 If any third party libraries are required for deployment (e.g. jdbc drivers if you are deploying to a database) these should also be placed under `WEB-INF/lib`.
- 6 Restart the `cd_upload` Web application.
- 7 The directory `cd_upload` is created from the `cd_upload.war` file, and is accessible using `http://localhost:8080/cd_upload/`. To access the `httpupload` servlet, use the URL `http://localhost:8080/cd_upload/httpupload`
- 8 Configure your Publication target, in the Content Manager, to publish to HTTP. Enter the following URL in the Publication target: `http://localhost:8080/cd_upload/httpupload`.

## 6.5 Developing custom Senders and Receivers

This section describes how to create custom Sender and Receiver modules and, as an example, describes how to create an SMTP Sender and a POP Receiver.

### To create a custom Sender:

- 1 Create a subclass of `com.tridion.transport.Sender` that implements the `send()` method.
- 2 Compile the class.
- 3 Add the class to the transport service configuration file (`cd_transport_conf.xml`):

```
<Senders>
  <Sender Type="Type" Class="com.tridion.examples.TypeSender"/>
</Senders>
```

- 4 Create a Protocol Schema for the Sender.
- 5 Use the Protocol Schema in a Publication Target to define the values.

The custom Sender will retrieve the values defined by the Protocol Schema using its `configure()` method.

### To create a custom Receiver

- 1 Create a subclass to `com.tridion.transport.Receiver` that implements the `listen()` method.
- 2 Compile the class.
- 3 Add the custom Receiver to the Deployer configuration file (`cd_deployer_conf.xml`). For example, for a POP3 Receiver:

```
<ReceiverBindings>
  <Receiver Type="Type"
Class="com.tridion.examples.TypeReceiver">
</ReceiverBindings>
<Receivers>
  <TypeReceiver MyParameter="Myvalue"/>
</Receivers>
```

The custom Receiver will receive the configuration values in its `configure()` method. The custom Receiver is loaded and initialized when the Deployer starts.

For examples of a custom Sender and Receiver, refer to:

- Appendix A "Example: An SMTP Custom Sender" on page 157
- Appendix B "Example: Custom POP3 Receiver" on page 159





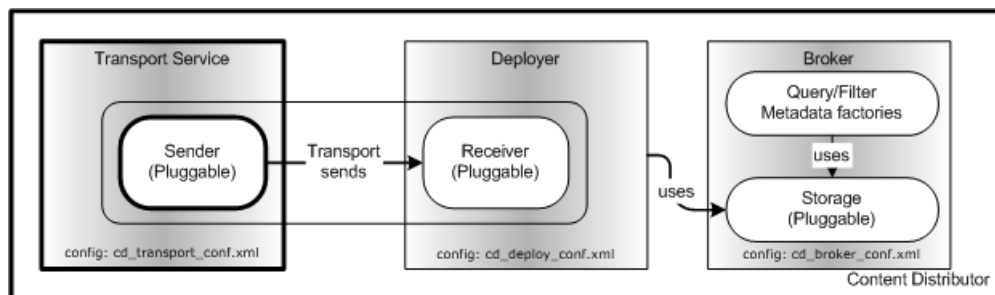
# Chapter 7 Transport Service configuration

The Transport Service resides on the Content Manager server. When content is published from the Content Manager, the Content Manager publisher prepares the content using the published content and resolves and renders the content.

The publisher packages the content into a Transport Package and delivers this package to the Transport Service. This Transport Package contains files (binary and Pages). The protocol information is based on the Publication Targets to which the content was published and the configuration of the transport service.

The Transport Service then uses the Transport Packages protocol information to delegate the transport of the package to a protocol-specific Sender (such as HTTP) on the Content Manager. All Sender-related configuration information is specified in a Transport Service configuration file.

Using the specified configuration, the Sender then sends the Transport Package to a Receiver that is invoked by the Content Distributor.



**Figure 7-1 Transport Service**

The following protocols are supported by Tridion Sender-Receiver pairs:

- Local Copy
- FTP
- SFTP
- HTTP(S)

The Content Distributor system allows you to publish your content to multiple targets using multiple protocols. This allows you, for example, to publish your content over HTTP to your live server, while simultaneously transferring a backup of the content to a remote location using FTP.

**Important:**

For Local File Copy, the User account that runs the transport service must have permission to write to the machine on which the Deployer runs. Either ensure that the folder the Transport Package is copied to is on the Transport Service machine, or create a user that is domain aware to run the Transport Service and grant this user permission on the Deployer machine.

## 7.1 Configuring the Transport Service

The Transport Service configuration specifies the configuration values that the publisher requires to initiate and handle a publish action. Using the transport configuration file, you can configure the following:

- `WorkFolder` — The location where the Sender looks for the Transport Package.
- `Logging` — The behaviour and location of one or more logging streams.
- `Senders` — The Sender types that you intend to use.
- `Pollers` — The interval and number of attempts made to poll the Receiver to determine if content has successfully been transported to the Receiver or not.

Configure the Transport Service using the configuration file `cd_transport_conf.xml`. The configuration of this file is validated against the schema `cd_transport_conf.xsd`.

The following example of the transport service configuration file uses the local Sender:

```
<?xml version="1.0" encoding="UTF-8"?>
<TransportService>
  <WorkFolder Location="C:\tridion\work"/>
<Logging>
  <Logger Level="error">
    <FileLogger Location="C:\tridion\log\cd_transport.log"/>
  </Logger>
</Logging>
<Senders>
  <Sender Type="Local" Class="com.tridion.transport.LocalCopySender"/>
</Senders>
</TransportService>
```

The following sections provide more detailed descriptions of the different elements and attributes that you can configure within the configuration file.

*See also* Appendix C "Sample Transport Service configuration file" on page 163 provides a sample Transport Service configuration file.

**Note** You can also configure the Java Virtual Machine that the Tridion Transport Service starts. See Appendix J, "Configuring the Java Virtual Machine for COM services" for details.

### 7.1.a WorkFolder

The `WorkFolder` element specifies where the Sender looks for the Transport Package. The Transport Package is a zipped archive of the published content, and is created, and placed in the Work folder by the publisher.

### 7.1.b Logging

The `Logging` element specifies the behavior and location of one or more logging streams. Level attribute specifies the amount of information being logged. The logging levels are:

- `info` — Information messages.
- `warning` — Potential error conditions.
- `error` — Errors that prevent the software from functioning correctly.
- `fatal` — Unrecoverable errors.

The `FileLogger` logs to the log file specified by the `Location` attribute:

```
<FileLogger Level="info" Location="C:\tridion\log\cd_transport.log"/>
```

You can also print log messages to a console using the `ConsoleLogger` element.

### 7.1.c Senders

The `Senders` element defines the Sender types you intend to use. Senders are configured in the Management System interface as Publication Target Destinations.

Specify custom Senders by providing a `Type` that matches the root element name of a Management System Protocol Schema. The `Class` attribute specifies the Java class that implements the functionality for a Sender.

For example:

```
<Sender Type="FTP" Class="com.tridion.transport.FTPSender"/>
```

Make sure the class is registered on the system `CLASSPATH` environment variable.

The following are the available Sender types, and the classes responsible for implementing the respective protocols:

- `Local` — `com.tridion.transport.LocalCopySender`
- `FTP` — `com.tridion.transport.FTPSender`
- `SFTP` — `com.tridion.transport.SFTPSender`
- `HTTP(S)` — `com.tridion.transport.HTTPSSender`

For more information about Senders and Receivers, refer to Chapter 6 "*Senders and Receivers*" on page 29.

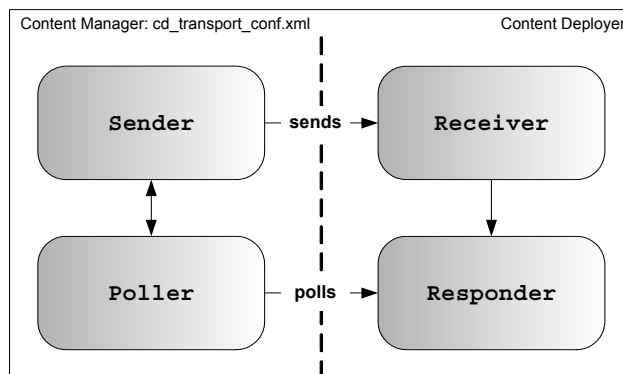


### 7.1.d Poller

The `Poller` element is an optional sub-element of the `Sender` element. The default `Sender` configuration polls every 4 seconds for one hour after a publish action (with no poller configured). You are not required to modify the default configuration.

When published content is transported to the Receiver on the Content Distributor, the `Sender` creates a poller:

- If the Receiver receives and deploys content correctly, it creates a responder. The responder takes the transaction ID of the transported content and writes a file called *TransactionID.xml* to the content deployer's working directory. TransactionID is an alpha-numeric identification of the transported content. The poller looks for the *TransactionID.xml* file over the predefined transport protocol, FTP, SFTP, or HTTP(S) by polling the responder for this file.
- If the deployment or transport fail, the poller reports the failure to the transport service, which reports the failure to the Content Manager.



**Figure 7-2 Poller and Responder**

If you want to change how frequently a poller polls or the polling interval, you can configure two name-value pairs on the poller element:

- `MaxAttempts` — The maximum number of times to poll for a certain transaction. "-1" is no maximum, or poll indefinitely.
- `Interval` — The time interval at which to poll in milliseconds.

If you use Tridion's built-in Local Copy Sender, you can also specify an alternative polling location in the `Location` attribute. If omitted, the location defaults to the deployment path.

Table 7-1 provides example polling configurations for FTP, HTTP(S) and Local Copy.



**Table 7-1 Pollers for FTP and HTTPs**

Protocol	Example
FTP	<pre> &lt;Senders&gt;   &lt;Sender Type="FTP"     Class="com.Tridion.transport.FTPSender"&gt;     &lt;Poller&gt;       &lt;Param Name="MaxAttempts" Value="10"/&gt;       &lt;Param Name="Interval" Value="5000"/&gt;     &lt;/Poller&gt;   &lt;/Sender&gt; &lt;/Senders&gt; </pre>
HTTP(S))	<pre> &lt;Senders&gt;   &lt;Sender Type="HTTP"     Class="com.Tridion.transport.HTTPSender"&gt;     &lt;Poller&gt;       &lt;Param Name="MaxAttempts" Value="10"/&gt;       &lt;Param Name="Interval" Value="5000"/&gt;     &lt;/Poller&gt;   &lt;/Sender&gt; &lt;/Senders&gt; </pre>
LocalCopy <sup>a</sup>	<pre> &lt;Senders&gt;   &lt;Sender Type="Local"     Class="com.Tridion.transport.LocalCopySender"&gt;     &lt;Poller Location="c:\MyPollingLocation\"&gt;       &lt;Param Name="MaxAttempts" Value="10"/&gt;       &lt;Param Name="Interval" Value="5000"/&gt;     &lt;/Poller&gt;   &lt;/Sender&gt; &lt;/Senders&gt; </pre>

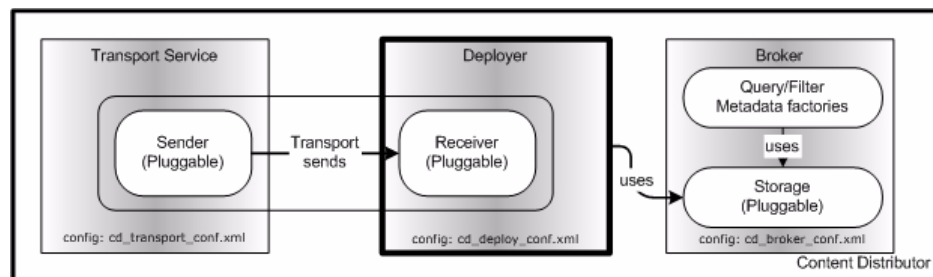
a. The Location attribute of the Poller element is optional.





# Chapter 8 Deployer configuration

The Deployer handles Transport Packages and is responsible for the Receiver's configuration.



**Figure 8-1 Content Deployer**

The Receiver accepts the Transport Package from the Sender, but does nothing with the package except pass it to the Deployer.

Processors in the Deployer deploy or undeploy special types of content using modules that carry out the instructions. Content Deployer modules delegate the content to handlers within the Content Broker. The Processors and Modules are defined in the Deployer configuration file.

### 8.1 Configuring the Deployer

The `Deployer` configuration specifies all configuration values required to retrieve and process Transport Packages received from the Transport Service. Using the Deployer configuration file, `cd_deploy_conf.xml`, you can configure the following elements:

- `WorkFolder` — Identifies the location in which temporary files are stored (Uncompressed Transport Packages, Transaction information, Temporary processor files)
- `Logging` — The behaviour and location of one or more logging streams
- `Processors` — The actions that the Deployer can perform.
- `Transformer` — Enables you to publish content to ASP.NET
- `Receiver Bindings` — The types of Receivers that the Deployer can use.
- `Receivers` — Specifies which Receiver instances the Deployer starts.

The configuration of this file is validated against `cd_deploy_conf.xsd`.

The following example describes the configuration of the deployer for the following:

- `WorkFolder location` — `C:\Tridion\work`  
`Cleanup` — set to `True`, all files are deleted when the Transport Package processing is complete.
- `Logging level` — `error`  
`FileLogger` — log file and path specified
- `Processor actions`, and their related classes — actions such as `Deploy for Pages`, `Components`, and so on.

- Receiver bindings — Type of Receiver and the class responsible for receiving on that protocol.
- Receivers — Receiver configuration.

```

<?xml version="1.0" encoding="UTF-8"?>
<Deployer>
  <WorkFolder Location="C:\Tridion\work" Cleanup="true"/>
  <Logging>
    <Logger Level="error">
      <FileLogger Location="C:\Tridion\log\cd_deployer.log"/>
    </Logger>
  </Logging>
  <Processors>
    <Processor Action="Deploy" Class="com.tridion.deployer.Processor">
      <Module Type="PageDeploy"
        Class="com.tridion.deployer.modules.PageDeploy"/>
      <Module Type="BinaryDeploy"
        Class="com.tridion.deployer.modules.BinaryDeploy"/>
      <Module Type="ComponentDeploy"
        Class="com.tridion.deployer.modules.ComponentDeploy"/>
      <Module Type="TemplateDeploy"
        Class="com.tridion.deployer.modules.TemplateDeploy"/>
      <Module Type="SchemaDeploy"
        Class="com.tridion.deployer.modules.SchemaDeploy"/>
      <Module Type="ComponentPresentationDeploy"
        Class="com.tridion.deployer.modules.ComponentPresentationDeploy"/>
    </Processor>
    <Processor Action="Undeploy"
      Class="com.tridion.deployer.Processor">
      <Module Type="PageUndeploy"
        Class="com.tridion.deployer.modules.PageUndeploy"/>
      <Module Type="ComponentPresentationUndeploy"
        Class="com.tridion.deployer.modules.PageUndeploy"/>
    </Processor>
  </Processors>
  <ReceiverBindings>
    <Receiver Type="HTTPSReceiver"
      Class="com.tridion.transport.HTTPSReceiver"/>
    <Receiver Type="FileReceiver"
      Class="com.tridion.transport.FileReceiver"/>
  </ReceiverBindings>
  <Receivers>
    <FileReceiver Location="C:\Tridion\incoming" Interval="1"/>
    <!--<HTTPSReceiver MaxSize="10000000"/>-->
  </Receivers>
</Deployer>

```

*See also* Appendix D "*Sample Deployer configuration file*" on page 164 provides a more detailed sample Deployer configuration file.

**Note** You can also configure the Java Virtual Machine that the Tridion Content Deployer service starts. See Appendix J, "*Configuring the Java Virtual Machine for COM services*" for details.

The following sections describe the different deployer elements and attributes that you can configure:

- WorkFolder
- Logging
- Processors
- Receiver Bindings
- Receivers

### 8.2 WorkFolder

The WorkFolder stores temporary files, including:

- Uncompressed Transport Packages
- Transaction information
- Temporary processor files

You can empty the WorkFolder when the Transport Package is finished processing by setting the `Cleanup` attribute to `"true"`. For example:

```
<WorkFolder Location="C:\Tridion\work" Cleanup="true"/>
```

If you do not want to empty the folder, set clean up to `"false"`.

### 8.3 Logging

The `Logging` element specifies the behavior and location of one or more logging streams. The `Level` attribute specifies the type of information that is logged:

- `Info` — Information messages
- `Warning` — Potential error conditions
- `Error` — Errors that prevent the software from functioning correctly
- `Fatal` — Unrecoverable errors

The `Location` attribute determines the location of the log file:

```
<FileLogger Location="c:\Tridion\log\cd_deployer.log"/>
```

You can also print log messages to a console using the `ConsoleLogger` element.

## 8.4 Processors

The `Processors` element defines the actions that the Deployer can perform. The behavior of the deployer is defined by:

- the type of Processors
- the specified modules
- custom Deployer behavior.

The `Processors` element can have the following attributes:

- `Action` — Determines how the Deployer triggers the Processor to process an incoming Transport Package.
- `Class` — Defines the Processor class that is used to process the action.

The default Processor is:

```
<Processor Action="Deploy" Class="com.tridion.deployer.Processor">
```

You should not have to change the configuration of this file. However, if you have written your own custom Processor, you can add it within the `Processors` element. The Deployer will call the Processor elements in the order in which they appear in the configuration file.

### 8.4.a Developing custom processors

To develop a custom processor, you must write a Processor object that extends `com.tridion.deployer.Processor`. The new object you create will override the default methods and will provide methods that are required by the custom implementation.

For example, if you override `process(TransportPackage data)`, a custom processor can perform different or additional actions than the standard processor on incoming Transport Packages.

To add extra processors to the calling chain, you must modify `cd_deployer_conf.xml`. Add extra `processor` elements under the `Processors` element with an `action` attribute that matches the action for which the processor is registered.

**Example** In the following example, for the `Deploy` action, the `Processors` configuration first uses the standard Tridion Deployment processor and then uses a custom Processor class (`mypackage.MyProcessor`):



```

<Processors>

<Processor Action="Deploy" Class="com.tridion.deployer.Processor">
  <Module Type="PageDeploy"
  Class="com.tridion.deployer.modules.PageDeploy"/>
  <Module Type="BinaryDeploy"
  Class="com.tridion.deployer.modules.BinaryDeploy"/>
  <Module Type="ComponentDeploy"
  Class="com.tridion.deployer.modules.ComponentDeploy"/>
</Processor>

<Processor Action="Deploy" Class="mypackage.MyProcessor">
  <!--custom configuration for this processor can go here -->
</Processor>

</Processors>

```

The Processors are called in the order in which they included in the xml file. In this example, the `process()` method is called on `com.tridion.deployer.Processor` first, and then is called on `mypackage.MyProcessor`.

If a `ProcessingException` is thrown by the `process()` method, all further processing is aborted, and other Processors in the chain are not called.

## 8.5 Module

The `Module` element defines modules that are used to delegate content to handlers in the Content Broker. The `Module` is triggered by a Processor to process incoming instructions. The `Module` element can have the following attributes:

- The `Type` attribute must be unique and serves as a symbolic identifier. For example, "PageDeploy"
- The `Class` attribute defines the implementation used for any type of module. For example, "com.tridion.deployer.modules.PageDeploy"

Table 8-1 describes the default Deployer modules.

**Table 8-1 Actions & modules. (Sheet 1 of 2)**

Action	Module
PageDeploy	com.tridion.Deployer.modules.PageDeploy
BinaryDeploy	com.tridion.Deployer.modules.BinaryDeploy
ComponentDeploy	com.tridion.Deployer.modules.ComponentDeploy
TemplateDeploy	com.tridion.Deployer.modules.TemplateDeploy
SchemaDeploy	com.tridion.Deployer.modules.SchemaDeploy



**Table 8-1 Actions & modules. (Sheet 2 of 2)**

Action	Module
ComponentPresentation Deploy	com.tridion.Deployer.modules.Component PresentationDeploy
PageUndeploy	com.tridion.Deployer.modules.Page Undeploy
ComponentPresentation Undeploy	com.tridion.Deployer.modules.Component PresentationUndeploy

### 8.5.a Developing custom Deployer modules

You can customize and extend the Deployer by writing custom Deployer modules in Java. These modules can extend the default modules to implement any desired functionality using the Transport Package data-structures.

A custom module is essentially a Java class that is loaded and called by the Deployer whenever a Transport Package is being processed.

For example, some application servers need to flush their caches whenever new content is deployed. The following class demonstrates this:



## Chapter 8 Deployer configuration

```
package com.tridion.examples;

import com.tridion.transport.transportpackage.*;
import com.tridion.configuration.*;
import com.tridion.deployer.ProcessingException;
import com.tridion.deployer.Processor;
import com.tridion.deployer.Module;

import com.perfectsoftware.AppServer.*;

public class CacheFlusher extends Module {
    //imaginary appserver API
    AppManager server = CachingAppServer.getManagerInstance();

    public CacheFlusher(Configuration config, Processor processor)
    throws ConfigurationException {
        super(config, processor);
    }

    // This method is called once for each TransportPackage that is
    // deployed.
    public void process(TransportPackage data) throws
    ProcessingException {
        try {
            server.flush(config.getStringValue("AppInstance"));
        } catch (ConfigurationException e) {
            log.error("Could not get custom configuration", e);
        }
    }
}
```

Once this class is compiled and available to the Deployer it can be configured in the `cd_deployer_conf.xml` with the following elements:

```
<Processor Action="Deploy" Class="com.tridion.Deployer.Processor">
  <Module Type="PageDeploy"
    Class="com.tridion.Deployer.modules.PageDeploy"/>
  <Module Type="BinaryDeploy"
    Class="com.tridion.Deployer.modules.BinaryDeploy"/>
  <Module Type="ComponentDeploy"
    Class="com.tridion.Deployer.modules.ComponentDeploy"/>
  <Module Type="TemplateDeploy"
    Class="com.tridion.Deployer.modules.TemplateDeploy"/>
  <Module Type="SchemaDeploy"
    Class="com.tridion.Deployer.modules.SchemaDeploy"/>
  <Module Type="ComponentPresentationDeploy"
    Class="com.tridion.Deployer.modules.ComponentPresentationDeploy"/>
  <Module Type="CacheFlusher"
    Class="com.tridion.examples.CacheFlusher">
  <AppInstance>default</AppInstance>
</Module>
</Processor>
```

The custom module is configured as the last module for the Processor that handles a deployment action. As a result, it is called after all other modules have finished processing. It is possible to use custom configuration for a module by adding XML elements to the Module configuration. This configuration information is passed to the custom Module constructor.

### 8.5.b Modifying the behavior of default modules

The default modules behavior can be modified using inheritance. In the following example, a number of custom keywords are added to the Component metadata.



```
package com.tridion.examples;

import com.tridion.deployer.Processor;
import com.tridion.deployer.ProcessingException;
import com.tridion.configuration.*;
import com.tridion.transport.transportpackage.*;

public class ExtendedComponentDeploy extends
com.tridion.deployer.modules.ComponentDeploy {

    public ExtendedComponentDeploy(Configuration config, Processor
processor) throws ConfigurationException {
        super(config, processor);
    }

    protected void processComponent(Component component) throws
ProcessingException {
        Category category = component.getCategory("Custom");
        if (category == null) {
            //create a new Category if it does not exist yet
            category = new Category("Custom");
        }
        // Add a keyword
        category.addKeyWord("Custom Keyword");
        // Add or update the category
        component.addCategory(category);
        super.processComponent(component);
    }
}
```

This extended module is used by the Deployer, if it is configured in the `cd_deployer_conf.xml` configuration file.

```
<Module Type="ComponentDeploy"
Class="com.tridion.examples.ExtendedComponentDeploy"/>
```

### 8.5.c Testing custom modules

To test any custom modules you can intercept Transport Packages by using the FTP transport method while the Deployer is not active.

You can copy these packages to a separate location on the file system and into the folder that is monitored by a running Deployer that has the custom module configured in its Processors configuration section.

## 8.6 Transformer

The optional `Transformer` element can be used if you want to publish content to ASP.NET.

Unlike other Output formats, the Content Manager does not transform the content to the published format. Instead, the Content Manager transports the content in an intermediate format called TCDL. The Content Deployer then transforms this format in to ASP.NET.

Use the optional `Transformer` element can be used if you want to publish content to ASP.NET. Add the `Transformer` element as a sub-element to the `PageDeploy` module. The `Transformer` element specifies the following `Class` attribute:

```
Class="com.tridion.transformer.ASPDotNETTransformer"
```

The following example depicts this element used for `PageDeploy`:

```
<Processor Action="Deploy" Class="com.tridion.deployer.Processor">
  <Module Type="PageDeploy"
    Class="com.tridion.deployer.modules.PageDeploy">
    <Transformer Class="com.tridion.transformer.ASPDotNETTransformer"
      />
  </Module>
  <Module Type="SchemaDeploy"
    Class="com.tridion.deployer.modules.SchemaDeploy"/>
  <Module Type="ComponentPresentationDeploy"
    Class="com.tridion.deployer.modules.ComponentPresentationDeploy">
  </Module>
  <Module Type="BinaryDeploy"
    Class="com.tridion.deployer.modules.BinaryDeploy"/>
  <Module Type="ComponentDeploy"
    Class="com.tridion.deployer.modules.ComponentDeploy"/>
  <Module Type="TemplateDeploy"
    Class="com.tridion.deployer.modules.TemplateDeploy"/>
</Processor>
```

When publishing to ASP.NET, the following conditions apply:

- The Deployer is configured to use the `ASPDotNETTransformer`
- The Publication Target to which content is published specifies the TCDL (Tridion Content Deployer Language) as the output language
- The Page Template file extension is `.aspx`
- The ASP.NET libraries are installed on the presentation server



## 8.7 ReceiverBindings

The `ReceiverBindings` element defines the types of Receivers that the Deployer can use. The `ReceiverBindings` element can have the following attributes:

- `Type` — identifies the Receiver  
For example, "HTTPSReceiver"
- `Class` — defines the implementation used for receiving the Transport Packages  
For example, `com.tridion.transport.HTTPSReceiver`

## 8.8 Receivers

The `Receivers` element specifies which Receiver instances the Deployer starts. The element names for these Receivers must match the `Type` attribute defined in the `ReceiverBindings` section.

Depending upon the protocol that you use, you must configure your Web or Application server to handle the incoming Transport Packages. This section describes the setup required for the various supported protocols:

- Local Copy
- FTP
- SFTP
- HTTP(s)

Tridion supplies several protocol-specific Sender-Receiver pairs with the Content Distributor installation:

- Local Copy — The LocalCopy Sender transfers the Transport Package to a corresponding File Receiver on the Deployment server.
- FTP — The FTP Sender uses File Transfer Protocol to transfer the Transport Packages to a corresponding File Receiver on the Deployment server.
- SFTP — The SFTP Sender uses Secure File Transfer Protocol to transfer the Transport Packages to a corresponding File Receiver on the Deployment server.
- HTTP(S) — The HTTP(S) Sender allows you to use HTTP, or HTTPS to encrypt the content of your Transport Package, for transport to a corresponding HTTP(S) Receiver on the Deployment server.

The Transport Service configuration and Deployer configuration must include the Sender-Receiver pair that your organization wants to use.

**Table 8-2 Protocols and their Receivers.**

Protocol	Class name	Description
Local Copy	<code>com.tridion.transport.FileReceiver</code>	Receives packages copied over to the local file system.
FTP	<code>com.tridion.transport.FileReceiver</code>	Receives packages that an FTP server places on the file system.
SFTP	<code>com.tridion.transport.FileReceiver</code>	Receives packages that an SFTP server places on the file system.
HTTP(S)	<code>com.tridion.transport.HTTPSReceiver</code>	Receives HTTP(S) packages.

*See also* For more information about Senders and Receivers, refer to Chapter 6 "*Senders and Receivers*" on page 29.







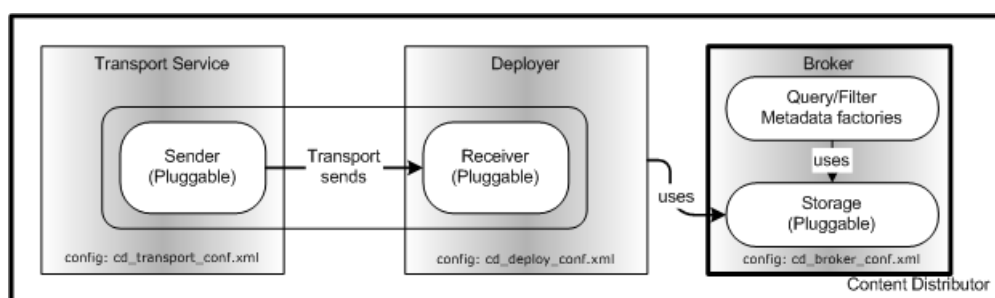
# Chapter 9 Broker configuration

The handlers within the Content Broker determine where different types of content are stored. The Content Distributor supports the following storage media:

- MS SQL
- Oracle
- Tamino
- IBM DB2
- File system

The handlers also keep track of what uses what using reference counting. Reference counting ensures that unused content is undeployed.

Based on the interface that a handler talks to, content can be stored in a SQL database (MS SQL, Oracle, DB2), a file system, or a Tamino database. The interfaces have specific methods and classes. Broker bindings bind a symbolic name (such as FTP Sender) to an implementation class of an interface. It is possible to use these different storage media next to each other for different types of content.



**Figure 9-1 Content Broker**

The Content Broker can also perform filters on the stored content. Filters use Component metadata to select a subset of Components, or Component Presentations, based on a set of criteria, such as IDs or priorities. You can use filters to, for example, personalize content or to return content based on search results. For more information on metadata and filters, see Chapter 10 "Using metadata" on page 73.

## 9.1 Configuring the Content Broker

The configuration file for the Content Broker defines the following:

- Global settings apply to all areas of Content Broker functionality and include the following:
  - Logging settings specify the behavior and location of logging.
  - Object Caching settings
  - Storage configuration settings
  - Database settings
  - Bindings settings
- Publications settings
  - Directory settings
  - Publication-specific settings

The following example describes the configuration of the Broker:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration Version="5.1">
  <Global>
    <Logging>
      <Logger>
        <FileLogger Level="info" Location="./cd_broker.log"/>
      </Logger>
    </Logging>
    <Storage>
      <Database Type="sql" Username="USERNAME" Password="PASSWORD"
        Url="jdbc:oracle:thin:@DATABASE_HOST:PORT:SID"
        Driver="oracle.jdbc.driver.OracleDriver">
      <Pool Type="jdbc" Size="10"/>
    </Database>
  </Storage>
  <Bindings>
    <Binding Name="LinkInfo"
      Class="com.tridion.broker.linking.SQLLinkInfoHome"/>
    <Binding Name="PageMeta"
      Class="com.tridion.broker.pages.meta.SQLPageMetaHome"/>
    <Binding Name="ComponentMeta"
      Class="com.tridion.broker.components.meta.OracleComponentMetaHome"
      />
    <Binding Name="BinaryMeta"
      Class="com.tridion.broker.binaries.meta.SQLBinaryMetaHome"/>
    <Binding Name="Reference"
      Class="com.tridion.broker.references.SQLReferenceHome"/>
    <Binding Name="XSLT" Class="com.tridion.broker.xslt.FSXSLTHome"/>
    <Binding Name="Schema"
      Class="com.tridion.broker.schemas.FSSchemaHome"/>
    <Binding Name="ASPCOMPONENTPRESENTATION"
      Class="com.tridion.broker.componentpresentations.FSASPCOMPONENTPRESENTATIONHOME"/>
    <Binding Name="JSPCOMPONENTPRESENTATION"
      Class="com.tridion.broker.componentpresentations.FSJSPCOMPONENTPRESENTATIONHOME"/>
  </Bindings>
</Configuration>
```

```

<Binding Name="XMLComponentPresentation"
  Class="com.tridion.broker.componentpresentations.OracleXMLComponen
  tPresentationHome"/>
<Binding Name="TextComponentPresentation"
  Class="com.tridion.broker.componentpresentations.OracleTextCompone
  ntPresentationHome"/>
<Binding Name="ComponentPresentationMeta"
  Class="com.tridion.broker.componentpresentations.meta.SQLComponent
  PresentationMetaHome"/>
</Bindings>
</Global>
<Publications DefaultRootLocation="c:/inetpub/wwwroot">
<Publication Id="1" DataRoot="c:/inetpub/wwwroot/pub1"
  DocumentRoot="c:/inetpub/data/pub1">
</Publication>
</Publications>
</Configuration>

```

*See also* Appendix E "Sample Broker configuration file" on page 166 provides a sample Deployer configuration file.

**Note** You can also configure the Java Virtual Machine that the Tridion Content Broker service starts. See Appendix J, "Configuring the Java Virtual Machine for COM services" for details.

The following sections describe the different broker elements and attributes that you can configure:

- Logging
- Object caching
- Storage configuration
- Bindings
- Publications

## 9.2 Logging

The `Logging` element specifies the behavior and location of one or more logging streams. The `Level` attribute specifies the amount of information being logged. The logging levels are:

- `info` — Information messages.
- `warning` — Potential error conditions.
- `error` — Errors that prevent the software from functioning correctly.
- `fatal` — Unrecoverable errors.

The `FileLogger` logs to the log file specified by the `Location` attribute:

```
<FileLogger Level="info" Location="C:\tridion\log\cd_broker.log"/>
```

You can also print log messages to a console using the `ConsoleLogger` element.

An administrator should make sure that the account used by Tridion Linking has rights to write to the log file, and create directories if needed.

## 9.3 Object caching

Object caching is an optional setting that allows you to store the most commonly used, or resource intensive objects in a cache. The cache keeps these objects available for the applications that request them, rather than reinitializing them each time they are requested.

The object caching mechanism allows you to specify the objects to cache. The object cache supports multiple Java Virtual Machine (JVM) instances using Remote Method Invocation (RMI). This allows for inter-JVM communication.

You can configure the following elements in the `cd_broker_conf.xml`:

- `ObjectCache` — Determines whether caching will occur.
- `Policy` — Determines what policy the cache will use to determine what objects stay in the cache and what objects are removed from the cache when it is full.

### Example structure

```
<ObjectCache Enabled="true">

  <Policy Type="LRU" Class="com.tridion.cache.LRUPolicy">
    <Param Name="Size" Value="64"/>
  </Policy>

  <Features>
    <Feature Type="DependencyTracker"
      Class="com.tridion.cache.DependencyTracker"/>
  </Features>

  <CacheBindings>
    <CacheBinding Name="BinaryMeta"
      Class="com.tridion.broker.binaries.meta.CachedBinaryMetaHome"/>
  </CacheBindings>

  <RemoteSynchronization Host="localhost" Port="3333" Queuesize="20"/>

</ObjectCache>
```

### 9.3.a ObjectCache

The `ObjectCache` element is an optional element of the `Global` element of the Broker configuration files.

If this element is not present, no caching is performed.

```
<Global>
  <Logging>
    <Logger>
      <FileLogger Level="info" Location="./cd_broker.log"/>
    </Logger>
  </Logging>
  <ObjectCache Enabled="true">
```



The `ObjectCache` has an `Enabled` attribute:

- If `Enabled="true"`, object caching is performed using the `ObjectCache` elements.
- If `Enabled="false"`, object caching is not performed and any `ObjectCache` elements are ignored.

### 9.3.b Policy

The `Policy` element determines what policy the cache uses to determine what objects stay in the cache and what objects are removed from the cache when it is full.

The `Policy` element has the following mandatory attributes:

- `Type` — The Least Recently Used (LRU) policy governs how objects are cached. When the cache reaches its maximum size, the least recently used objects are removed from the cache.
- `Class` — The `Class` attribute must specify the fully qualified class name of a class that extends `com.tridion.cache.Policy` and implements the `CacheProcessor` interface. The only policy class currently supported is `com.tridion.cache.LRUPolicy`

The `Policy` element also has an optional `Param` element that supplies additional policy specification configuration values. The `Param` element contains the following attributes:

- `Name` — The name of the configuration parameter
- `Value` — The value of the configuration parameter

For example, the `LRUPolicy` has one `Param` called `Size` and a `Value` attribute that represents the maximum number of objects that the cache can contain.

```
<Policy Type="LRU" Class="com.tridion.cache.LRUPolicy">
  <Param Name="Size" Value="64"/>
</Policy>
```

### 9.3.c Features

The `Features` element specifies classes that add additional functionality to the cache.

The `Features` element is optional within the `ObjectCache` element. It may contain zero or more `Feature` elements.

For example:

```
<Features>
  <Feature Type="DependencyTracker"
    Class="com.tridion.cache.DependencyTracker"/>
</Features>
```

The `Features` element can also contain a `Param` element that can be used to supply additional feature-specific name-value pairs. The name and value attributes must be called `Name` and `Value`, but can contain any values required.

### 9.3.d CacheBindings

The optional `CacheBindings` element specifies whether objects that are managed from a specific home object are cached. `CacheBindings` may contain zero or more `CacheBinding` elements.

`CacheBinding` elements must contain the following attributes:

- **Name** — The binding name of the home object add caching to. This must match the `Name` attribute used in the `Bindings` section of the configuration file.
- **Class** — The fully qualified class name of the cached home object.

The following example depicts a `CacheBinding` for binary metadata:

```
<CacheBindings>
<CacheBinding Name="BinaryMeta"
Class="com.tridion.broker.binaries.meta.CachedBinaryMetaHome"/>
</CacheBindings>
```

Table 9-1 describes the Home object classes that implement caching.

**Table 9-1 CacheBinding attributes**

Type	Name attribute value	Class attribute value
Linking	LinkInfo	com.tridion.broker.linking.CachedLinkInfoHome
Content Distributor	ComponentPresentation	componentpresentations.CachedComponentPresentationHome
	ComponentPresentationMeta	componentpresentations.meta.CachedComponentPresentationMetaHome
	PageMeta	pages.meta.CachedPageMetaHome
	XSLT	xslt.CachedXSLTHome
WAI	CustomerCharacteristic	personalization.CachedCustomerCharacteristicHome
	TrackingKey	personalization.CachedTrackingKeyHome
	Timeframe	timeframes.CachedTimeframeHome
	TrackedComponentLink	tracking.componentlinks.CachedTrackedComponentLinkHome
	TrackedComponent	tracking.components.CachedTrackedComponentHome
	TrackedPage	tracking.pages.CachedTrackedPageHome
	User	user.CachedUserHome

### 9.3.e RemoteSynchronization

The optional `RemoteSynchronization` element specifies a remote Cache Channel Service. The remote Cache Channel Service is used to send messages between caches that are running on separate virtual machines. (For example, if the Broker and the Deployer run on separate virtual machines.)

The Cache Channel Service must be running and listening on the configured host and port for remote synchronization to function.

If this element is omitted, the cache does not use any inter-virtual machine cache communication.

You must use the `RemoteSynchronization` element in conjunction with the Cache Channel service in order for objects to be updated or removed from the Broker's object cache when they are published or unpublished using the Deployer.

If the `RemoteSynchronization` element is present, the cache subscribes to an event channel (see "*Cache Channel Service*" on page 67).

The `RemoteSynchronization` element can contain the following attributes:

- `Host` — The hostname of the machine running the remote cache channel service. If `Host` is omitted, `localhost` is assumed as the hostname.
- `Port` — The port number the remote cache channel service listens on. If this is omitted, the default RMI port, 1099, is assumed.
- `Queuesize` — The size of this cache's event queue. If this is omitted then a maximum size of 128 events is assumed.

```
<RemoteSynchronization Host="localhost" Port="3333" Queuesize="20"/>
```

### 9.3.f Cache Channel Service

The remote Cache Channel Service is run as a separate application. For example, as a Windows service or as a standalone application.

This section describes:

- Installing the Cache Channel service on Windows
- Installing the Cache Channel service on Unix

#### To install the Cache Channel Service on Windows platforms:

- 1 Copy the `cd_cacheservice.exe` from your installation CD to the server.
- 2 Type the following command in the Command Prompt:

```
cd_cacheservice.exe -install
```

this installs the `CacheChannelService` as a Windows service on the `localhost`, listening on the default RMI port, 1099.



**To install the Cache Channel Service on Windows platforms:**

- 3** An additional option is to provide the host and port when installing the service using the `-host=` parameter as follows:

```
cd_cacheservice.exe -install -host=127.0.0.1:3030
```

which installs the Cache Channel Service as a Windows service on the localhost, listening on port 3030. If this value is set, it is automatically stored in the Windows registry using the following key:

```
HKEY_CLASSES_ROOT\AppID\{7337B68B-39E0-47FD-8E4D-17907B4C54C8}
}
```

with the value: Host

The service can be removed by entering the following at the command prompt:

```
cd_cacheservice.exe -remove
```

This also removes the Windows registry entries for the service.

**Note** You can also configure the Java Virtual Machine that the Tridion Cache Service starts. See Appendix J, "Configuring the Java Virtual Machine for COM services" for details.

**To install the cache channel service on Unix:**

```
#!/bin/sh
java -cp ./lib/cd_core.jar com.tridion.cache.CacheChannelService
```

You can configure this service using a single command line parameter of the form `hostname:port`

- `hostname` — the name of the host on which this cache channel service should run
- `port` — (optional) the port number on which this cache channel service should run

If this part of the parameter is excluded the default RMI port of 1099 is used. If no command line argument is passed then the service will run on the localhost, listening on port 1099.

## 9.4 Storage configuration

The following databases are supported:

- Microsoft SQL server
- Oracle
- Tamino
- IBM DB2

The Broker configuration file for storage defines the following:

- `Logging` — Log level, log location, and so on.
- `Bindings` — The Java classes used for each area of database functionality, such as User, Linking, and so on.



Database elements configure settings for a specific database. Multiple Database elements can be used. Each Database element has a *type*, and only one element can be specified for a certain type. Because of this, you can specify a maximum of two Database elements.

The following attributes can be specified:

- *Type* — The type of the database, this is *sql* for MS-SQL, Oracle, and DB2, or *tamino* if you are using Tamino.
- *Username* — The user to be used for database interaction.
- *Password* — The password to be used for database interaction.
- *Url* — The url the driver should connect to.
- *Driver* — The fully qualified classname for the driver to be used.

---

#### Important:

If the type of database is *sql*, use all of the attributes or specify the JNDIName.

If the type of database is *tamino*, specify all attributes with the exception of *Driver*.

---

#### Database connection pooling

For Database elements of type SQL, you can specify pooling with a *Pool* element.

This element has the following attributes:

- *Type* — The type is either of the following:
  - ▶ *jdbc* — a more configurable connection pool manager. If additional connections are attempted, the connection is queued.
 

For *Pool Type="jdbc"*, include the following attributes: *Size*, *MonitorInterval*, *IdleTimeout*, and *CheckOutTimeout*.
  - ▶ *tridion* — only allows you to specify the size of the pool. If additional connections are attempted, an exception is raised.
 

For the *Pool Type="tridion"*, include the *Size* attribute only.
- *Size* — The size of the pool, that is, the number of connections permissible in the pool. This setting is not used by the *jdbc* or *jdbc2* connection pool types.
- *MonitorInterval* — Number of seconds between checks on the pool.
- *IdleTimeout* — Number of seconds a connection can remain idle before it is closed.
- *CheckOutTimeout* — Number of seconds a connection can remain checked out before it is returned to the pool.

The following is an example of the connection pooling configuration:

```
<Database Type="sql" Username="USERNAME" Password="PASSWORD"
Url="jdbc:oracle:thin:@DATABASE_HOST:PORT:SID"
Driver="oracle.jdbc.driver.OracleDriver">
<Pool Type="jdbc" Size="10" MonitorInterval="60" IdleTimeout="120"
CheckoutTimeout="120" />
</Database>
```

## 9.5 Bindings

The `Bindings` element contains mappings from symbolic names to classes in the form of `Binding` elements. A `Binding` element has the following attributes:

- `Name` — The symbolic name for an item that needs to be stored.
- `Class` — The class implementing the interface that is expected for a certain symbolic name.

The following default bindings use the File system only. These bindings can be overridden for different storage types.

```
<Binding Name="ASPComponentPresentation"
Class="com.tridion.broker.componentpresentations.FSASPComponent
PresentationHome"/>
<Binding Name="Binary" Class="com.tridion.broker.binaries.FSBinaryHome"/>
<Binding Name="BinaryMeta"
Class="com.tridion.broker.binaries.meta.XMLFileBinaryMetaHome"/>
<Binding Name="ComponentMeta"
Class="com.tridion.broker.components.meta.XMLFileComponentMeta
Home"/>
<Binding Name="ComponentPresentation"
Class="com.tridion.broker.componentpresentations.FSTextComponent
PresentationHome"/>
<Binding Name="ComponentPresentationMeta"
Class="com.tridion.broker.componentpresentations.meta.XMLFile
ComponentPresentationMetaHome"/>
<Binding Name="JSPComponentPresentation"
Class="com.tridion.broker.componentpresentations.FSJSPComponent
PresentationHome"/>
<Binding Name="LinkInfo"
Class="com.tridion.broker.linking.FSCSVLinkInfoHome"/>
<Binding Name="Page" Class="com.tridion.broker.pages.FSPageHome"/>
<Binding Name="PageMeta"
Class="com.tridion.broker.pages.meta.XMLFilePageMetaHome"/>
<Binding Name="Reference"
Class="com.tridion.broker.references.FileReferenceHome"/>
<Binding Name="Schema" Class="com.tridion.broker.schemas.FSSchemaHome"/>
<Binding Name="Template" Class="com.tridion.broker.xslt.FSXSLTHome"/>
<Binding Name="TextComponentPresentation"
Class="com.tridion.broker.componentpresentations.FSTextComponent
PresentationHome"/>
<Binding Name="XMLComponentPresentation"
Class="com.tridion.broker.componentpresentations.FSXMLComponent
PresentationHome"/>
<Binding Name="XSLT" Class="com.tridion.broker.xslt.FSXSLTHome"/>
```

For information about database specific bindings, see Appendix F "*Broker bindings for MS SQL, Oracle, Tamino and DB2 databases*" on page 171.

For information about WAI bindings, see Appendix G "*WAI-Specific bindings*" on page 173. For more information about the Tridion Web and Application Server Integration, refer to Part "*Web and Application Server Integration*" on page 117.

## 9.6 Publications

The `Publications` element specifies the default setting for all Publications. This information can be overridden in Publication elements.

The `Publications` element includes the following:

- `DefaultRootLocation` — A directory for both pages and binaries. By default, this folder is also used to find the meta-data.
- `Publication` — Publication specific settings. The ID indicates the id of the publication. This should be a number. The following optional attributes override storage settings:
  - ▶ `DataRoot` — The directory in which meta-data should be stored and retrieved on the file system.
  - ▶ `DocumentRoot` — The base-directory in which pages and binaries are located.







# Chapter 10 Using metadata

Metadata is descriptive information that is associated with every Component Presentation, Binary, Component, Page, and link published to the Content Distributor.

Metadata is deployed by the following systems:

- Publisher — Collects the metadata.
- Transport System — Creates and transports the Transport Package, which includes metadata, to your delivery system.
- Content Deployer — Unpacks the Transport Package and passes the metadata to the Content Broker.
- Content Broker — Stores the metadata in the specified storage medium, and makes it accessible to scripted calls.

Metadata is published to the Content Distributor in two forms:

- Mandatory
- Custom

This chapter describes:

- Mandatory metadata
- Custom metadata
- Metadata storage
- Metadata examples
- Filters
- Querying custom metadata

## 10.1 Mandatory metadata

Mandatory metadata is always published. The Content Distributor relies on the fact that the mandatory metadata exists. The mandatory metadata is stored separately from Pages, Component Presentations, and binaries.

This section describes the following types of mandatory metadata:

- Mandatory link metadata
- Mandatory Component Metadata
- Mandatory Page Metadata
- Mandatory Component Presentation Metadata
- Mandatory binary metadata

### 10.1.a Mandatory link metadata

When Component Presentations are published, on Pages, the following metadata is also published:

- List of pages that contain a given Component Presentation — used by Component Linking
- List of Component Presentations that are on a given page — used by tracking & profiling (WAI)

#### Component example

```
<component-link id="4592">
  <page component-template-id="4588" id="4593" position="0"
  template-priority="200" url="/SG/kevin/simpleMETA.jsp"/>
</component-link>
```

#### Page example

```
<page-link id="4593" url="/SG/kevin/simpleMETA.jsp">
  <component id="4592" position="0" template-id="4588"
  template-priority="200"/>
</page-link>
```

### 10.1.b Mandatory Component Metadata

When a Component is published, the following metadata is also published:

- Item information
  - ID
  - Author (login name, such as tridion\egilmore)
  - Title
  - Underlying Component schema
  - Minor Version
  - Major Version

- ▶ Publication ID
  - ▶ Owing Publication ID
- Date and time information
  - ▶ Creation
  - ▶ Modification
  - ▶ Initial publication
  - ▶ Last publish
- Categorization information:
  - ▶ Categories and Keywords — used for profiling and personalization, and searching. These elements are always published. but can be empty if you have not yet assigned Categories and Keywords to a Tridion R5 item, such as a Page, or a Component.

### Component metadata example

The following is an example of Component Metadata, published to the file system:

```
<?xml version="1.0" encoding="UTF-8"?>
<component-meta id="12">
<title>Tridion offers benchmark comparison of Internet strategies</title>
<owning-publication id="20"/>
<version major="2" minor="1"/>
<schema id="99"/>
<created timestamp="2003-07-13T23:58:00"/>
<modified timestamp="2003-07-14T22:58:00"/>
<first-published timestamp="2003-07-15T22:58:00"/>
<last-published timestamp="2003-07-17T22:58:00"/>
<categories>
<category name="News">
<keyword>internet</keyword>
<keyword>company</keyword>
</category>
</categories>
</component>
```

### 10.1.c Mandatory Page Metadata

When a page is published, the following metadata is published:

- Item information
  - ▶ ID
  - ▶ Title
  - ▶ Page template used for publishing the page
  - ▶ Minor Version
  - ▶ Major Version
  - ▶ Publication ID
  - ▶ Owing Publication ID



- Date / time
  - ▶ Creation
  - ▶ Modification
  - ▶ Initial Publication
  - ▶ Last published
- Location information:
  - ▶ Complete file system path of Page
  - ▶ Complete URL of Page (appended after document root URL)
- Categorization information:
  - ▶ Categories and Keywords — used for profiling and personalization, and searching. These elements are always published. but can be empty if you have not yet assigned Categories and Keywords to a Tridion R5 item, such as a Page, or a Component.

### Page metadata example

The following is an example of Page Metadata, published to the file system:

```
<?xml version="1.0" encoding="UTF-8"?>
<page id="12">
<title>Home page</title>
<owning-publication id="1"/>
<version major="2" minor="1"/>
<created timestamp="2003-07-13T23:58:00"/>
<modified timestamp="2003-07-14T22:58:00"/>
<initial-publication timestamp="2003-07-15T22:58:00"/>
<last-publication timestamp="2003-07-17T22:58:00"/>
<categories/>
<path>c:\visitorsWeb\docs\news\index.jsp</path>
<url-path>/news/index.jsp</url-path>
<template id="4594"/>
</page>
```

### 10.1.d Mandatory Component Presentation Metadata

When Component Presentations are published, the following mandatory metadata is published:

- Template used for Component Presentation
  - ▶ ID of template
  - ▶ Priority of template
  - ▶ Output format of template
- Component Information
  - ▶ Id of Component (which is used to retrieve Component metadata)

The metadata described above is used by the Linking module, but is also available through the TOM. This make it possible for Template designers to access component metadata on the Content Distributor, for example:



### Component Presentation metadata example

The following is an example of published Component Presentation Metadata, published to the file system:

```
<?xml version="1.0" encoding="UTF-8"?>
<component-presentation>
<component id="13"/>
<component-template id="43" priority="200" output-format="text/html"/>
</component-presentation>
```

### 10.1.e Mandatory binary metadata

When Binaries are published, the following mandatory metadata is published:

- Binary meta ID
- Path
- Description
- Type
- URL-Path

### Binary metadata example

```
<?xml version="1.0" encoding="UTF-8"?>
<binary-meta id="tcd:pub[20]/component[4613]">
<path>c:\visitorsWeb\docs\images\img_0395s_tcm20-4613.jpg</path>
<description/>
<type>image/jpeg</type>
<url-path>/images/img_0395s_tcm20-4613.jpg</url-path>
</binary-meta>
```

## 10.2 Custom metadata

Custom metadata is defined using metadata schemas on the Content Manager. This information is highly customizable and can be used in a wide variety of ways.

**Note** Currently, Custom metadata is published for Components, *only*.

The Content Broker makes it possible to define queries on custom Component metadata.

### Custom Component metadata example

The following is an example of Custom Component Metadata, published to the file system:

```
<?xml version="1.0" encoding="UTF-8"?>
<component-meta id="12">
<title>Tridion offers benchmark comparison of Internet strategies</title>
<owning-publication id="20"/>
<version major="2" minor="1"/>
<schema id="99"/>
<created timestamp="2003-07-13T23:58:00"/>
```

```

<modified timestamp="2003-07-14T22:58:00"/>
<first-published timestamp="2003-07-15T22:58:00"/>
<last-published timestamp="2003-07-17T22:58:00"/>
<categories>
<category name="News">
<keyword>internet</keyword>
<keyword>company</keyword>
</category>
</categories>
<custom-meta>
<meta>
<embeddate type="dateTime">2003-06-05T12:08:00</embeddate>
<text type="normalizedString">some meta text</text>
<number type="decimal">123456789</number>
<exlink>http://www.msn.com</exlink></meta>
</custom-meta>
</component-meta>

```

**Note** The `type` attribute is added to the Custom metadata only when you are using the File System as your storage medium.

### 10.2.a Custom metadata examples

The following is a fragment of XML metadata from a Component. It contains some basic metadata about the Component and an additional embedded schema, containing more useful information about the Component.

```

<?xml version="1.0" encoding="UTF-8"?>
<metadata>
<author>Niels</author>
<some-embedded-schema>
<country><!--also embedded-->
<code>nl</code>
<language>dutch</language>
<fullname>The Netherlands</fullname>
</country>
<city>Amsterdam</city>
</some-embedded-schema>
</metadata>

```

#### Truncating metadata

Truncation removes all nested elements, and only stores the top-level elements. The example above, is stored as:

```
author=Niels
```

#### Flattening metadata

Flattening removes the hierarchy, and all elements are stored as though they are on the same level. For example:

```
author=Niels
code=nl
language=dutch
fullname=The Netherlands
city=Amsterdam
```



### Prefixing metadata

Prefixing maintains the hierarchy by prefixing each element name, with the parent element name. For example:

```
author=Niels
some-embedded-schema.country.code=nl
some-embedded-schema.country.language=dutch
some-embedded-schema.country.fullname=The Netherlands
some-embedded-schema.city=Amsterdam
```

### 10.2.b Custom metadata fields with multiple values

It is possible to have multiple values in a Custom Metadata field, but these multiple values are not stored separately in SQL, they are concatenated.

For example:

```
<EmbedField>
<TextField>First Value</TextField>
<TextField>Second Value</TextField>
</EmbedField>
```

is transformed to:

```
"TextField"="First Value, Second Value"
```

## 10.3 Metadata storage

The Content Distributor can store the content in a variety of storage media. The following types are supported:

- File System — directory structure
- SQL — MS-SQL 2000, IBM's DB2, and Oracle
- Tamino — Hierarchical XML database

### 10.3.a File system storage

Metadata is stored in the owning Publication's folder of the `data` folder, such as `\data\Pub1`, where 1 is the ID of the owning Publication. Inside this folder, folders are created to hold different types of metadata.

- The folder name for Pages is: `<datadir>/pagemeta/`
- The folder name for LinkInfo is: `<datadir>/linkinfo/`
- The folder name for Binaries is: `<datadir>/binarymeta`
- The folder name for Components is: `<datadir>/componentmeta/`
- The folder name for Component Presentations is: `<datadir>/componentpresentationmeta/`

where *<datadir>* is the data folder of the Publication, such as `c:\visitorsWeb\data\pub1`, which is the data folder for a Publication, with ID of 1.

### 10.3.b Configuring the storage options

To configure the storage option which best suits your implementation, you must add the following lines to the Bindings section of your `cd_broker_conf.xml`:

```
<Bindings>
  <Binding Name="ComponentMeta"
    Class="com.tridion.broker.components.meta.MsSqlComponentMetaHome">
    <Param Name="CustomMetaProcessor"
      Value="com.tridion.broker.components.meta.YourChoiceOfProcessorHere" />
    </Binding>
</Bindings>
```

Depending on your choice of metadata storage format, change the value of the `YourChoiceOfParameterHere` to one of the following values:

- Truncating — `TruncatingCustomMetaProcessor`
- Flattening — `FlatteningCustomMetaProcessor`
- Prefixing — `PrefixingCustomMetaProcessor`

**Note** While metadata can be retrieved from a file system, it is not possible to perform queries on it. Queries are only possible on a database, such as MS-SQL, Oracle, DB2, or Tamino.

For more information about configuring bindings for the Content Broker, refer to 9.5 "Bindings" on page 70.

### 10.3.c Database storage

SQL is used to store the metadata and link information, using either MS SQL or Oracle. The Data Repository is not physically separated for different Publications, which means each table that holds metadata or link information can be used by multiple Publications.

Table 10-1 describes the table structure for metadata.

**Table 10-1 SQL table columns for Component Metadata.**

Column name	Type	Value
ITEM_REFERENCE_ID	number	NOT NULL
PUBLICATION_ID	number	NOT NULL
MAJOR_VERSION	number	NULL
MINOR_VERSION	number	NULL
OWNING_PUBLICATION_ID	number	NULL
ITEM_TYPE	number	NOT NULL
TITLE	string	NOT NULL
CREATION_DATE	datetime	NULL

**Table 10-1 SQL table columns for Component Metadata.**

Column name	Type	Value
INITIAL_PUBLICATION_DATE	datetime	NULL
LAST_PUBLISHED_DATE	datetime	NULL
TRUSTEE	string	NOT NULL
MODIFICATION_DATE	datetime	NULL
FILENAME	string	NULL
SCHEMA_ID	number	NULL
PAGE_TEMPLATE_ID	number	NULL
URL	string	NULL

### Component Presentations

Metadata describing Component Presentations is structured differently, because it is not a defined entity on the Management System. Only during preview or publishing is a Component Presentation created by combining a Component Template with a Component.

### Linking information

Linking to Pages is performed based on the availability of metadata of the Pages. If Page metadata is present, the Page Link is resolved.

Component Links do not use the standard metadata of Components. Instead, this functionality uses `LinkInfo` to determine a target Component Presentation. If a suitable target Component Presentation is found, the Page ID is used to generate a Page Link.

### 10.3.d Storing custom metadata in SQL databases

Custom metadata can be as complex as the designer wants it to be. However, this complexity can present issues on storage. To address this the metadata is altered in the following, configurable, ways:

- Truncating — Removes all metadata beyond a certain level
- Flattening — Puts all metadata on the same level
- Prefixing — Prefixes all nested metadata with the name of the preceding element.

See "*Custom metadata examples*" on page 78 for examples.

The Custom metadata is stored as name-value pairs. For example:

`<name>SenanGilmore</name>` is stored as:

```
name=SenanGilmore
```

---

#### Caution:

This pertains to SQL databases only (MS-SQL, Oracle and DB2). Both Tamino and File System storage media preserve the hierarchy of the information.

---

Table 10-2 describes the table structure for custom metadata:

**Table 10-2 SQL table columns for Custom Metadata.**

Column name	Type	Value
PUBLICATION_ID	number	NOT NULL
ITEM_ID	number	NOT NULL
ITEM_TYPE	number	NOT NULL
KEY_NAME	string	NOT NULL
KEY_TYPE	number	NOT NULL
KEY_DATE_VALUE	datetime	NULL
KEY_STRING_VALUE	string	NULL
KEY_FLOAT_VALUE	number (float)	NULL

## 10.4 Metadata examples

The following examples show the retrieval of metadata for a Component from ASP and from JSP.

### 10.4.a ASP example

The following example shows the retrieval of all metadata for a Component from ASP:

```

<%
Option Explicit

Dim ComponentMetaFactory, ComponentMeta, CustomMeta, TheValue

'Create a ComponentMetaFactory object
Set ComponentMetaFactory =
Server.CreateObject("cd_broker.ComponentMetaFactory")

'Retrieve a ComponentMeta object from the factory
Set ComponentMeta = ComponentMetaFactory.GetComponentMeta("tcm:0-180-1",
"tcm:1-2116")

'Check if the ComponentMeta could be retrieved
If (NOT ComponentMeta Is Nothing) Then
'Print out the ComponentMeta properties
Response.Write "<br/>Id: " & ComponentMeta.Id
Response.Write "<br/>Title: " & ComponentMeta.Title
Response.Write "<br/>ModificationDate: " & ComponentMeta.ModificationDate
Response.Write "<br/>InitialPublicationDate: " &
ComponentMeta.InitialPublicationDate
Response.Write "<br/>LastPublicationDate: " &
ComponentMeta.LastPublicationDate
Response.Write "<br/>CreationDate: " & ComponentMeta.CreationDate
Response.Write "<br/>SchemaId: " & ComponentMeta.SchemaId
Response.Write "<br/>MinorVersion: " & ComponentMeta.MinorVersion

```

```

Response.Write "<br/>MajorVersion: " & ComponentMeta.MajorVersion
Response.Write "<br/>PublicationId: " & ComponentMeta.PublicationId

'Retrieve the CustomMeta from the ComponentMeta
Set CustomMeta = ComponentMeta.CustomMeta

'Check if the CustomMeta could be retrieved
If (NOT CustomMeta Is Nothing) Then
'Get the value for field "test1"
TheValue = CustomMeta.GetValue("test1")
Response.Write "CustomMetaValue for key ""test1"": " & TheValue
End If
End If

'Release the COM objects
Set CustomMeta = Nothing
Set ComponentMeta = Nothing
Set ComponentMetaFactory = Nothing
%>

```

### 10.4.b JSP example

The following example shows the retrieval of all metadata for a Component from JSP:

```

<%@ page import="com.tridion.meta.ComponentMeta,
com.tridion.meta.ComponentMetaFactory,
com.tridion.meta.CustomMeta" %>
<%
//Create a ComponentMetaFactory object
ComponentMetaFactory componentMetaFactory = new
ComponentMetaFactory("tcm:0-180-1");

//Retrieve a ComponentMeta object from the factory
ComponentMeta componentMeta = componentMetaFactory.getMeta("tcm:1-2116");

//Check if the ComponentMeta could be retrieved
if (componentMeta != null) {
//Print out the ComponentMeta properties
out.println("<br/>Id: " + componentMeta.getId());
out.println("<br/>Title: " + componentMeta.getTitle());
out.println("<br/>ModificationDate: " +
componentMeta.getModificationDate());
out.println("<br/>InitialPublicationDate: " +
componentMeta.getInitialPublicationDate());
out.println("<br/>LastPublicationDate: " +
componentMeta.getLastPublicationDate());
out.println("<br/>CreationDate: " + componentMeta.getCreationDate());
out.println("<br/>SchemaId: " + componentMeta.getSchemaId());
out.println("<br/>MinorVersion: " + componentMeta.getMinorVersion());
out.println("<br/>MajorVersion: " + componentMeta.getMajorVersion());
out.println("<br/>PublicationId: " + componentMeta.getPublicationId());

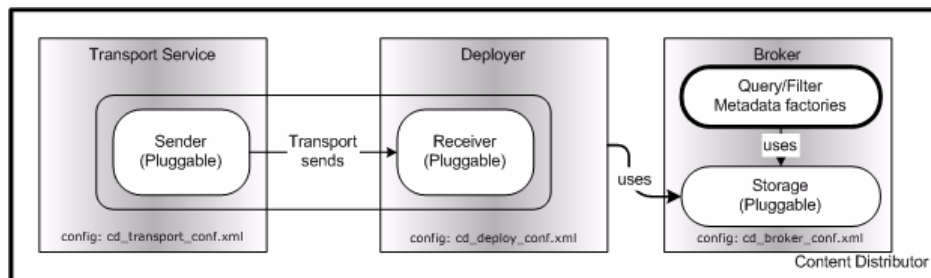
//Retrieve the CustomMeta from the ComponentMeta
CustomMeta customMeta = componentMeta.getCustomMeta();

```

```
//Check if the CustomMeta could be retrieved
if (customMeta != null) {
//Get the value for field "test1"
Object theValue = customMeta.getValue("test1");
out.println("CustomMetaValue for key \"test1\": " + theValue);
}
}
%>
```

## 10.5 Filters

Filters provide a way to select a subset of Components. To query custom metadata from your Content Delivery database you create queries in your templates.



**Figure 10-1 Queries and filters**

You can use filters to select a subset of Components. All filters result in a list of Component IDs (either using numbers, or by objects that encapsulate the IDs together with other information).

You can filter using information about Components, rather than using actual content.

You can use the following types of filters for Dynamic assembly:

- **Search Filter** — A search filter is based on the filter mechanism, but provides the possibility to extend the filter information with user dependant criteria.
- **Related Items Filter** — A filter that lists items related to a specific component, based on keywords.
- **Most Visited Components Filters** — Filters that show a list of most visited Components based on Tracking information. For more information about using the Web and Application Integration to track visitor behavior, refer to Chapter 14 "Profiling and personalization" on page 119.

This chapter describes:

- Accessing a single Component Presentation

### 10.5.a Accessing a single Component Presentation

You can access a single Component Presentation in one of the following ways:

- Supplying a Component ID in combination with a template ID
- Supplying the id that results in the Component with the highest priority



- Supplying the Component id together with a priority, which returns the first match

All these possibilities are shown in the following example:

First create a `ComponentPresentationFactory` for the correct Publication:

```
ComponentPresentationFactory factory = new
ComponentPresentationFactory(12);
```

or using a TCM:URI:

```
ComponentPresentationFactory factory = new
ComponentPresentationFactory("tcm:0-12-1");
```

The factory can then be used to get a `ComponentPresentation`:

```
ComponentPresentation ps = factory.getComponentPresentation(123,55);
ComponentPresentation ps1 =
factory.getComponentPresentationWithHighestPriority(123);
```

or

```
ComponentPresentation ps = factory.getComponentPresentation("tcm:12-123",
"tcm:12-55-32");
```

If a URI is available for the `ComponentPresentation`, the following code can be used:

```
ComponentPresentation ps1 =
factory.getComponentPresentationWithHighestPriority(new
TCMURI("tcm:12-123").getItemId());
}
```

### 10.5.b Search filter

Search needs input from users to build Dynamic filters. It is a normal filter, but takes Strings as parameters.

The mapping of parameter names to search names can be explicit (the Template Designer takes form parameters and uses names that the filter understands) or implicit (the form parameter names are chosen in such a way that translation is not required).

**Note** Search filters should be specified in the Page Template.

A Search filter searches the Component metadata of available Component Presentations for matches based on certain criteria. A standard Component filter is available in the Content Distributor API. A Component filter filters on the following criteria:

- Keywords
- Schema
- Publication date

Filters always return the Component IDs as an array. This array can then be used to call a renderer, which renders all Components based on extra criteria.

The following example shows a search filter, which performs a search for components based on a schema with an ID greater than 1, in the "news" category, containing the keyword "financial", and prints the number of results, and the results themselves.

```
int publicationID=1;
//where 1 is the publication id

SearchFilter filt = new SearchFilter(publicationURI);

Query q = new Query();
q.addCriteria(ComponentMetaHome.FIELD_SCHEMA, ">", 1);
q.addOperator(QueryOperator.OR);
q.addCriteria(ComponentMetaHome.FIELD_CATEGORY, "=", "news");
q.addOperator(QueryOperator.AND);
q.addCriteria(ComponentMetaHome.FIELD_KEYWORD, "=", "financial");

String[] results = filt.match(q.toString(), "title=desc", 20);

out.println("<BR>There are " + results.length + " results<pre>");

// go through the results
for (int i = 0; i < results.length; i++) {
// get the component metadata
ComponentMetaFactory factory = new ComponentMetaFactory(publicationURI);
ComponentMeta meta = factory.getMeta( results[i]);
out.println( "getSchemaURI: "+meta.getSchemaURI());
out.println( "getTitleURI: "+meta.getTitle());
}
```

### 10.5.c Generating Presentation Filters

You can call Content Distributor functionality from either Page Templates or Component Templates. However, it is recommended to only do this from Component Templates, so that the standard page layout used for Pages that do not use filters is maintained. This chapter focuses on using Component Templates for this purpose.

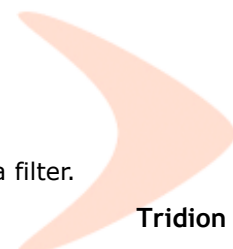
Certain Component Presentations on the Deployment Server need to be assembled on a Page by the Content Distributor when requested. To do so, you use a Placeholder that exists on the Management System and can be added to a Page. When published, the Placeholder is transformed into scripted code that assembles Component Presentations at runtime.

Placeholders are conceptual, rather than a predefined entity on the Management System. Template Designers build Placeholders by using:

- A Schema that specifies fields used as input for a Filter.
- A Component that fills in properties for a Filter.
- A Component Template that uses properties of the Component to generate the code for a Filter on the Content Distributor.

### 10.5.d Result modifiers

It is possible to specify a maximum number of results from a filter.



You can also sort the results of a filter:

- Direction (ascending or descending)
- Field to sort on (for example the title in Component metadata)

## 10.6 Querying custom metadata

To query custom metadata from your Content Delivery database, you must create queries in your templates. These queries can be added to your search filters, thereby extending the possibilities of your searches.

Custom metadata can have the following values:

- String — used for Text and links. tcm:URI are used for links, except for external links.
- Number — Used for all numbers.
- Date — Used for all dates.

No other values are supported.

### 10.6.a Queries against SQL databases

This section describes how to query custom metadata stored in MS-SQL, Oracle and DB2.

Everything entered after your query is appended to the following query:

```
"SELECT PUBLICATION_ID, ITEM_ID FROM CUSTOM_META WHERE PUBLICATION_ID=?  
AND ITEM_TYPE=? AND " + nativeQuery;
```

the `nativeQuery` is replaced by your query values. If your values are:

```
KEY_NAME='city' and KEY_STRING_VALUE='Woerden'
```

The resulting query is:

```
"SELECT PUBLICATION_ID, ITEM_ID FROM CUSTOM_META WHERE PUBLICATION_ID=?  
AND ITEM_TYPE=? AND KEY_NAME='city' and KEY_STRING_VALUE='Woerden' "
```

The question marks are filled in by the system. The publication id is the id of the publication used during the construction of your `SearchFilter` object in JSP. In ASP you must supply the Publication id in the method call.

**Note** The `ITEM_TYPE` is always 16 (Component).

You can also use `KEY_DATE_VALUE` and `KEY_FLOAT_VALUE`.







# Chapter 11 Tamino

This chapter describes how to configure and use the Software AG Tamino XML database with the Content Distributor.

For more information about Tamino, see the Tamino documentation provided with the Tamino installation set.

## 11.1 Configuration

This section describes the configuration required to use the Content Distributor with Tamino.

You must specify a Tamino database in the Broker configuration file (`cd_broker_conf.xml`), with the following credentials:

- username
- password
- database URL

For example:

```
<Configuration>
  <Global>
    . . .
  <Storage>
    <Database Type="tamino" Username="cd_broker" Password="cd_broker"
    Url="http://localhost/tamino/cd_broker"/>
  </Storage>
</Global>
. . .
</Configuration>
```

The following Tamino specific storage classes must be added to the Broker configuration `Bindings` section:

```
<Binding Name="PageMeta"
Class="com.tridion.broker.pages.meta.TaminoPageMetaHome"/>
<Binding Name="ComponentMeta"
Class="com.tridion.broker.components.meta.TaminoComponentMetaHome"/>
<Binding Name="ComponentPresentationMeta"
Class="com.tridion.broker.componentpresentations.meta.TaminoComponentPres
entationMetaHome"/>
```

```

<Binding Name="XMLComponentPresentation"
Class="com.tridion.broker.componentpresentations.TaminoXMLComponentPresenta
tionHome"/>
<Binding Name="Schema"
Class="com.tridion.broker.schemas.TaminoSchemaHome"/>

```

**Table 11-1 Tamino bindings and their purpose.**

<b>Binding</b>	<b>Class purpose</b>
PageMeta	Stores Page metadata in Tamino
ComponentMeta	Stores Component metadata in Tamino
ComponentPresentationMeta	Stores ComponentPresentation metadata in Tamino
XMLComponentPresentation	Determines if a Component Presentation is generated by a Component Template or if the unprocessed XML is available. If the unprocessed XML is available it will be inserted into Tamino in its natural form that can be queried by Tamino.
Schema	Converts a CMS/W3C Schema into a Tamino Schema that allows unprocessed XML to be inserted and queried.

## 11.2 Using Tamino

To Publish Components to Tamino the following items are required in the Content Manager:

- Schema defining the Components to store in Tamino
- Component Template producing XML Documents or XML Fragments
- Components to be inserted into Tamino

After creating these items correctly configuring the Content Broker, Publishing to Tamino is the same as Publishing any other type of Component Presentation.

### 11.2.a Schemas

For all Component types you want to query in Tamino, you must have schemas with a defined root element. These root elements are matched with Tamino 'doctypes'.

**Note** To create a nested XML structure in Tamino, you must use embedded schemas.

If Component Template-generated XML is used, there are no restrictions on schemas.

---

#### **Caution:**

There is no check in the Management System to ensure that root element names are unique. It is the responsibility of site designers to ensure uniqueness. If multiple Schemas have the same root element, then an error is raised during the deployment process. As a result, much of the Published content is not stored by Tamino.

---

## 11.2.b Component Templates

You can publish two types of XML content to Tamino. In both cases the Template Designer must ensure that the output format of the Component Template is set to XML. The Component Template must be linked to the Schemas of the Components published to Tamino.

The two XML Component publishing scenarios are described below:

- Unprocessed XML — When a template designer leaves the Component Template blank, the XML structure for the rendered Component is published with the same structure as it is stored on the Management System.
- Component Template Rendered XML — When a Template Designer fills in the Management System Template part of a Component Template, they can generate XML in the same way they would generate HTML/JSP/ASP code.

---

### Caution:

If the XML generated by the Component Template is not valid XML or depends on namespaces for element name uniqueness it is not accepted by Tamino and an error occurs during the deployment process.

---

## 11.2.c Components

The `ino:docname` attribute assigns URIs to Components or Component Presentations. Tamino checks for Component updates or inserts.

The Components that are mapped to a Content Manager schema can be accessed using their URI as `ino:docname`, for example `tcm:2-67-16`. Component Presentations can be accessed using their URI within the `tcd:` namespace, for example `tcd:/pub[2]/componentpresentation[76,67]`.

All Component Presentations are stored with the Component Presentation doctype. Depending on the type of Component Template used, the body of the `ComponentPresentation` element contains the ComponentPresentation XML content. If a Schema that matches the content was published, the `ComponentPresentation` element contains a reference to the XML content in its own doctype.

The following example is a Component Presentation including content:

```
<component-presentation component-id="76" ino:id="4" template-id="67">
  <sample>
    <element>body</element>
  </sample>
</component-presentation />
```

The following example shows the reference to content:

```
<component-presentation component-id="67" ino:id="1"
  reference="tcm:2-67-16" template-id="66"/>
```



### 11.2.d Updating a Component in Tamino

Any changes to a Component or ComponentPresentation are updated in Tamino, if they are republished, and if the content can still be mapped to the schema in Tamino.

### 11.2.e Unpublishing from Tamino

ComponentPresentations are always removed if they are unpublished. Components are removed from Tamino if there are no ComponentPresentations referring to them.

### 11.2.f Dynamic Content

The following example shows usage of a single XSLT rendering a component on a page, based on a URL query variable.

```
<%
  { //Define separate scope to prevent variable name clashes
    //Assumes a publication Id variable is set in the page header
    ComponentPresentationFactory factory =
      new ComponentPresentationFactory(publicationId);

    // the cId parameter comes from the Dynamically generated link to
    // the page that contains this code.
    int componentId = Integer.parseInt(request.getParameter("cId"));

    //This value is hardcoded when this code is generated by the
    //component template
    int templateId = '12';
    XMLComponentPresentation cp =
      factory.getComponentPresentation(componentId, templateId);
    // do the actual XSLT transformation
    out.print(XSLTProcessor.getString(cp, PageID));
  }
%>
```

### 11.2.g Schema mapping

There are two scenarios for mapping schemas.

- Open schema with a `<ComponentPresentation>` root element — This scenario is used for ComponentPresentations that are generated by a Component Template and for "raw" XMLComponentPresentations that have no defined root element.
- One-to-one mapping of a Tridion R5 schema to a Tamino Schema.



### 11.2.h XSLT transformation of W3C schemas

The Tamino 3 Schema is capable of dealing with W3C schema with some limitations and needs some additional information for storage purposes. The W3C schema and the Tamino 3 Schema are both XML, and so can be transformed by XSLT.

The stylesheet does the following:

- Strip additional info
- Remove tags and attributes not supported by Tamino
- Convert non-supported XSD datatypes to supported ones
- Add standard indexing to all elements.

### 11.2.i ReferenceCounting

A single component is kept in the database for multiple ComponentPresentations. The Component will not be removed before all ComponentPresentations that refer to this component are unpublished. Once all Components that use a schema are unpublished the Schema will also be 'undefined' (removed) in Tamino.

## 11.3 Querying Tamino

This section lists several examples of querying Tamino using JSP, or ASP. The following examples are used:

- Querying Tamino in JSP
- Querying Tamino in ASP
- Retrieving a single component in JSP
- Retrieving a single component in ASP

#### Querying Tamino in JSP

```
<%@page import="com.tridion.dcp.filters.*,
com.tridion.dcp.filters.query.*, com.tridion.web.*, com.tridion.util.*"%>
<%
{
    TaminoFilter filter = new TaminoFilter("tcm:0-2-1");
    XSLTProcessor processor = new XSLTProcessor();
    String query = "/TaminoFilter[@ino:docname='tcm:2-67-16']";
    XMLQueryResult placeholder = filter.matchTamino(query, 1);
    XMLQueryResult result = filter.matchTamino("/Article", placeholder,
10);

    out.print(processor.getString(result, "tcm:0-2-1", "tcm:2-66-32"));
}
%>
```

#### Querying Tamino in ASP

```
<%
```

```

Set objFilter = Server.CreateObject("cd_broker.TaminoFilter")
Set objProcessor = Server.CreateObject("cd_broker.XSLTProcessor")

'Process form input, build Tamino query
componentId = Request.QueryString("componentURI")
strQuery = "/Article[@ino:docname='" & componentId & "']"

'Set query and display parameters
Set result = objFilter.MatchTamino("tcm:0-2-1", strQuery, 1)
Response.Write (objProcessor.GetStringFromQueryResult(result,
"tcm:0-2-1", "tcm:2-61-32"))

Set objProcessor = Nothing
Set objFilter = Nothing
%>

```

### Retrieving a single component in JSP

```

<%@page import="com.tridion.dcp.filters.*,
com.tridion.dcp.filters.query.*, com.tridion.web.*"%>
<%
{
String componentId = request.getParameter("componentURI");

if(componentId != null) {
TaminoFilter filter = new TaminoFilter("tcm:0-2-1");
XSLTProcessor processor = new XSLTProcessor();
String query = "/Article[@ino:docname='" + componentId + "']";
XMLQueryResult result = filter.matchTamino(query, 1);
out.print (processor.getString(result, "tcm:0-2-1", "tcm:2-61-32"));
}
}
%>

```

### Retrieving a single component in ASP

```

<%
Set objFilter = Server.CreateObject("cd_broker.TaminoFilter")
Set objProcessor = Server.CreateObject("cd_broker.XSLTProcessor")

'Process form input, build Tamino query
componentId = Request.QueryString("componentURI")
query = "/TaminoFilter[@ino:docname='tcm:2-67-16']"
'Set query and display parameters

Set placeholder = objFilter.MatchTamino("tcm:0-2-1", query, 1)
Set result = objFilter.MatchTamino("tcm:0-2-1", "/Article", 10,
placeholder)

Response.Write (objProcessor.GetStringFromQueryResult(result,
"tcm:0-2-1", "tcm:2-66-32"))

Set objProcessor = Nothing
Set objFilter = Nothing
%>

```



### 11.3.a Generating links from an XSLT using the Linking API

The following example shows Tamino Filter data fields used to generate Placeholder code and linking to detail page.

```
<TaminoFilter ino:id="1">
  <Query>/Article</Query>
  <MaxResults>10</MaxResults>
  <Detail href="tcm:2-62" title="Article Detail" type="simple"/>
  <TemplateId>tcm:2-66-32</TemplateId>
  <TCMMetadata/>
</TaminoFilter>
```

The following example shows the use of the Linking API from within XSLT:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
  xmlns:xql="http://metalab.unc.edu/xql/"
  xmlns:lxslt="http://xml.apache.org/xslt"
  xmlns:tcd="http://www.tridion.com/tcd"
  extension-element-prefixes="tcd">

  <xsl:output method="html" encoding="UTF-8"/>

  <xsl:param name="publicationURI" />
  <xsl:param name="pageURI" />
  <xsl:param name="componentTemplateURI" />

  <lxslt:component prefix="tcd">
    <lxslt:script lang="javaclass" src="xalan://com.tridion" />
  </lxslt:component>

  <xsl:template match="xql:result">
    <xsl:apply-templates select="Article"/>
  </xsl:template>

  <xsl:template match="Article">
    <h1><xsl:value-of select="Title"/></h1>
    <i><xsl:copy-of select="Intro"/></i>
    <xsl:call-template name="ComponentLink">
      <xsl:with-param name="componentURI" select="@ino:docname"/>
      <xsl:with-param name="linkText" select="'Read More'"/>
      <xsl:with-param name="target" select="../TaminoFilter/Detail/@href"/>
    </xsl:call-template>
  </xsl:template>

  <xsl:template name="ComponentLink">
    <xsl:param name="componentURI"/>
    <xsl:param name="linkText"/>
    <xsl:param name="target"/>

    <xsl:variable name="componentLinkInstance"
      select="tcd:linking.ComponentLink.new($publicationURI)"/>
    <xsl:variable name="query" select="concat('componentURI=',
      $componentURI)"/>
    <xsl:variable name="link" select="tcd:getLink($componentLinkInstance,
      $pageURI, $target, $componentTemplateURI, '', $linkText, false, true)"/>
```

```

<xsl:value-of select="tcd:setParameters($link, $query)"/>

<xsl:value-of select="$link" disable-output-escaping="yes"/><br/>
</xsl:template>

</xsl:stylesheet>

```

### 11.3.b Component link resolution with XSLT

Considering the following XML fragments:

```

<article>
<title>XSLT is a powerful but complex monster</title>
<author xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="simple"
xlink:href="tcm:6-74" xlink:title="David Buddha"/>
...
</article>

<author>
<first_name>Hans</first_name>
<last_name>Speijer</last_name>
<email>david@buddha.com</email>
</author>

```

The following stylesheet fragment shows an example of link resolution in XSLT:

```

<xsl:template match="article">
<h1><xsl:value-of select="title"/></h1>
<i>by <xsl:apply-templates select="document(author/@href)"/></i>
</xsl:template>

<xsl:template match="author">
<xsl:value-of select="first_name"/> <xsl:value-of select="last_name"/>
<xsl:template/>

```

## 11.4 TaminoFilter

The TaminoFilter allows you to execute XPath Queries on content stored in the Tamino Database. Queries on Tamino using the TaminoFilter result in XMLQueryResult objects. These objects can be supplied to the XSLTProcessor, or again to the TaminoFilter to filter merge the result with the result of a new query.

### 11.4.a Using TaminoFilter to query Tamino

#### Creating a TaminoFilter and Executing a Query in ASP

```
<%
Dim TaminoFilter, XSLTProcessor, XMLQueryResult
Set TaminoFilter = Server.CreateObject("cd_broker.TaminoFilter")
Set XMLQueryResult = TaminoFilter.MatchTamino("tcm:0-1-1", "/*[Author
~='David'", 10)
%>
```

#### Creating a TaminoFilter and Executing a Query in JSP

```
<%@page import="com.tridion.dcp.filters.*,
com.tridion.dcp.filters.query.*, com.tridion.web.*" %>
<%
TaminoFilter filter = new TaminoFilter("tcm:0-1-1");
XMLQueryResult result = filter.matchTamino("/*[Author ~='David'",
10);
%>
```

### 11.4.b Merging Queries

#### ASP Sample Executing and merging multiple queries

```
<%
Dim TaminoFilter, XSLTProcessor, XMLQueryResult, XMLQueryResult2
Set TaminoFilter = Server.CreateObject("cd_broker.TaminoFilter")
Set XMLQueryResult = TaminoFilter.MatchTamino("tcm:0-1-1", "/*[Author
~='David'", 10)
Set XMLQueryResult2 = TaminoFilter.MatchTamino("tcm:0-1-1",
"/*[Author ~='Adrian'", 10, XMLQueryResult)
%>
```

#### JSP Sample Executing and merging multiple queries

```
<%@page import="com.tridion.dcp.filters.*,
com.tridion.dcp.filters.query.*, com.tridion.web.*" %>
<%
TaminoFilter filter = new TaminoFilter("tcm:0-1-1");
XMLQueryResult result = filter.matchTamino("/*[Author ~='David'",
10);
XMLQueryResult result2 = filter.matchTamino("/*[Author ~='Adrian'",
result, 10);
%>
```





# **Part II**

---

# **Linking**







# Chapter 11 Linking

The Content Manager allows you to create links between Content Manager items. These links are references to another item within the Content Manager:

- Page links are links between Pages. Template designers typically create these links in Page Templates to create navigation in a published site
- Binary links are links from Pages or Components to Multimedia Components (such as Word documents, PDFs, and images). These links are often created in Components as Multimedia Component link fields, or in Component format areas as Multimedia Component links.
- Component links are links to a Component. These links can be created in a Component as Component link fields or as Component links in Component format areas, or as, for example, a "Read more" link that is generated by a Component Template.

When Pages that contain links are published to the Content Distributor, code is generated and inserted into published Pages. On a published Page, Linking uses this code to determine how valid or invalid links are presented.

Tridion Linking validates and resolves hyperlinks when a visitor views a Page. At request time, Linking looks for the metadata of the requested resource. This metadata will only exist if the related Page, Component, or Binary is published.

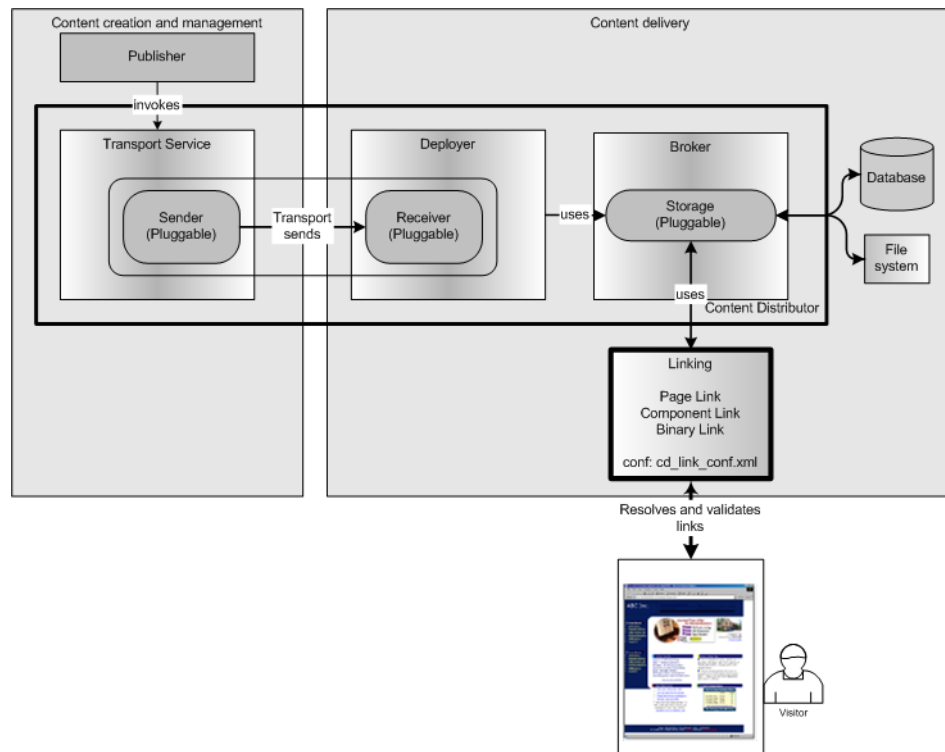
Linking information is stored by the Content Broker. This information includes the data that Linking needs to resolve and validate links between Components, Component Presentations, and Pages:

- Page metadata — Used to find the URI of a certain Page
- Binary metadata — Used to find the URI of a certain binary
- Link info — Describes the relationship between Pages and Component Presentations that are assigned to them

You can store link information and metadata on a file system or in a SQL database (MS SQL, DB2, and Oracle).

If this metadata is found, Linking creates a hypertext reference (HREF) in the published Page. If no metadata is found for the item that is linked to, either no link is generated, or text with no linking information is displayed.

The resulting behavior of text with invalid links depends on the code that was generated when Pages were published. For example, if a link is in running text, you may want to display the text but not as a hyperlink. If a link is a "Read more" link, you may not want the link to appear at all.



**Figure 11-1 Linking configuration**

Note: Linking validates links to Tridion content only. Links to external sites are not validated.

## 11.1 Link validation and resolution

This chapter describes how the Linking module validates and generates the following types of links:

- Page links
- Component links
- Binary links

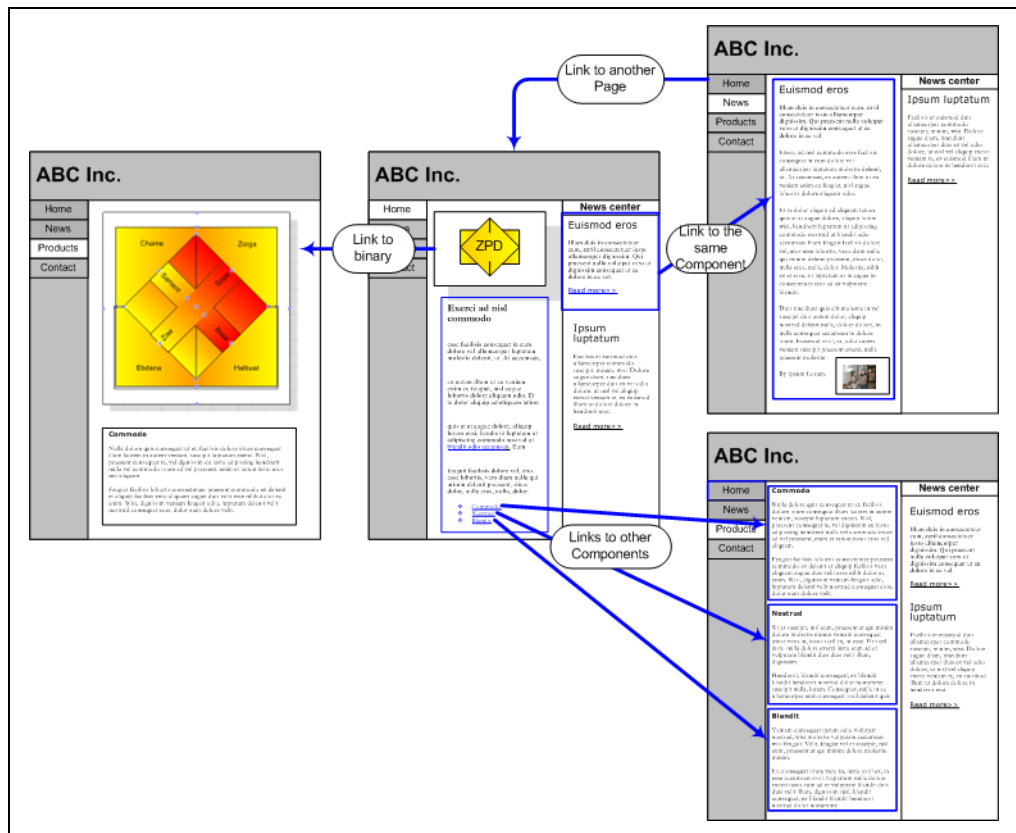


Figure 11-2 Hyperlinks in a published site

## 11.2 Page links

Page links are links to other Tridion Pages.

Template designers create Page links within a template. Usually Page links are created within a Page Template to create navigation structures. In the Content Manager, a Page Template may, for example, contain a script that looks for the index page within the current structure group.

When the Page is published, code is generated that instructs the Content Distributor to create the link to the Page with the associated URI.

When a visitor accesses a Page with the link, Linking reads relevant metadata from the datastore and looks for the link. If the Page to which the link points is found, a link is created. If the Page to which the link points is not found, no link is generated.

The following example depicts the code that is generated on a Page before the viewed and which Tridion Linking uses to resolve the Page link:

**Example** The page link shown below creates a link to a page with `tcm:203-1528-64` in JSP.

```
String linkAsStr =
pageLink.getLinkAsString("tcm:203-1528-64","anchor", "onmouseover="this.
style.color='red'"" onmouseout="this.style.color='blue'"" , "This is a
Page Link", true, "");
```

**Example** The page link shown below creates a link to a page with `tcm:203-1528-64` in ASP.

```

dim pLinkAsStr
pLinkAsStr =
pLink.GetLinkAsString("tcm:203-1528-64", "anchor", "onmouseover=""this.sty
le.color='red'"" onmouseout=""this.style.color='blue'""", "Page link
text", True, "")

```

## 11.3 Binary links

Binary links link to binary files such as images, pdfs, Word documents, etc. In the Content Manager, these binary files are Multimedia Components.

You can use the `BinaryLink` class to link to a binary based on a Tridion URI.

Binary links are created in the Content Manager in the following ways:

- Multimedia Component links added to Components in format areas or in Multimedia Component Link fields
- Links to binary files created in templates

The default Template Building Block contains code that will generate linking code to binary files. Published Pages that refer to this linking code contain scripting that is used by Linking to ensure that the link is resolved and valid.

When a Page is published that contains binary links, code is generated that instructs the Content Distributor to create a link to the binary file with the associated URI.

When a visitor accesses a Page with the link, Linking reads the binary metadata in the datastore. If the binary to which the link points is found, a link is created. If the binary to which the link points is not found, no link is generated.

The following examples depict the code that is generated on a Page before it is viewed and which Tridion Linking uses to resolve the binary link:

### JSP

*Example* `binaryLink.getLinkAsString("tcm:203-1488", "", "Binary link text", "attrs", true);`

### ASP

*Example* `<%
Dim BinaryLink
set BinaryLink = Server.CreateObject("cd_link.BinaryLink")
%>
<%= BinaryLink.GetLinkAsString("tcm:0-1-1", "tcm:1-43", "", "linkText",
"", true) %>`



## 11.4 Component links

Component Links are links to:

- the same Component
- a different Component

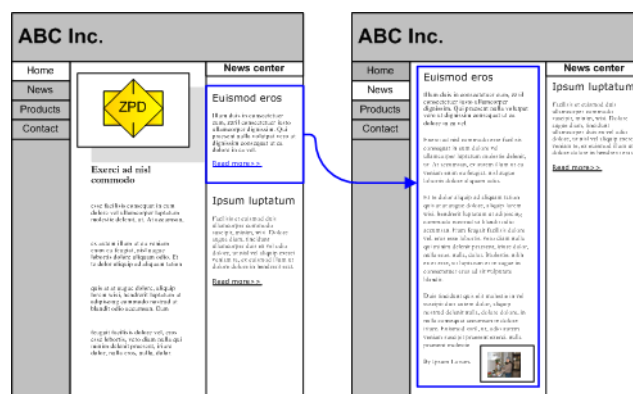
### 11.4.a Components that link to the same Component

A Component may appear more than once in a site. For example, an article may be presented in two different ways:

- It may appear on a summary page with just the title of the Component, a short summary, and a Read more link
- It may also appear on a full article page in which the full content of the Component is displayed.

While the content for this content is derived from the same Component, in each of these cases, the Component is combined with a different Component Template. In this situation, you want to link from the summary to the full article.

Typically, "Read more" links are generated by Component Templates and are not part of the original Component. The priority of the Component Template ensures that the "Read more" link directs visitors to the Page which contains the full article. In this example, the summary article Component Template would be assigned a priority of Low, while the full article Component Template would be assigned a priority of High. Linking determines what Component Presentation to link to using the priority of the Component Template.



**Figure 11-3 Component Presentation with a link to the same Component**

For more information about how linking uses Component Template priorities and the directory hierarchy to determine which Component to link to, refer to *"Resolving Component links"* on page 107.

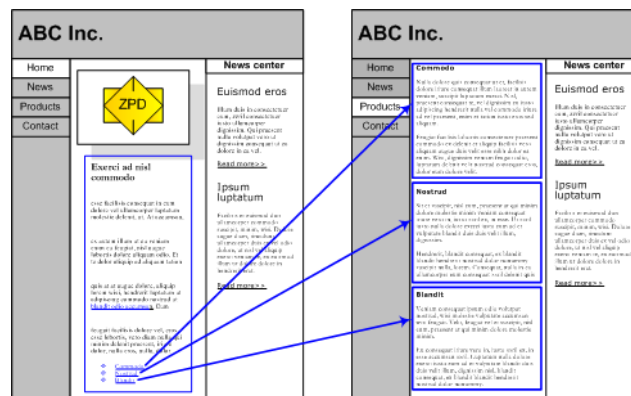


## 11.4.b Components that link to a different Component

A Component link can link to different Components if, for example, links are generated by the Page Template or Component Template based on Components that are stored in the same folder, or if a Component actually contains links to other Components in format areas or in Component Link fields.

You can design a schema that will enable authors to link to other Components or Multimedia Components within the Content Manager. A Schema can include:

- Component Link fields — In a Component these fields can store links to other Components
- Multimedia Link fields — In a Component these fields can store links to Multimedia Components (binaries)
- Format areas — In a Component, text can be formatted to hyperlink to Components or to Multimedia Components



**Figure 11-4 Component link to a different Component**

For more information about how linking uses Component Template priorities and the directory hierarchy to determine which Component to link to, refer to "*Resolving Component links*" on page 107.

## 11.4.c Creating Component links in templates

Use the following Linking API references for examples of how to create Component Links:

- Linking\_JSP\_51SP4.zip — The Java API for linking
- Linking\_ASP\_51SP4.chm — The COM API for linking
- ContentDelivery\_NET\_51SP4.chm — The .NET API for Content Delivery products

You can create a Component link from a template using the `getLinkAsString` method in the `ComponentLink` class.

When identifying a link using the URI of the Component in a template, use the following syntax:

```
getLinkAsString(String pageURI, String ComponentURI, String
templateURI, String linkAttr, String linkText, boolean textOnFail)
```

**JSP**

**Example** The Component link shown below creates a link from Page `tcm:0-203-1`, Component `tcm:203-1489`, with Component Template `tcm:0-957-32`.

```
ComponentLink componentLink = new ComponentLink("tcm:0-203-1");
String linkString =
ComponentLink.getLinkAsString("tcm:0-203-1","tcm:203-1489","tcm:0-957-32",
"", "Component link text", true);
```

**ASP**

**Example** The Component link shown below creates a link from Page `tcm:0-203-1`, Component `tcm:203-1489`, with Component Template `tcm:0-957-32`.

```
dim ComponentLink
set ComponentLink = createObject("cd_link.ComponentLink")
dim cLinkAsStr
cLinkAsStr = ComponentLink.GetLinkAsString("tcm:0-203-1","tcm:203-1489", _
"tcm:0-957-32","", "Component link text", True, 1)
```

**11.4.d Creating Component Link objects in a Page Template**

The Default Component Template always creates a new ComponentLink object when it encounters a Link (for instance originating from a text-area).

If many Component Links exist on a Page, you should create a Component Link object in the Page Template. You can then reuse this object for every Component Link that needs to be created.

**11.4.e Resolving Component links**

Linking resolves Component Links to the same Component, the link is created based on Component Template priorities and the directory hierarchy of the published site.

**Component Template priorities**

Linking resolves Component Links to the same or a different Component using the priority of the Component Template and then the directory hierarchy on which the Components are used in Pages. A Component will always link to a Component that is rendered with a Component Template of the highest possible priority.

You set the priority on the Component Templates in the Content Manager. The priority system allows Linking to determine which published Component Presentation the link will lead to. If the Component is used on more than one Page, Linking will generate a link to the Component Presentation that is rendered with a Template of the highest priority. It is possible to assign the following priorities to a Component Template:

- **Never link**
- **Low**
- **Medium**
- **High**



A link will only be generated to a Component that uses a Component Template with a higher priority i.e. low to high or medium but not medium to low or high to medium or low.

For example, if you would like a summary Component to link to the same Component as a full article, you would assign a Component Template the priority of "Low" or "Never Link" for the summary article and a priority of "High" for the full Component.

**Note** A Component Link will never link to a Component that is rendered using the same Component Template. This prevents Pages from having links to exactly the same Component Presentation.

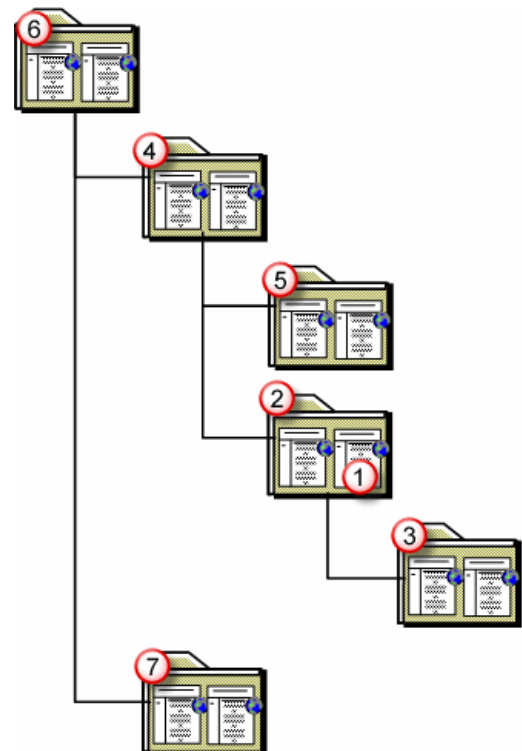
### Directory hierarchy

If the Component is published more than once using Component Templates of equivalent priority, the "closest" Component with the highest priority in the directory hierarchy is used. Linking is designed to look first in nested directories and if no link is found then it look in parent directories. The following section describes the directory hierarchy that linking uses.

The hierarchy is determined by the distance between the Page with the link and the location of the referred item within the directory. Linking assigns nested links (links that move down into a directory) a low value (for example one (1)) and parent links (links that move up in the directory) a high value of (for example 100). Linking selects the item with the lowest possible value.

When selecting the Page, binary or Component to link to, Linking checks for an applicable item in the following order (the numbers in the diagram refer to the numbers in the explanation):

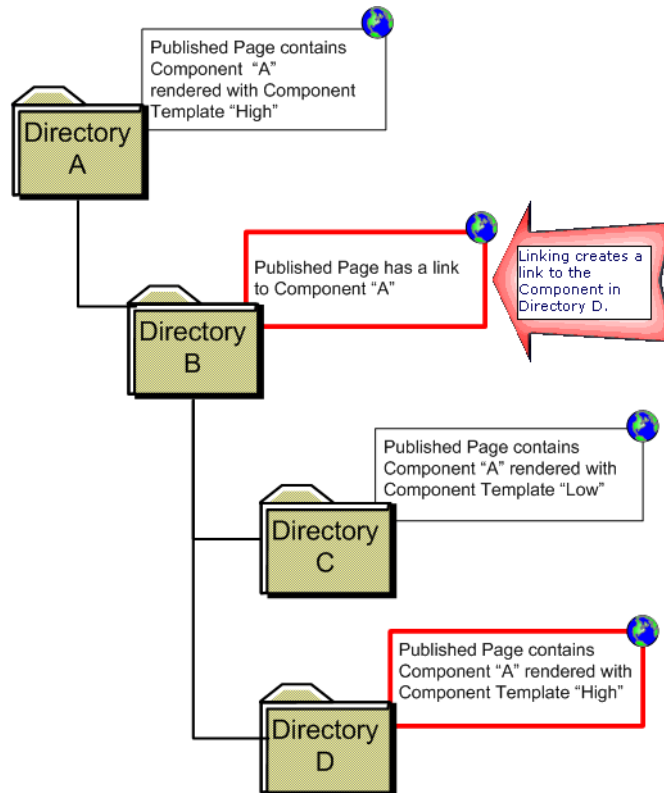
- 1 Same Page as the link. (Value 0)
- 2 A Page in the same directory. (Value 0)
- 3 A Page in a child directory. (Value 1)
- 4 A Page in the parent directory. (Value 100)
- 5 A Page in a sibling directory. (Value 101)
- 6 A Page in the grandparent directory. (Value 200)
- 7 A Page in the sibling of the parent directory. (Value 201)



If more than one possible candidate exists, Linking creates an HREF to the most recently published Page.



In the following example (figure 11-5), the Component is published on two different Pages using a Component Template with the priority "High". To determine which Component to create link to, Linking looks at the value of each of the Directories. The Page published in directory A has a value of 100. The Page published in directory D has a value of 1. Therefore, Linking creates a link to the Component published on the Page in Directory D.



**Figure 11-5 Component links resolved using Template priority and directory hierarchy**







# Chapter 12 Linking configuration

The following configuration values influence the way in which linking generates and uses links:

- Linking uses configuration details that are configured in the Linking configuration file: `cd_link_conf.xml`. Use this configuration file to configure logging settings as well as optional Publication specific settings that are used to resolve links.
- The Broker configuration file (`cd_broker_conf.xml`) contains bindings that are used to write and retrieve metadata including the following classes:
  - ▶ `LinkInfo` — Used to store metadata that is used to resolve Page and Component links. This class controls how linking metadata is stored. By default, Linking uses comma separated values on the file system.
  - ▶ `ComponentPresentationMeta` — Used to store Dynamic Component Presentations
  - ▶ `PageMeta` — Used to store Page metadata
  - ▶ `BinaryMeta` — Used to store binary metadata
- Within Tridion Component Templates, Page Templates and Template Building Blocks, a number of methods can be used to create and render links. The default templates use the following methods:
  - ▶ Function `DisplayPublishedComponentLink(ByVal linkedComponent, ByVal linkText, ByVal targetLanguageId)`
  - ▶ Sub `ResolveComponentLinks(ByVal domdoc)`

**Note** You can also configure the Java Virtual Machine that the Tridion Dynamic Linking service starts. See Appendix J, "*Configuring the Java Virtual Machine for COM services*" for details.

This chapter describes:

- Linking configuration file
- Linking and Content Broker configuration

## 12.1 Linking configuration file

The Linking module is configured using Linking configuration file (`cd_link_conf.xml`) to determine the following information:

- Logging settings — determine the behavior and location of one or more logging streams
- Publication settings — a container in which you can configure linking settings for specific Publications.

The configuration file is validated against the `cd_lnk_conf.xsd` schema.

The following example depicts a sample linking configuration file:

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration Version="5.1">
  <Global>
    <Logging>
      <Logger Level="info">
        <FileLogger Level="info" Location="D:\Tridion\logs\cd_link.log"/>
        <!--<ConsoleLogger Level="info" Trace="on"/>-->
      </Logger>
    </Logging>
  </Global>
  <Publications>
    <Publication Id="1">
      <Host Domain="localhost" Port="80" Protocol="http" Path=""/>
      <Linking ComponentAnchors="false"/>
    </Publication>
    <Publication Id="2">
      <Host Domain="www.myWebsite.com" Port="80" Protocol="http" Path=""/>
      <Linking ComponentAnchors="true"/>
    </Publication>
  </Publications>
</Configuration>
```

### 12.1.a Logging

The `Logging` element specifies the behavior and location of one or more logging streams. The `Level` attribute specifies the amount of information being logged. The logging levels are:

- `info` — Information messages.
- `warning` — Potential error conditions.
- `error` — Errors that prevent the software from functioning correctly.
- `fatal` — Unrecoverable errors.

The `FileLogger` logs to the log file specified by the `Location` attribute:

```
<FileLogger Level="info" Location="C:\tridion\log\cd_link.log"/>
```

You can also print log messages to a console using the `ConsoleLogger` element.

An administrator should make sure that the account used by Tridion Linking has rights to write to the log file, and create directories if needed.

### 12.1.b Publications

The `Publications` element is a container for the `Publication` element.

It is possible to specify the following attributes for each `Publication` using a `Publication` sub-element:

- `Domain` — The domain of the Web site. The default domain is `localhost`.
- `Port` — The port of the Web site. The default port is `80`.
- `Path` — The path of the virtual directory of your Web site. The default path is `/`.
- `Protocol` — The protocol used for the Web site. The default protocol is `HTTP`.

For the `Publication` element, you can also specify the following attributes about how links are generated using the `Linking` sub-element:

- `ComponentAnchors` — Indicates if anchors should be added to `Component` links so that the browser jumps to the position of the targeted `Component` `Presentation`. The value can be either `true` or `false`.
- `AddComponentLinkInfo` — Indicates if additional information should be added to the `QueryString` of a `Component` Link URL. The value can be either `true` or `false`.

If this value is `true`, then a string that contains the `Component` ID and the `Page` ID are added as CGI key/value pairs in the following form:

`ComponentId=x&SourcePageId=y`

**Note** If you are using the Tridion WAI Integration, you should set this value to `true` since the WAI uses this information for `ComponentLinkTracking`.

## 12.2 Linking and Content Broker configuration

Linking uses the following information that is provided by the Content Broker configuration file:

- The directory for all `Publications` in which pages, binaries and meta-data is published (this is the `DefaultRootLocation` element)
- Publication specific settings
  - ▶ `DataRoot` — The directory in which meta-data should be stored and retrieved on the file system.
  - ▶ `DocumentRoot` — The base-directory in which pages and binaries are located.

For more information about the Content Broker configuration file, refer to "*Broker configuration*" on page 61.

The metadata used by Tridion Linking can be stored in the following locations:

- File System
- SQL Databases



## 12.2.a Linking bindings

Linking bindings determine how metadata used by Linking is stored. These bindings are configured in the Content Broker configuration file (`cd_broker_conf.xml`).

The following bindings are used:

- `LinkInfo` — information that describes the relationship between Pages and Component Presentations. This is used for Component links.
- `PageMeta` — metadata about Pages. This data is used for Page links.
- `BinaryMeta` — metadata about binaries. This data is used for Binary links.

These bindings can be bound to an implementation class that handles the storage of the specific kind of information. The bindings can store metadata on the file system or in a SQL database.

For more information about configuring Content Broker bindings, refer to "*Bindings*" on page 70.

Table 12-1 describes the linking bindings that are used for the following storage options:

- File system — The default directory is `data/pubx`.
- SQL database — MS SQL, DB2, and Oracle

**Table 12-1 Linking bindings**

Symbolic name	Class	Description
File system		
<code>LinkInfo</code>	<code>com.tridion.Broker.linking.FSCSVLinkInfoHome</code>	Reads and writes link metadata to CSV files.
<code>LinkInfo</code>	<code>com.tridion.Broker.linking.FSXMLLinkInfoHome</code>	Reads and writes link metadata to XML files.
<code>PageMeta</code>	<code>com.tridion.Broker.pages.meta.XMLFilePageMetaHome</code>	Reads and write Page metadata to XML files.
<code>BinaryMeta</code>	<code>com.tridion.Broker.binaries.meta.XMLFileBinaryMetaHome</code>	Reads and write binary metadata to XML files.
SQL database		
<code>LinkInfo</code>	<code>com.tridion.Broker.linking.SQLLinkInfoHome</code>	Reads and writes link metadata to an SQL database.
<code>PageMeta</code>	<code>com.tridion.Broker.pages.meta.SQLPageMetaHome</code>	Reads and writes Page metadata to an SQL database.
<code>BinaryMeta</code>	<code>com.tridion.Broker.binaries.meta.SQLBinaryMetaHome</code>	Reads and writes binary metadata to an SQL database.

# **Part III**

---

# **Web and Application Server Integration**





# Chapter 13 Web and Application Server Integration

The Tridion Web and Application Server Integration (WAI) provides the following functionality:

- Profiling and personalization
- Dynamic content assembly

The WAI is installed on top of the Content Broker. The Broker accesses the metadata and Dynamic Component Presentations. The Broker framework is also used to store visitor profiles.

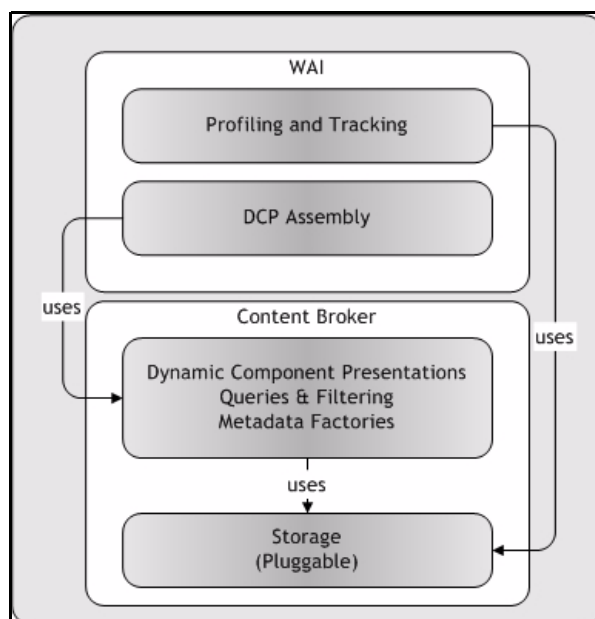


Figure 13-1 WAI architecture

## 13.1 Web and Application Server Integration (WAI)

The Tridion WAI provides further functionality for your published content:

- The ability to profile your visitors and personalize content
- The ability to present Dynamic content to visitors

### 13.1.a Profiling and personalization

In conjunction with explicit profile information that you may be gathering for your visitors (such as information gathered from Web forms), you can also track the types of content that a visitor accesses, and the specific Pages, Components, and links that a visitor uses.

The WAI uses cookies to identify your visitors. Once a visitor is identified, the visitor's browsing behavior can be tracked and added to their visitor profile on your database. It is possible to create profiles based on two types of visitor behavior:

- Tracking Keys provide information about the type of content that the visitor shows interest in
- Tracking provides information about the Pages, Components, and Links that a visitor was served within specified time periods.

Using information about your visitors, you can then define Target Groups. Target Groups are basically categories of visitors. For example, you could define a Target Group that included Women between the age of 30 and 50 who have shown interest in Graphic Design Software by defining a Target Group that identifies this type of visitor.

You can associate Target Groups with Component Presentations on a Page. When a known visitor enters the Page, you can ensure that the visitor is only shown the Component Presentations that apply to their personal profile.

### 13.1.b Dynamic content assembly

The WAI allows you to publish Components and Component Templates (Component Presentations) separately from Pages. These Component Presentations are not embedded in Pages, but are instead, published to a Broker content repository. These Component Presentations are called Dynamic Component Presentations.

These Component Presentations can be accessed in two ways:

- If the Component Presentation is added to a Page, the published Page includes code that points to the Component Presentation. The Component Presentation is rendered on the Page when the visitor views the Page.
- If the Component Presentation is only published to the Content Broker, you can use queries or filters in the Content Distributor to retrieve this content.

Unlike Component Presentations that are embedded on a Page, Dynamic Component Presentations are republished separately. This means that if a Dynamic Component Presentation is added to a Page, you only need to republish the Component Presentation rather than the entire Page to update content.



# Chapter 14 Profiling and personalization

The WAI enables you to profile your web site visitors:

- Tracking Keys — Tracking Keys identify the type of content a visitor is has viewed. Tracking Key values are increased or decreased based on the content on the Pages that a visitor accesses.
- Tracking — Tracking information tracks the Pages and Components that a visitor requested. Tracking information is time period specific and can be used to obtain web site statistics for specific time periods. You can track:
  - ▶ Pages requested by a visitor
  - ▶ Components viewed by a visitor
  - ▶ Component Links that have been followed by a visitor

Tracking Keys and Tracking identify individual visitors using cookies. When a visitor accesses your site, the WAI Page object is created, which checks if the visitor has a cookie:

- if not, a cookie is generated, which maps to a newly created record for the visitor
- if so, the visitor is identified using the cookie

The Tracking Key and Tracking information that you gather is stored on a SQL (MS SQL, DB2, Oracle) database. Depending on what you have implemented (Tracking Keys and/or Tracking), the following information is stored in the Content Broker database:

- Users — Information about your visitor. The User Id is used to identify each visitor for Customer Characteristics, Tracking Keys, Page visits, Component visits, and Component link visits.
- Customer Characteristics — This is information that you may be gathering using a Web form or other direct visitor input. This information is not automatically gathered by Tridion, but needs to be implemented by WAI API calls on the Content Delivery server.
- Tracking Keys — Values of the Tracked Keywords. Depending upon your configuration, values for Keywords are incremented or decremented depending on what a visitor accesses on your site.
- Tracking — Pages, Components, and Component Links that a visitor requests within a timeframe.

- Timeframes — Timeframes are used by Tracking (not by Tracking Keys). For example, timeframes can be used to identify what the peak web site use is after you change your home page.

You can personalize the content that a visitor is shown on your web site using the Tracking Key and Customer Characteristic information that you have gathered. You can create Target Groups that identify types of visitors.

When you define a Target Group, you identify the Customer Characteristics and Tracking Key information that corresponds with visitor profiles. You then associate Target Groups with Component Presentations on Pages. When these Pages are published and a known visitor visits the Page, the visitor is shown only the Component Presentations that match their profile.

This chapter describes:

- Setting up Profiling
- Tracking Keys
- Tracking
- Configuring the WAI for profiling
- Personalizing content

### 14.1 Setting up Profiling

On a published Page, code on the Page activates Tracking Key collection and/or a track dispatcher that gathers information about served Pages, Components, and Component Links. This code also creates a WAI Page object handles the identification and persistence of visitors using cookies.

To generate this code, you add a call to Page Templates that enables Tracking Keys and Tracking. You activate tracking using the following call:

```
[% Call ActivateTracking(True) %]
```

When a Page is published, this call is transformed into the code that activates Tracking Keys and Tracking if configured.

This chapter describes how to perform the following tasks to set up Tracking Keys:

- Create Categories and Keywords for your content
- Configure Tracking Key elements in the WAI configuration file (cd\_wai\_conf.xml). The following elements contain values that are used by Tracking Key collection: Global, Presentations, Host, Personalization, Keys
- Create Schemas for the content for which you want to collect Tracking Key information, and use a Category to define a list field in this Schema
- Create Content based on the Schemas and use a Keyword in a list field
- Create Component Templates that use the Schema as a Linked Schema and identify the Category as a Tracked Category

This chapter describes how to perform the following task to set up Tracking:

- Configure the Tracking related elements in the WAI configuration file (cd\_wai\_conf.xml). The following elements contain values that are used by Tracking: Global, Presentations, Host, Tracking, Timeframe, Components, Pages, ComponentLinks, Exclude, and CustomerCharacteristics.

## 14.2 Tracking Keys

Tracking Key information provides specific information about the types of content visitors access. Tracking Keys can be used to track Keywords that are associated with specific content on your web site.

You identify the types of content a visitor accesses by assigning Tracking Keys to content. These Tracking Keys are based on Content Manager Categories and Keywords. You assign Content Manager Categories to content (Components) and layout (Component Templates) in the Content Manager.

For example, if a section of your Web site has a section that contains software reviews, you may want to create Categories and Keywords that resemble the types of content contained in the reviews.

You could create a Category called "Software" and then create the following associated Keywords:

- Graphics and publishing
- Internet applications
- Music and video
- Operating systems
- Utilities

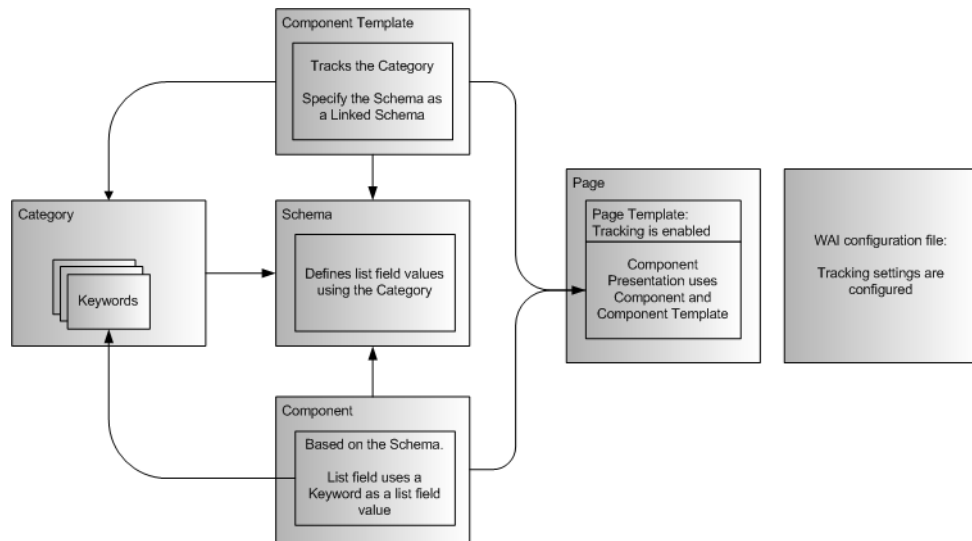
When the Category is used to identify list field values in a Schema, Authors can select a predefined Keyword to fill in a field about that content. If Tracking Keys are enabled for the published content, the values of the Tracking Key is incremented in the database that stores visitor information. You can then track the types of software reviews that the visitor is most interested in.

For example, one or more Components use the Keyword called "Graphics and publishing". A visitor enters your web site and goes to a Page that contains Components that use the "Graphics and publishing" Keyword.

If the visitor does not have a cookie, the WAI generates a cookie, creates a new record for the user and increments the "Graphics and publishing" keyword in the database. If the visitor has a cookie, the WAI uses the cookie to identify the visitor and increments the "Graphics and publishing" keyword in the database.

See Figure 14-1 "Setting up Tracking Keys" on page 122, which depicts the relationships between items that you use to set up Tracking Keys on content.





**Figure 14-1 Setting up Tracking Keys**

This section describes the items and settings required to implement Tracking Keys:

- Creating Categories and Keywords
- Using Categories and Keywords as Schema list field values
- Tracking Categories in Component Templates

*See also* For information about configuring the WAI configuration file for use with Tracking Keys, refer to "Configuring the WAI for profiling" on page 126.

### 14.2.a Creating Categories and Keywords

Categories and Keywords are used as Tracking Keys. You can create Categories and Keywords to create general classification and related values:

- A Category is a general classification in which a list of Keywords can be created.
- A Keyword is a value in a Category.

Create Categories and Keywords that are relevant to the type of content that you would like to see tracked within your Web site.

#### Creating a Category

Categories and Keywords are managed for a Publication. If the Publication is a parent in a Blueprint, the Categories and Keywords are shared to any child Publications, in which they can be localized and edited.

The following items can use Categories and Keywords:

- Schema list fields where a Category determines the Keywords that can be selected from a list field in a Component
- Component as list values where a Schema uses a Category to define a list field's values
- Component Templates where a Category is used as trackable metadata

**Prerequisites** To create a Category, you must have Category Management rights.

**To create a Category:**

- 1 In CM Explorer, select the Publication in which you want to create a Category.
- 2 Double click on the Category node in the navigation tree. Click the **Add Category** button the toolbar. An Add Category window appears.
- 3 On the **General** tab, fill in the following fields:
  - **Name:** this is the name of the Category as it will appear in selectable lists in the Content Manager
  - **XML Name:** this is the XML name.

**Note** The XML Name field can only contain letters without accents (A-Z, a-z), digits (0-9), underscores “\_” and/or hyphens “-”. The first character must be a letter or an underscore character.

- 4 If you have Permission Management rights, you can modify the security settings for this Category.
- 5 Click **Save & Close** on the toolbar.

**Results** A Category is created. Keywords can now be created within this Category.

**Creating a Keyword**

Create a Keyword to add a value to a Category.

**Prerequisites** To add a Keyword to a Category, you must have Category Management rights and write permissions for the Category.

**To add Keywords to a Category:**

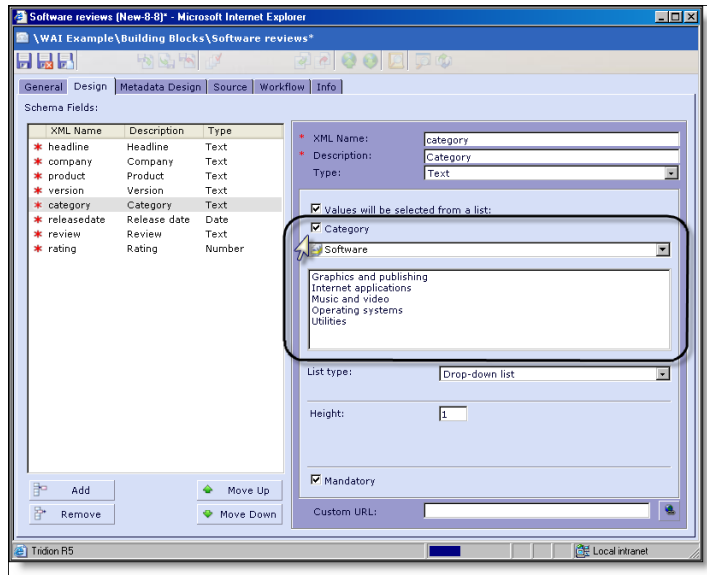
- 1 In CM Explorer, select the Publication in which you want to add Keywords to a Category.
- 2 In the list view, select a Category.
- 3 Right-click and select **New>Keyword**.
- 4 Type a value in the New Keyword window.
- 5 Do one of the following:
  - To continue to add Keywords, click the **Save & New** button on the toolbar.
  - To complete adding Keywords, click the **Save & Close** button on the toolbar.

**Results** The Category in which the Keyword is created is updated to include the new Keyword.

**14.2.b Using Categories and Keywords as Schema list field values**

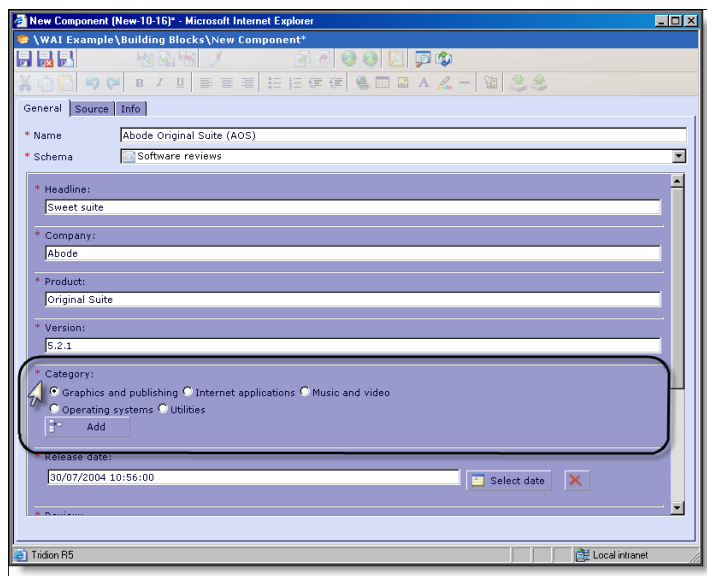
To apply a tracking key to a Component, you can use a Category within a Schema list field. When users create Components based on this Schema, they can select one of the Category’s Keywords as a value for this list field.

Using the previous example, if you wanted to track reviews using the "Software" Category, you could create a list field in a Schema, and use the predefined Category to add the Software Keywords as possible values to the Schema.



**Figure 14-2 Schema list field and field values defined by a Category**

When a user creates a Component based on this Schema, the user can select a Category list value (a Keyword) to define the value.



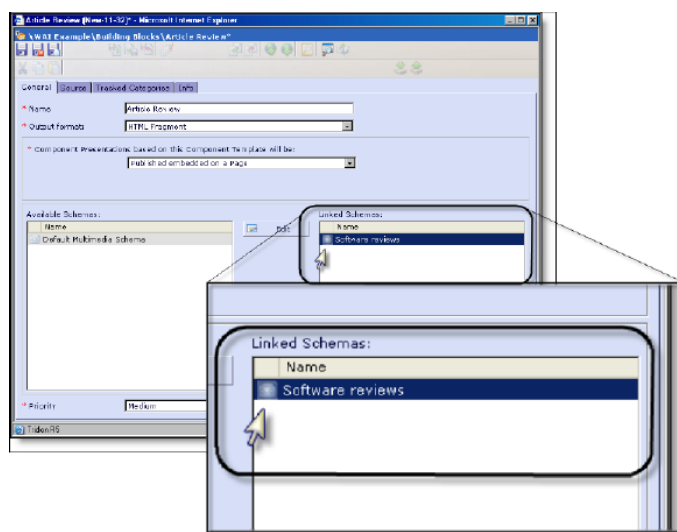
**Figure 14-3 Component that uses a Keyword as a field value**



### 14.2.c Tracking Categories in Component Templates

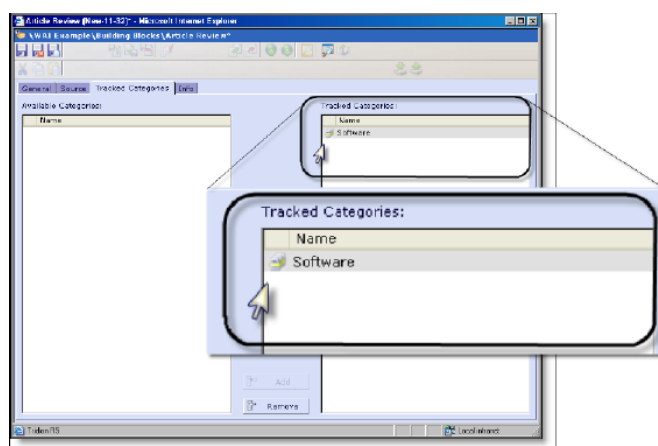
To track Categories for Components, design a Component Template that will enable the tracking mechanism:

- For the Component Template output, select either ASP or JSP
  - ASP VBScript
  - ASP JScript
  - JScript Scripting
- Add Schemas that use Categories and Keywords as list values as Linked Schemas for this Component Template



**Figure 14-4 Linked Schema**

- Add one or more Category to the Component Template as a "Tracked Category"



**Figure 14-5 Tracked Categories**

When a Page that uses this Component Template is published, code is generated in the published Page. When visitors access the Page on which this code is generated, the any Keyword used in the Component associated with Tracked Category is tracked and its value increases.

## 14.3 Tracking

Tracking gathers information about the Pages and Components that a visitor was actually served. Tracking information is time period specific and can be used to obtain web site statistics for specific time periods. You can track:

- Pages: statistics for Pages requested.
- Components: Components on the Page
- Component Links: Provides information about the specific link that a visitor looked for.

Unlike Tracking Keys, Tracking does not gather information about content, but instead, can be used to gather information about which Pages, Components, and/or Component Links a visitor actually viewed.

Tracking is configured in the WAI configuration file (`cd_wai_conf.xml`).

You can retrieve this data from the database using the JSP API. (Note: For ASP and ASP.NET you must go directly to the database to retrieve this information.)

*See also* To configure the WAI for Tracking, refer to "Configuring the WAI for profiling" in the next section.

## 14.4 Configuring the WAI for profiling

The WAI configuration file (`cd_wai_conf.xml`) configures profiling for both Tracking Keys and Tracking. The configuration of this file is validated against the schema `cd_wai_conf.xsd`. You can configure the following settings:

- Global — configuration details that apply to all functionality
  - ▶ Logging — specifies the behavior and location of one or more logging streams
- Presentations
  - ▶ Presentation — A `Presentation` element identifies that parts of your web site for which you would like to gather Tracking Key information or for which you would like to perform Tracking. The `Path` element determines what hosts are used for a `Presentation` and how the functionality for those hosts should behave. For each `Presentation`, you can specify the following:
    - `Host` — identifies a host that should be part of the `Presentation`
    - `Personalization` — configures tracking and profiling information
    - `Tracking` — properties for tracking
    - `Timeframe` — properties for timeframes used in tracking
    - `Components` — enables the tracking of Component visits
    - `Pages` — enables the tracking of Page requests
    - `ComponentLinks` — enables tracking of Component Links
    - `Keys` — enables Tracking Keys
    - `Exclude` — exclude Pages, Components, and Paths from being tracked
    - `CustomerCharacteristics` — configures whether or not values should be trimmed

**Example**

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration Version="5.0">
  <Global>
    <Logging>
      <Logger Level="error">
        <FileLogger Level="info" Location="C:\Tridion\log\cd_wai.log"/>
      </Logger>
    </Logging>
  </Global>
  <Presentations>
    <Presentation Id="1">
      <Host Domain="www.visitorsWeb.com" Port="80" Protocol="http"
        Path="/" />
      <Host Domain="localhost" Port="80" Protocol="http" Path="/" />
      <Personalization Enabled="true" Persistence="cookies">
        <Cookie Name="TCMR5_1234567" Expires="39000" />
      </Personalization>
      <Tracking>
        <Timeframe Type="hourly" Multiplier="1" Autofill="true" />
        <Pages Enabled="true" />
        <Components Enabled="true" Max="10" Averaging="true" />
        <ComponentLinks Enabled="true" />
        <Keys Enabled="true" Increment="50" Decrement="1" Averaging="false"
          ComponentLinks="true" />
      </Tracking>
      <Exclude>
        <Pages>1;2;4-100;125</Pages>
        <Components />
        <Paths />
      </Exclude>
      <CustomerCharacteristics PreserveWhitespace="true" />
    </Presentation>
  </Presentations>
</Configuration>
```

### 14.4.a Global logging settings

The `Global` element specifies configuration logging configuration details that apply to all WAI functionality. The `Global` sub-element, `Logging`, specifies the behavior and location of one or more logging streams. The `Level` attribute specifies the amount of information being logged. The logging levels are:

- `info` — Information messages.
- `warning` — Potential error conditions.
- `error` — Errors that prevent the software from functioning correctly.
- `fatal` — Unrecoverable errors.

The `FileLogger` logs to the log file specified by the `Location` attribute:

```
<FileLogger Level="info" Location="C:\tridion\log\cd_wai.log"/>
```

If the directory structure specified by the location does not exist, it will be created automatically. Ensure that the account used by the WAI has rights to write to the file log and to create directories.

### 14.4.b Presentations

The `Presentations` element defines a set, subset, or superset of directories in which you would like to profile your visitors.

A `Presentation` specifies a directory in which profiling is implemented. You can specify multiple `Presentation` elements. You can add a `Presentation` element for each set of directories in which you want to specify tracking parameters, and assign an `Id` attribute for that `Presentation`.

The `Id` has no relationship with Content Manager items or `Ids`. For example:

```
<Presentation Id="1">
```

For each `Presentation` element, you can specify the following subelements:

**Table 14-1 Presentation subelements**

Element	Description
Host	<p>The <code>Host</code> element identifies a host that should be part of the <code>Presentation</code>. It is possible to have multiple <code>Host</code> elements for one <code>Presentation</code>. Each <code>Host</code> element has the following attributes:</p> <ul style="list-style-type: none"> <li>• <code>Domain</code> — The domain of the Web site. The default domain is <code>localhost</code>.</li> <li>• <code>Port</code> — The port of the Web site. The default port is <code>80</code>.</li> <li>• <code>Path</code> — The path of the virtual directory or a subset of your Web site. The default path is <code>/</code>.</li> <li>• <code>Protocol</code> — The protocol used for the Web site. The default protocol is <code>http</code>.</li> </ul> <pre>&lt;Host Domain="www.visitorsWeb.com" Port="80" Protocol="http" Path="/"&gt;</pre>
Personalization	<p>The <code>Personalization</code> element enables profiling. This applies to both <code>Tracking Keys</code> and to <code>Tracking</code>. The <code>Personalization</code> element has the following attributes:</p> <ul style="list-style-type: none"> <li>• <code>Enabled</code> — Turns the tracking on or off, where <code>true</code> is on and <code>false</code> is off.</li> <li>• <code>Persistence</code> — This should be set to <code>Cookies</code>. This will ensure that a cookie is created when a visitor goes to your site. The WAI will generate a cookie.</li> </ul> <pre>&lt;Personalization Enabled="true" Persistence="Cookies"&gt;</pre>
Cookie	<p>The <code>Cookie</code> element specifies properties for cookies and has the following attributes:</p> <ul style="list-style-type: none"> <li>• <code>Name</code> — Name of the cookie</li> <li>• <code>Expires</code> — The expiration time of the cookie measured in seconds.</li> </ul> <p>A cookie with a unique value identifies each visitor. These cookies are generated by the WAI on the Content Delivery server. When a visitor visits your site for the first time, the code on the Page invokes the WAI that checks if the visitor has a cookie for the site. If the visitor has no cookie, the WAI generates a user record that is persisted to the user record with a cookie.</p> <pre>&lt;Cookie Name="TDS1234567" Expires="39000"/&gt;</pre>

**Table 14-1 Presentation subelements (Continued)**

Element	Description
Timeframe	<p>The <code>Timeframe</code> element determines the time increment during which items are tracked for Tracking. This <code>Timeframe</code> element has the following attributes:</p> <ul style="list-style-type: none"> <li>• <code>Type</code> — The type of timeframe: <code>hourly</code>, <code>daily</code>, <code>weekly</code>, or <code>monthly</code>.</li> <li>• <code>Multiplier</code> — Influences the timeframe type by adding a multiplier. For example, if the <code>Timeframe</code> type is <code>hourly</code> and the <code>Multiplier</code> is 2, the period over which visitor behavior is tracked is two hours.</li> <li>• <code>Autofill</code> — Indicates if missing timeframes should be created or not. Timeframes can be missing if the server has been offline for a certain period of time.</li> </ul> <p>The tracking mechanisms are based on counters, which are increased by each request for the tracked item. The usage of a counter requires granularity to make the tracking information useful.</p> <p>See "<i>Timeframes</i>" on page 131 for more detailed information about the <code>Timeframe</code> element.</p> <pre>&lt;Timeframe Type="hourly" Multiplier="2" Autofill="false"/&gt;</pre>
Page	<p>The <code>Page</code> element enables you to track Page requests, where <code>true</code> is enabled and <code>false</code> is disabled.</p> <pre>&lt;Page Enabled="true"/&gt;</pre>
Component	<p>The <code>Components</code> element enables the tracking of Component visits. The <code>Components</code> element has the following attributes:</p> <ul style="list-style-type: none"> <li>• <code>Enabled</code> — <code>true</code> is enabled and <code>false</code> is disabled. If this value is enabled, Components are tracked.</li> <li>• <code>Max</code> — The maximum number of Components that are allowed to be on a Page for the Components to be tracked.</li> <li>• <code>Averaging</code> — <code>true</code> is enabled and <code>false</code> is disabled. If Averaging is enabled, a Component Tracking key is increased by an average based on the number of Components on the Page. For example, if 4 Components exist on the Page, the tracking is increased by 1/4.</li> </ul> <pre>&lt;Components Enabled="true" Max="5" Averaging="true"/&gt;</pre>
ComponentLinks	<p>The <code>ComponentLinks</code> element enables tracking of Component Links.</p> <p>To enable tracking in Component Links, set the <code>Enabled</code> attribute to <code>true</code>.</p> <p><b>Note:</b> You must also set the <code>AddComponentLinkInfo</code> attribute of the <code>Linking</code> element in the linking configuration file (<code>cd_link_conf.xml</code>) to <code>true</code>.</p> <pre>&lt;ComponentLinks Enabled="true"/&gt;</pre>



**Table 14-1 Presentation subelements (Continued)**

Element	Description
Keys	<p>The <code>Keys</code> element contains Tracking Key attributes that allow you to determine how your Tracking Keys are generated:</p> <ul style="list-style-type: none"> <li>• <code>Enabled</code> — This element enables Tracking Keys to be incremented or decremented in the database</li> <li>• <code>Increment</code> — This is the amount that a Tracking Key is increased</li> <li>• <code>Decrement</code> — This is the amount that a Tracking Key is decremented when another Tracking Key is incremented</li> <li>• <code>Averaging</code> — If you have multiple Tracking Keys on a Page, the increment value is increased by the average value of the number of Tracking Keys on the Page</li> <li>• <code>ComponentLinks</code> — Uses the Component Link information to only increment the Tracking Keys for that specific Component</li> </ul> <pre>&lt;Keys Enabled="true" Increment="50" Decrements="1" Averaging="false" ComponentLinks="true"/&gt;</pre>
Exclude	<p>The <code>Exclude</code> element allows you to exclude Pages, Components, and Paths from Tracking Keys or Tracking. To do so, identify the following subelements:</p> <ul style="list-style-type: none"> <li>• <code>Pages</code> — Exclude Pages by Id</li> <li>• <code>Components</code> — Exclude Components by Id</li> <li>• <code>Paths</code> — Exclude paths by identifying the path</li> </ul> <p>For more detailed information about Exclude options, refer to <i>"Excluding Pages, Components or Paths from being tracked"</i> on page 131.</p> <pre>&lt;Exclude&gt; &lt;Pages&gt;1;2;4-100;125&lt;/Pages&gt; &lt;Components&gt;7&lt;/Components&gt; &lt;Paths&gt;**/news/*;/jobs/**&lt;/Paths&gt; &lt;/Exclude&gt;</pre>
Customer Characteristics	<p>If you gather information from your customers using web forms, you can use the <code>CustomerCharacteristics</code> element to determine if submitted values should be trimmed. Trimming refers to the removal of leading or trailing spaces, tabs, or returns that a visitor may have entered into a web form and that you may not want to preserve in your database.</p> <p>If the <code>PreserveWhiteSpace</code> attribute is set to <code>true</code>, the value will not be trimmed, if the element is set to <code>false</code>, the values are trimmed.</p> <pre>&lt;CustomerCharacteristics PreserveWhiteSpace="true"/&gt;</pre>

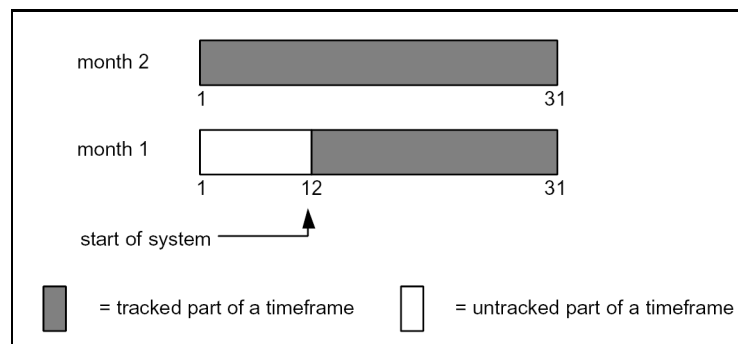
### 14.4.c Timeframes

Timeframe types are related to the calendar. That is, a monthly timeframe always starts at the first of the month, and ends on the last day of the month. Table 14-2 lists the available timeframes:

**Table 14-2 Timeframes**

Timeframe	Start	End
Monthly	1st day of the month	Last day of the month
Weekly	Monday	Sunday
Daily	0:00 hours	23:59 hours
Hourly	1 minute of the hour	60th minute of the hour

If the system starts on a date other than the first of the month, and the specified timeframe is monthly, a new timeframe is started for that specific month, but the start date, for example, the 12th, is included. See figure 14-6 for more information:



**Figure 14-6 Tracking and timeframes**

The end date for the timeframe is also included.

You can also use a multiplier in combination with the timeframe. This makes it possible to track every two or four hours for example. The multiplier must be a number between 1 and the maximum amount of logical multipliers of that timeframe. That is, if the timeframe is hourly, this multiplier is a number between 1 and 24.

**Note** Decimal multipliers are not allowed.

After a timeframe has expired, the active tracking record is archived and a new record is created to store information for the new timeframe.

If you change the timeframe configuration, you must restart the server. The system checks for new timeframes after every reboot. If the timeframe configuration is different from the last stored record, the system backs up the previous tracking record and begins a new record.

### 14.4.d Excluding Pages, Components or Paths from being tracked

To exclude Pages or Components by ID, you can specify ranges using hyphens (-) and entries semi-colons (;).

## Chapter 14 Profiling and personalization

You can exclude one or more paths from the tracking framework using patterns. These patterns can be a combination of strings and special wildcard indicators, such as '\*\*', '\*' and "?". Use a semi-colon to separate the paths.

The name to be matched is split up in path segments. A path segment is the name of a directory or file, bounded by /.

For example, `abc/def/ghi/xyz.java` is made up of the segments `abc`, `def`, `ghi` and `xyz.java`.

The segments of both the actual path name, and the specified exclusion pattern, are compared. If the pattern matches the existing path, the path, or files contained there, depending on what exclusion is specified, is excluded from tracking.

The following are some exclusion rules:

- When `**` is used for a path segment in the pattern, then it matches zero or more path segments of the name.
- If an exclusion pattern starts with a `/`, the string to compare against must also start with a `/`.
- If an exclusion pattern does not start with a `/`, the string to compare against must not start with a `/`.
- When a name path segment is matched against a pattern path segment, the following special characters can be used:
  - ▶ `*` matches zero or more characters
  - ▶ `?` matches one character.

Table 14-3 lists some exclusion pattern examples:

**Table 14-3 Examples of Exclusion patterns.**

Pattern	Matches
<code>**/*.jsp</code>	All <code>.jsp</code> files in a directory tree.
<code>test/a???.jsp</code>	All files which start with an <code>a</code> , two more characters and then <code>.jsp</code> , in a directory called <code>test</code> .
<code>**</code>	Everything in a directory tree.
<code>**/test/**/XYZ*</code>	All files, or directories, that start with <code>XYZ</code> and reside in a parent directory called <code>test</code>



## 14.5 Personalizing content

You can personalize the content that visitors are shown based on profiling information that you have about your visitors.

Personalized content is based on Target Groups. A Target Group defines visitor groups based on Tracking Key information that you have obtained from your visitor and possibly Customer Characteristics that you have obtained directly from a visitor through a web form.

**Note** Tridion products do not explicitly enable you to gather Customer Characteristics. Customer Characteristics usually consist of name value pairs. The WAI API enables you to use Customer Characteristics that you have gathered about specific visitors in conjunction with Tracking Keys in order to personalize web content. You can store Customer Characteristics on a Content Broker SQL database. Customer Characteristics are associated with a specific user ID.

To personalize content, you need to perform the following tasks:

- Define Target Groups in the Content Manager
- Associate Target Groups with Component Presentations on a Page

If you perform these tasks, known visitors will shown content based on the information that you gathered about them.

### 14.5.a Creating Target Groups

You create Target Groups in the Tridion Content Manager. When you add a Target Group, you specify the Customer Characteristics and/or Tracking Key values obtained from Customer Characteristics and Tracking Keys information. In addition, you can include Target Groups in another Target Group.

For example, a site features software reviews. You would like to show visitors that have indicated that they have interest in Graphic Design software a summary of new Mac OS graphic design products when then access your Review page. To do so, you can create a Target Group with the following definition:

- Customer Characteristics: You base these characteristics on information that you have obtaining information directly from your visitors through a Web form:
  - ▶ Profession = Graphic Designer
  - ▶ Operating System = Mac OS
- Tracking Keys:
  - ▶ Graphics and Publishing > 5
  - ▶ Mac OS > 5

In this case, you have created a Target Group that applies to Graphic Designers that work on Mac Operating Systems, and that have accessed Graphics and Publishing articles and Mac OS articles more than five times.

To create a Target Group, you must have Customer Management rights within the Publication in which you are creating a Target Group.

When you add a Target Group, you create a Target Group definition that identifies:

- Customer Characteristics
- Tracking Keys
- Target Group

### To create a Target Group:

- 1 In CM Explorer, navigate to the Publication and Folder in which you want to create a Target Group.
- 2 On the toolbar, click the **Add Target Group** button. A New Target Group window appears.
- 3 On the **General** tab, fill in the following fields:
  - **Name** — the name of the Target Group used in Tridion R5
  - **Description** — a description of the Target Group
- 4 On the **Definition** tab, you can add Characteristics, Tracking Keys, and Target Groups:
  - To add a Characteristic, click the **Characteristics** subtab, type a value in the first field, select an operator, and type an operand. Click Include or Exclude. (See also "*Defining Customer Characteristics*" on page 134)
  - To add a Tracking Key, click the **Tracking Keys** subtab, select a Category, select a Keyword, select an operator, and type a value. Click Include or Exclude. (See also "*Defining Tracking Keys*" on page 135)
  - To add a Target Group, click the **Target Group** subtab and select a previously created Target Group.
- 5 Click the **Save & Close** button on the tool bar.

**Results** You have created a Target Group. To apply these Target Group conditions to Pages, you must associate applicable Component Presentations on a Page with this Target Group.

### Defining Customer Characteristics

In a Target Group, you can specify visitor characteristics that you will use to determine whether content is displayed to a visitor.

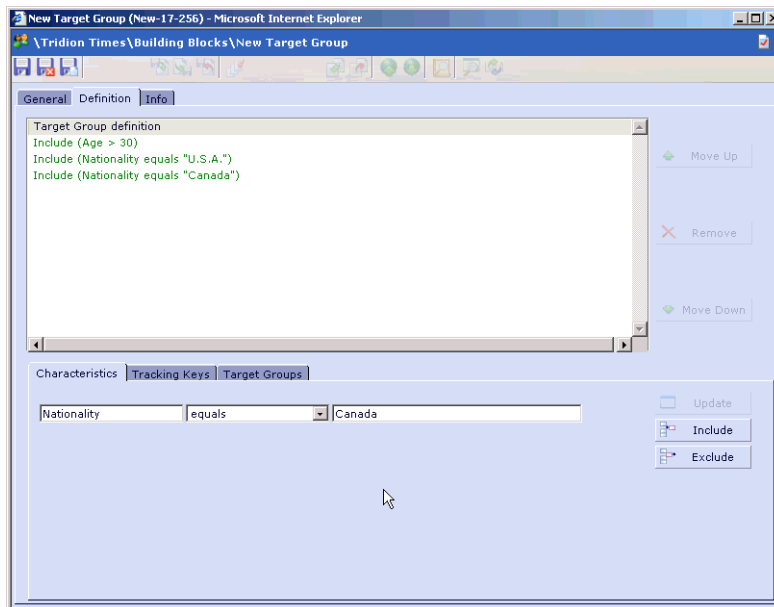
**Table 14-1 Characteristic operands.**

Operand	Use	Example
=	the tracked numeric characteristic is the same as the specified value	Year of birth = 1968
<	the tracked numeric value is less than the specific value	Age > 35
>	the tracked numeric value is greater than the specified value	Age < 18
<>	the tracked numeric value does not equal the specified value	Income <> 10000
equals	the tracked alpha-numeric value is the same as the specified value	Profession equals Technical Writer

**Table 14-1 Characteristic operands.**

Operand	Use	Example
contains	the tracked alpha-numeric value contains the specified value	Interests contain New Releases
starts with	the tracked alpha-numeric value starts with the specified value	Family Name starts with F
ends with	the tracked alpha-numeric value ends with the specified value	Last Version used ends with SP3

You can choose to include or exclude each of the values specified as a characteristic. For example, you can exclude all visitors for whom the profession equals Technical Writer and include all visitors for whom the value of Profession equals Software Developer.



**Figure 14-7 Characteristics**

### Defining Tracking Keys

When you specify a Tracking Key for a Target Group, you identify the whether or not a visitor should view Component Presentations based on their past visiting behavior when a visitors interest in a specific topic is shown by their visiting behavior. In this case, you have obtained the visitor behavior using Tracking Keys and have already associated published content with Categories and Keywords.

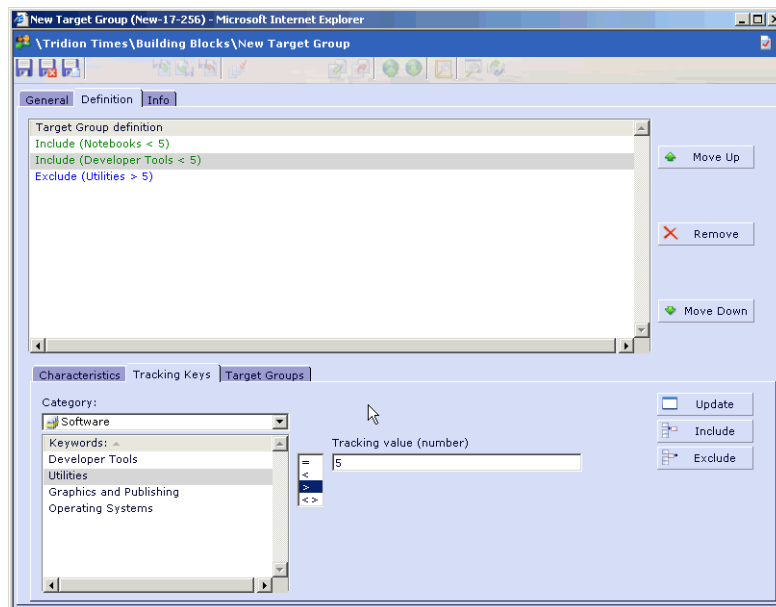
For example, the visitor has accessed a Page that contains Component Presentations that have the keyword value Graphic Design more than 5 times, the visitor will then be shown Component Presentations that are about Graphic Design.



When you specify a Tracking Key condition, you identify the Category, associated Keyword, operand, and value. The table below depicts the types of operands that can be used for Tracking Keys.

**Table 14-2 Operands**

Operand	Use	Example
=	the tracked Category and Keyword has been accessed the exact number of times specified	Software Operating Systems = 3
<	the tracked Category and Keyword has been accessed less times than the number specified	Windows < 3
>	the tracked Category and Keyword has been accessed more than the number specified	Mac OS > 3
<>	the tracked Category and Keyword does not equal the number specified	Linux <> 3



**Figure 14-8 Tracking Keys**

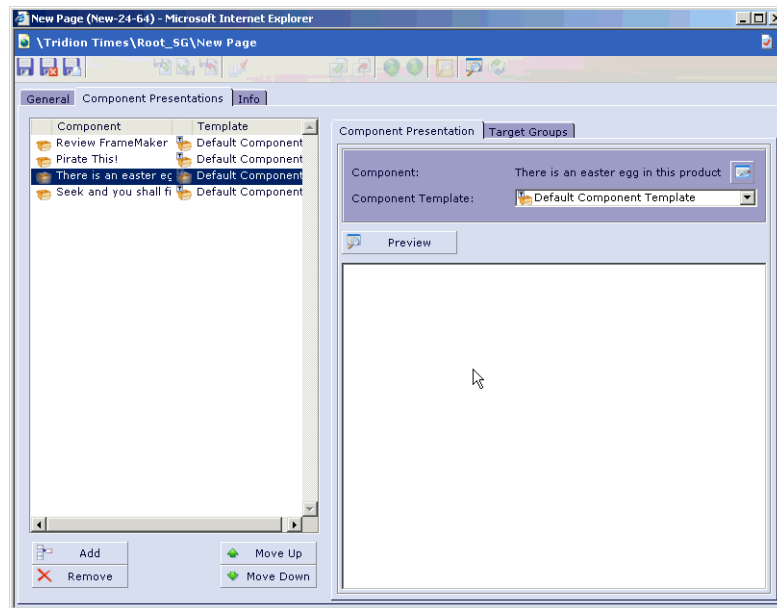
### 14.5.b Associating Target Groups with Component Presentations

After you have created a Target Group, you can specify the Target Group within a Component Presentation on a Page.

#### To specify a Target Group for a Component Presentation:

- 1 In CM Explorer, create or edit a Page.
- 2 On the **Component Presentation** tab, select the Component Presentation for which you want to set Target Groups.



**To specify a Target Group for a Component Presentation: (Continued)**

- 3** Click the **Target Group** tab.
- 4** To set a Component Presentation as viewable by a specific Target Group, select the Target Groups for the specified Component Presentation.
- 5** To exclude a Component Presentation as viewable by a specific Target Group, select Target Groups to exclude.
- 6** To make this Component Presentation viewable by all Target Groups, select Everyone.
- 7** **Save and Close** the Page.

**Results** When the Page is published, the conditions applied by the Target Groups will make applicable content available to the visitors that fulfill these conditions.







# Chapter 15 Dynamic content assembly

You can use the Tridion Web and Application Server Integration (WAI) to Dynamically assemble published content.

Dynamic assembly is possible when you use Dynamic Component Templates. When you use Dynamic Component Templates, Component Presentations are published separately from Pages.

You can use Dynamic Component Presentations (the combination of a Component and a Dynamic Component Template) to publish content in the following ways:

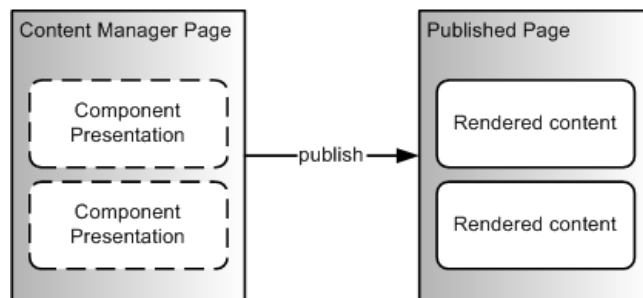
- Dynamic Component Presentations can be added to a Page and the Page can be published. The Dynamic Component Presentation is published separately from the Page. The rendered Page includes a line of code that is executed by the Content Distributor, which then retrieves the associated Component Presentation.
- Dynamic Component Presentations can be published completely separately from Pages and can then be retrieved using queries or filters.

This chapter describes:

- Using Dynamic Component Presentations
- Creating Component Templates for dynamic content assembly
- Publishing Pages that include Dynamic Component Presentations
- Configuring the Content Broker for Dynamic Component assembly
- Assembling Dynamic Component Presentations
- Assembling Dynamic JSP Component Presentations
- Assembling Dynamic ASP Component Presentations
- Dynamic Assembly XML

## 15.1 Using Dynamic Component Presentations

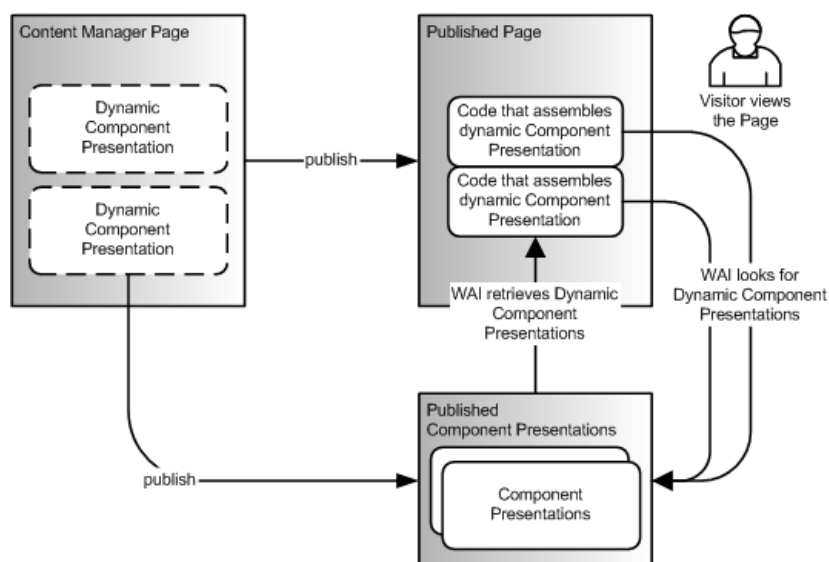
By default, the Content Manager allows you to create Pages and to add Component Presentations to Pages. These Component Presentations are the combination of content (Components) and content format (Component Templates). Published Pages that contain these Component Presentations include the content directly on the Page. To update any modified Components, you must republish the entire Page including any unchanged Component Presentations.



**Figure 15-1 Content published directly on a Page**

If a Page is published that contains Dynamic Component Presentations, when the Page is published Component Presentations are published to a separate content repository on the Content Broker. The Page is published and code is generated within the Page that identifies the location of the Dynamic content. When a visitor views the Page, the generated code in the Page retrieves the Component Presentation from the content repository.

If a published Dynamic Component Presentation is modified and republished, only the Dynamic Component Presentation is republished, rather than the entire Page.

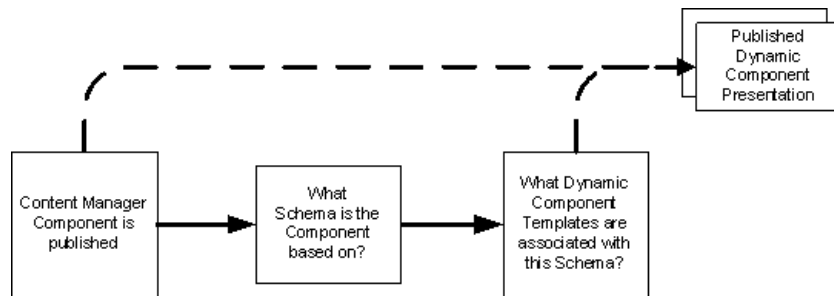


**Figure 15-2 Publishing Pages that have Dynamic Component Presentations**



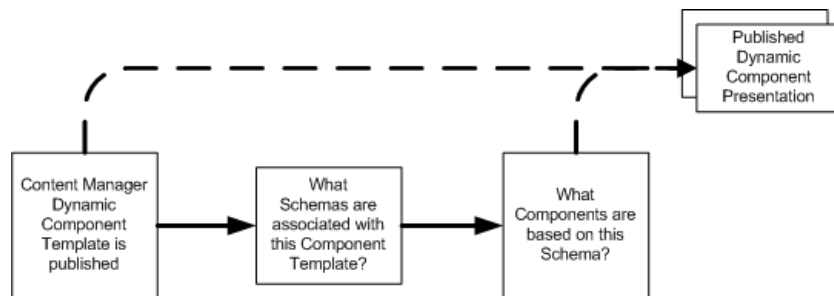
To publish Dynamic Component Presentations separately, you can publish a Component that is based on an allowed Schema for a Dynamic Component Template or publish a Dynamic Component Template:

- If you publish a Component that is based on a Schema associated with a Dynamic Component Template, a Dynamic Component Presentation that combines the Component and Component Template is published



**Figure 15-3 Publishing a Component to create Dynamic content**

- If you publish a Dynamic Component Template, Components that are based on Schemas associated with the template are published as Dynamic Component Presentations that combine the Dynamic Component Template and the Component



**Figure 15-4 Publishing a Component Template to create Dynamic content**

The Dynamic Component Presentation is published to the content repository.

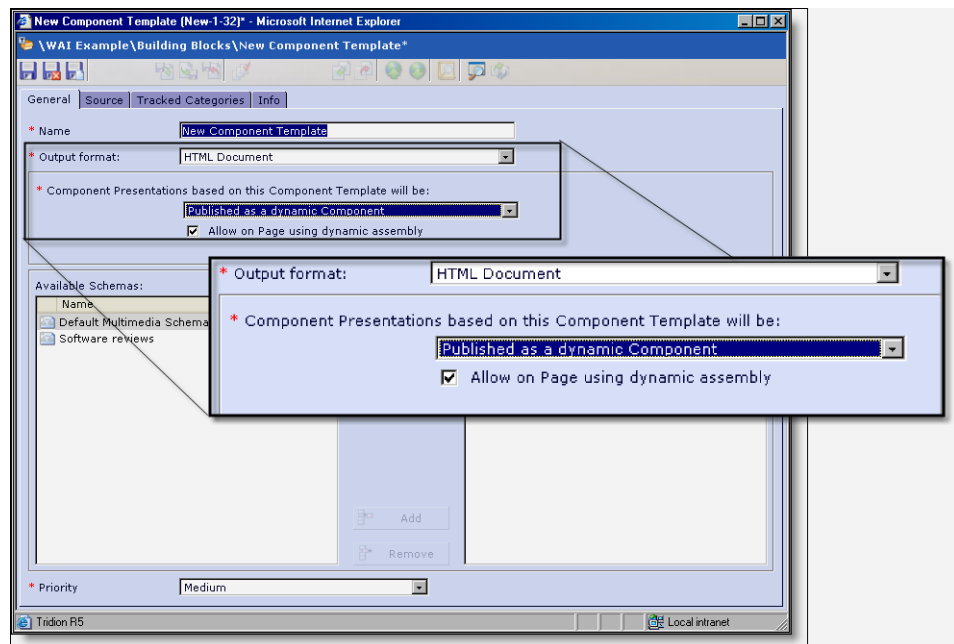
## 15.2 Creating Component Templates for dynamic content assembly

You create Component Templates used for dynamic content assembly in the Content Manager.

The following Component Template settings used for dynamic assembly are described in this section:

- Output format
- Publishing as Dynamic content





**Figure 15-5 Component Template settings used for Dynamic assembly**

### 15.2.a Output format

When you create a Component Template that can be used to create Dynamic Component Presentations, you can select a number of output formats. Some output formats have a functional impact throughout the Content Delivery system, whereas other output formats allow developers to take appropriate actions based on the type of output.

The high level distinction enables the WAI to process dynamic Component Presentations for that specific type:

- Text based — assembled as raw text without further processing. This content is neither rendered nor processed but is Dynamically embedded in the Page at request time.
- ASP — executed by the Windows Scripting Host in an ASP scripting context
- JSP — executed as a small JSP pages in the web container.
- XML — if available, an XSLT transformation is applied

Text based output includes HTML (fragments and documents), plain text, and other. ASP includes ASP VB Script and JScript. JSP contains JSP scripting code. XML includes documents and fragments. The following output formats are available:

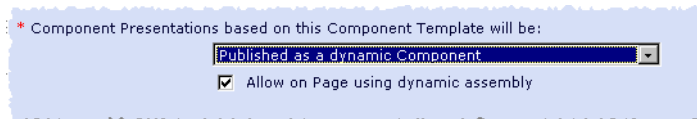
- HTML (fragment or document)
- Plain text
- Other
- ASP (VB Script or JScript)
- JSP Scripting
- XML (fragment or document)



### 15.2.b Publishing Dynamic Component Presentations

To produce Dynamic Component Presentations, you must select the following option in your Component Template:

**Component Presentations based on this Component Template will be:  
Published as a Dynamic Component**



**Figure 15-6 Component Template Published as Dynamic Component**

This setting ensures that Components that are published using this Component Template are published as Dynamic Component Presentations. These Dynamic Component Presentations are published to a content repository in the Content Broker database.

If you select the option "Allow on Page using Dynamic assembly", Component Presentations can be added to a Page but are still published as Dynamic Component Presentations to the Broker's content repository.

When the Page is published, a line of code is added to the Page that points to the Component Presentation. This statement is executed when a visitor views the Page. The Component Presentation is then retrieved from the Content Broker database.

If you do not select "Allow on Page using Dynamic assembly" you cannot use the Component Template on a Page, however can still publish the Component Presentation to the Broker repository. To publish Components using this type of Dynamic Component Template you can either:

- Publish a Component based on an associated Schema for this Component Template (see figure 15-3)
- Publish the Component Template to publish all Components based on the associated Schema for this Component Template (see figure 15-4)

### 15.2.c Publishing Pages that include Dynamic Component Presentations

Pages that include Dynamic Component Presentations should be published to Publication Targets for which the Target Language is ASP, ASP.NET, or JSP since dynamic assembly requires server side scripting.



## 15.3 Configuring the Content Broker for Dynamic Component assembly

You configure the location to which the Dynamic Component Presentations are published in the Content Broker configuration file (`cd_broker_conf.xml`).

The Broker bindings determine where these files are stored for each type (ASP, JSP, XML, Text based). Broker bindings are described in "*Bindings*" on page 70.

When storing dynamic Component Presentations on the file systems, the location of the different types of Dynamic Component Presentation output can be specified on a Publication level. For more information, refer to the same configuration file (`cd_broker_conf_sample.xml` on the installation CD-ROM).

**Note** You can also configure the Java Virtual Machine that the Tridion Web and Application Server Integration service starts. See Appendix J, "*Configuring the Java Virtual Machine for COM services*" for details.

## 15.4 Assembling Dynamic Component Presentations

The WAI has a public API that you can use to assemble Dynamic content.

For information on the WAI API, refer to the following API references:

- `WAI_ASP_51SP4.chm` — ASP API reference file
- `WAI_JSP_51SP4.zip` — HTML JavaDoc delivered with your installation set
- `ContentDelivery_NET_51SP4.chm` — .NET API reference file

The following sections provide examples of how to assemble Dynamic Component Presentations:

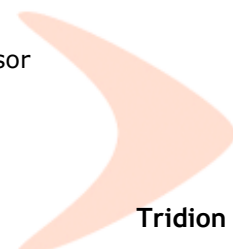
- `ComponentPresentationAssembler`
- `ComponentPresentation.getContent()`

### 15.4.a ComponentPresentationAssembler

The `ComponentPresentationAssembler` class assembles Dynamic Component Presentations on JSP or ASP Pages.

This class performs the following actions:

- Reads the metadata of the Component Presentations OR
- If the content is HTML, no processing is required since the content is processed by the HTML itself and the `ComponentPresentationAssembler` outputs the content directly.
- If the content is XML, ASP, or JSP, the `ComponentPresentationAssembler` delegates the content to processors:
  - ▶ XML with an XSLT is processed by the XSLT processor
  - ▶ ASP is processed by the ASP Processor
  - ▶ JSP is processed by the JSP Processor



For more information about invoke the `ComponentPresentationAssembler` refer to:

- "Using the `ComponentPresentationAssembler` in JSP" on page 146
- "Using the `ComponentPresentationAssembler` in ASP" on page 150

### 15.4.b `ComponentPresentation.getContent()`

You can use `ComponentPresentation.getContent()` to retrieve the raw content of the Component Presentation. You can retrieve a Component Presentation instance using the Component Presentation factory. This factory has multiple methods to create Dynamic Component Presentations.

#### ASP

```
<%
Dim ComponentPresentationFactory, ComponentPresentation
Set ComponentPresentationFactory =
Server.CreateObject("cd_broker.ComponentPresentationFactory")
Set ComponentPresentation =
ComponentPresentationFactory.GetComponentPresentation("tcm:1-234",
"tcm:1-345:32");
Response.Write ComponentPresentation.Content
%>
```

#### JSP

```
<%@ page import="com.tridion.dcp.*" %>
<% ComponentPresentationFactory cpf = new
ComponentPresentationFactory("tcm:0-1-1"); // Publication URI
ComponentPresentation componentPresentation =
cpf.GetComponentPresentation("tcm:1-234", "tcm:1-345:32");
// Component URI and Component Template URI
out.println(componentPresentation.getContent());
%>
```

## 15.5 Assembling Dynamic JSP Component Presentations

You can include Dynamic JSP Component Presentations if you use a JSP Web and Application server:

- Using the `JSPProcessor`
- Using the `ComponentPresentationAssembler` in JSP
- Using `DynamicComponentLink`

These Component Presentations should be published inside of the document root of your Web site.

This section describes:

- Using the `JSPProcessor`
- Using the `ComponentPresentationAssembler` in JSP
- Using `DynamicComponentLink`



### 15.5.a Using the JSPPProcessor

The JSPPProcessor executes JSP Dynamic Component Presentations by performing a server side include. This processor is located in the package `com.tridion.web.jsp`.

**Note** The assembled JSP Dynamic Component Presentations are written directly to the output stream and the JSPPProcessor returns nothing.

Passing a JSPPPage instance in the constructor creates a JSPPProcessor instance.

The following example shows how you can directly invoke the JSPPProcessor from a published Page:

#### Example

```
<%@ page import="com.tridion.web.jsp.*" %>
<%
JSPPPage waiPage = new JSPPPage(pageContext, "tcm:0-1-1");
JSPPProcessor jspProcessor = new JSPPProcessor(waiPage);
%>
```

The JSPPPage instance that is used in the constructor is passed to the Dynamic Component Presentations. The following code shows how you can access the JSPPPage instance from within a JSP dynamic Component Presentation that is being assembled:

#### Example

```
<%
JSPPPage waiPage = (JSPPPage)request.getAttribute("JSPPPage");
out.println(waiPage.getPresentation().getId());
%>
```

Because the JSPPPage is available, the Page and Publication ID can be requested from this object. These variables can be used for other functionality such as Linking.

#### Parameters

Since the web container executes the JSP Dynamic Component Presentation as a normal JSP Page, the following objects are also available:

- `javax.servlet.http.HttpServletRequest request`
- `javax.servlet.http.HttpServletResponse response`
- `javax.servlet.jsp.PageContext pageContext`
- `javax.servlet.ServletOutputStream out`

### 15.5.b Using the ComponentPresentationAssembler in JSP

You can use the ComponentPresentationAssembler to Dynamically add content of Dynamic Component Presentations to the content of your Page. The ComponentPresentationAssembler invokes handlers for JSP or XML/XSLT content.

The following example illustrates how to use the Component Presentation Assembler from a JSP Page:

#### Example

```
<%@ page import="com.tridion.web.jsp.JSPPPage,
com.tridion.web.jsp.ComponentPresentationAssembler"%>
<%
JSPPPage waiPage = new JSPPPage(pageContext, "tcm:28-834-64");
```

```

ComponentPresentationAssembler cpAssembler = new
ComponentPresentationAssembler(waiPage);
out.println(cpAssembler.getContent("tcm:28-780","tcm:28-821-32"));
%>

```

### 15.5.c Using DynamicComponentLink

The `DynamicComponentLink` validates links to Dynamic Component Presentations on Pages. The `DynamicComponentLink` validates links by making sure that the Page that is able to assemble Dynamic Component Presentations and the Dynamic Component Presentation are available in the Content Broker.

When `DynamicComponentLink` resolves a Link, the following parameters are added to the Query String:

- Component — The ID of the Component
- ComponentTemplate — The ID of the Component Template

`DynamicComponentLink` requires custom implementation.

The following examples shows:

- JSP Source Page — This Page invokes the `DynamicComponentLink`
- Target Page — This Page assembles a Dynamic Component Presentation based on the parameters in the query string
- Output — This shows the result

#### Source Page

##### Example

```

<%@page import="com.tridion.linking.DynamicComponentLink,
com.tridion.linking.Link"%>
<%!
final static String PUBLICATION_URI="tcm:0-2-1";
final static String COMPONENT_URI="tcm:2-34";
final static String COMPONENT_TEMPLATE_URI="tcm:0-0-0";
final static String PAGE_URI="tcm:2-328-64"; //this uri should point
to the DynamicComponentLinkDestination.jsp page.
final static String LINK_TEXT="This is a Dynamic component link!";
%>
<html>
<head>
<title>Dynamic Component Link Source Page</title>
</head>
<body>
Dynamic Component Link to componentpresentation
(componentId=<%=COMPONENT_URI%>,
componentTemplateId=<%=COMPONENT_TEMPLATE_URI%>) on page
<%=PAGE_URI%> in publication <%=PUBLICATION_URI%>:
<%
DynamicComponentLink dcl = new DynamicComponentLink(PUBLICATION_URI);
Link link = dcl.getLink(PAGE_URI, COMPONENT_URI,
COMPONENT_TEMPLATE_URI, "bla", LINK_TEXT, true);
out.println(link.toString());
%>
</body>
</html>

```

**TargetPage***Example*

```

<%@ page import="com.tridion.web.jsp.ComponentPresentationAssembler,
com.tridion.web.jsp.JSPPage"%><%!

//Constants:
final static String PUBLICATION_URI="tcm:0-42-1";
final static String PAGE_URI="tcm:42-3456-64";

%><%

//Get parameters from Request:
String componentURI = request.getParameter("Component");
String componentTemplateURI =
request.getParameter("ComponentTemplate");

%><html>
<head>
<title>Dynamic Component Link Destination Page</title>
</head>
<body>
Assembled Component:
<%

JSPPage waiPage = new JSPPage(pageContext, PAGE_URI);
ComponentPresentationAssembler cpa = new
ComponentPresentationAssembler(waiPage);
out.println(cpa.getContent(componentURI, componentTemplateURI));

%>
</body>
</html>

```

**Output**

The Source Page outputs the following:

*Example*

```

<a
href="/thepage.xsp?Component=234&ComponentTemplate=345">LinkText</a>

```

Assembled Component:

*Example*

```

<html>
<head>
<title>Dynamic Component Link Destination Page</title>
</head>
<body>
Assembled Component:
<b>This is the content of the component presentation!</b>
</body>
</html>

```





## 15.6 Assembling Dynamic ASP Component Presentations

You can include Dynamic Component Presentations using JSP. This section describes:

- Executing VBScript or JScript Dynamic Component Presentations
- Using the ComponentPresentationAssembler in ASP
- Using DynamicComponentLink

### 15.6.a Executing VBScript or JScript Dynamic Component Presentations

The `ASPPProcessor` executes VBScript and JScript Dynamic Component Presentations using IIS and ASP. After constructing the Component Presentations, the `Run`, or `Execute`, method is called to execute the ASP Dynamic Component Presentation. This is shown in the sample below:

#### Example

```
<%  
Dim ASPPage, ASPPProcessor  
Set ASPPage = Server.CreateObject("cd_wai.ASPPage")  
Call ASPPage.Init("tcm:1-345-64")  
Set ASPPProcessor =  
Server.CreateObject("cd_wai.ASPPProcessor")  
Call ASPPProcessor.Run(ASPPage, "tcm:1-234",  
"tcm:1-456-32")  
%>
```

#### Parameters

An ASP Dynamic Component Presentation is executed in a similar way as a normal ASP Page, therefore the following objects are available:

- `IRequest Request`
- `IResponse Response`
- `IServerObject Server`
- `IApplicationObject Application`
- `ISession Session`



### 15.6.b Using the ComponentPresentationAssembler in ASP

The ComponentPresentationAssembler allows you to Dynamically add content of Dynamic Component Presentations to the content of your Page. The ComponentPresentationAssembler invokes handlers for specific types of content, like ASP or XML/XSLT content.

The following example illustrates how to use the Component Presentation Assembler from an ASP Page:

#### Example

```
<%
Dim cpAssembler
Set cpAssembler =
Server.CreateObject("cd_wai.ComponentPresentationAssembler")
Response.Write(cpAssembler.getContent("tcm:22-225","tcm:22-227-32"))
Set cpAssembler = Nothing
%>
```

### 15.6.c Using DynamicComponentLink

The DynamicComponentLink validates links to Dynamic Component Presentations on Pages. The DynamicComponentLink validates links by making sure that the Page that is able to assemble Dynamic Component Presentations and the Dynamic Component Presentation are available in the Content Broker.

When DynamicComponentLink resolves a Link, the following parameters are added to the Query String:

- Component — The ID of the Component
- ComponentTemplate — The ID of the Component Template

DynamicComponentLink requires custom implementation.

The following examples shows:

- ASP Source Page — This Page invokes the DynamicComponentLink
- Target Page — This Page assembles a Dynamic Component Presentation based on the parameters in the query string
- Output — This shows the result

#### SourcePage

#### Example

```
<%
const PUBLICATION_ID=2
const COMPONENT_ID=34
const COMPONENT_TEMPLATE_ID=316
const PAGE_ID=328
const LINK_TEXT="This is a Dynamic component link!"

Dim DynamicComponentLink, Link
%>
<html>
<head>
<title>Dynamic Component Link Source Page</title>
</head>
<body>
```



```

Dynamic Component Link to componentpresentation
(componentId=<%=COMPONENT_ID%>,
componentTemplateId=<%=COMPONENT_TEMPLATE_ID%>) on page <%=PAGE_ID%>
in publication <%=PUBLICATION_ID%>:
<%
Set DynamicComponentLink =
Server.CreateObject("cd_wai.DynamicComponentLink")
Set Link = DynamicComponentLink.GetLink(PUBLICATION_ID, PAGE_ID,
COMPONENT_ID, COMPONENT_TEMPLATE_ID, LINK_TEXT, True, "")
Response.Write link.ToString()
%>
</body>
</html>

```

### TargetPage

#### Example

```

<%
const PUBLICATION_ID=2

Dim ComponentPresentationAssembler, componentId, componentTemplateId
//Get parameters from Request:
componentURI = CLNG(Request("Component"))
componentTemplateURI = CLNG(Request("ComponentTemplate"))

%>
<html>
<head>
<title>Dynamic Component Link Destination Page</title>
</head>
<body>
Assembled Component:
<%
Set ComponentPresentationAssembler =
Server.CreateObject("cd_wai.ComponentPresentationAssembler")
Response.Write(ComponentPresentationAssembler.GetContent
(componentId, componentTemplateId, PUBLICATION_ID))
%>
</body>
</html>

```

### Output

The Source Page outputs the following:

**Example** <a href="/thepage.xsp?Component=234&ComponentTemplate=345">LinkText</a>

Assembled Component:

#### Example

```

<html>
<head>
<title>Dynamic Component Link Destination Page</title>
</head>
<body>
Assembled Component:
<b>This is the content of the component presentation!</b>
</body>
</html>

```

## 15.7 Dynamic Assembly XML

If you select XML Document, you can also add a Dynamic Template (XSLT) to the Component Template. XML Dynamic Component Presentations are processed at assembly time using the XSLT supplied in the Dynamic Template of the Component Template.

This section describes how to use XSLTs on your XML output in the WAI.

XSLT can only be applied when the XSLT (Dynamic Template) for a Component Presentation is available in the Content Broker.

XSLT can be applied using an XSLTProcessor. This XSLTProcessor takes a Component and a Component Template as a parameter. The Component is used to locate the content, and the Template is used to locate the Dynamic template. The content is generally returned as a String.

The usage of the XSLTProcessor is shown below:

### ASP

The ASP XSLTDynamic Component PresentationProcessor returns a string representation of the result of the transformation.

```
<%
Dim XSLTProcessor
Set XSLTProcessor = Server.CreateObject("cd_wai.Dynamic Component
PresentationXSLTProcessor")
Response.Write XSLTProcessor.GetStringFromXMLComponentPresentation
("tcm:2-34", "tcm:2-45-32")
%>
```

### JSP

The JSP XSLT Dynamic Component Presentation Processor returns a string representation of the result of the transformation.

```
<%@ page import="com.tridion.web.XSLTProcessor" %>
<%
XSLTProcessor xslt = new XSLTProcessor();
out.println(xslt.getString("tcm:2-34", "tcm:2-45-32"));
%>
```

### Parameters

The following parameters are passed to the stylesheet:

- publicationId
- pageId
- componentId
- componentTemplateId
- publicationURI
- pageURI
- componentURI
- componentTemplateURI

The following sample indicates how to access these parameters from XSL:

```
<xsl:stylesheet>
<xsl:param name="publicationId" select="-1" />

<xsl:template match="/">
The publicationId was: <xsl:value-of select="$publicationId" />
</xsl:template>
</xsl:stylesheet>
```

### Generating links from an XSLT using the Linking API

The following example shows using the Linking API from within XSLT:

**Note** This example is specific to the XML Dynamic Component Presentations stored in the Tamino XML database. With minor modifications it can also be used for raw XML Dynamic Component Presentations.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:ino="http://namespaces.softwareag.com/tamino/response2"
xmlns:xql="http://metalab.unc.edu/xql/"
xmlns:lxslt="http://xml.apache.org/xslt"
xmlns:tcd="http://www.tridion.com/tcd"
extension-element-prefixes="tcd">
<xsl:output method="html" encoding="UTF-8"/>
<xsl:param name="publicationURI" />
<xsl:param name="pageURI" />
<xsl:param name="componentTemplateURI" />
<lxslt:component prefix="tcd">
<lxslt:script lang="javaclass" src="xalan://com.tridion" />
</lxslt:component>

<xsl:template match="/">
<xsl:apply-templates select="Article"/>
</xsl:template>

<xsl:template match="Article">
<h1><xsl:value-of select="Title"/></h1>
<i><xsl:copy-of select="Intro"/></i>
<xsl:call-template name="ComponentLink">
<xsl:with-param name="componentURI"
select="@ino:docname"/>
<xsl:with-param name="linkText" select="'Read More'"/>
<xsl:with-param name="target"
select="../TaminoFilter/Detail/@href"/>
</xsl:call-template>
</xsl:template>

<xsl:template name="ComponentLink">
<xsl:param name="componentURI"/>
<xsl:param name="linkText"/>
<xsl:param name="target"/>
<xsl:variable name="componentLinkInstance"
select="tcd:linking.ComponentLink.new($publicationURI)"/>
<xsl:variable name="query"
select="concat('componentURI=', $componentURI)"/>
<xsl:variable name="link"
select="tcd:getLink($componentLinkInstance, $pageURI, $target,
```

```
$componentTemplateURI, '', $linkText, false, true)"/>  
<xsl:value-of select="tcd:setParameters($link, $query)"/>  
<xsl:value-of select="$link"  
disable-output-escaping="yes"/><br/>  
</xsl:template>  
</xsl:stylesheet>
```



# Appendices

---





## Appendix A Example: An SMTP Custom Sender

The following SMTP custom class defines the SMTP Sender:

```

package com.tridion.examples;

import com.tridion.transport.transportpackage.TransportPackage;
import com.tridion.configuration.*;
import com.tridion.logging.*;
import com.tridion.util.ZipUtils;
import java.io.*;
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class SMTPSender extends Sender {

    public final static String SMTPSERVER_CONFIG = "SMTPServer";
    public final static String USERNAME_CONFIG = "UserName";
    public final static String PASSWORD_CONFIG = "Password";
    public final static String TO_EMAIL_CONFIG = "EmailAddress";

    public final static String FROM_EMAIL = "transport@tridion.com";
    public final static String FROM_NAME = "Tridion Transport Service STMP
        Sender";
    public final static String SUBJECT = "Transport Package";

    private String host;
    private String userName;
    private String password;
    private String email;
    private static Logger log = LoggerFactory.getDefaultLogger();

    public SMTPSender(Configuration config) throws
        ConfigurationException {
        super(config);
        configure(config);
    }

    public void configure(Configuration configuration) throws
        ConfigurationException {
        host = configuration.getStringValue(SMTPSERVER_CONFIG, "localhost");
        userName = configuration.getStringValue(USERNAME_CONFIG, "anonymous");
        password = configuration.getStringValue(PASSWORD_CONFIG,
            "transport@tridion.com");
        email = configuration.getStringValue(TO_EMAIL_CONFIG,
            "Webmaster@localhost.com");
    }

    public void send(TransportPackage data) throws TransportException {
        try {
            log.debug("Zipping " + data.getLocation() + " to " + workingFolder);
            File zipFile = ZipUtils.createArchive(data.getLocation(),
                workingFolder.getFolder());

            Properties props = System.getProperties();
            props.put("mail.smtp.host", host);

```

```
Session session = Session.getDefaultInstance(props, null);

MimeMessage msg = new MimeMessage(session);
msg.setFrom(new InternetAddress(FROM_EMAIL, FROM_NAME));
InternetAddress[] to = {new InternetAddress(email)};
msg.setRecipients(Message.RecipientType.TO, to);
msg.setSubject(SUBJECT);

MimeBodyPart mbp = new MimeBodyPart();
FileDataSource fds = new FileDataSource(zipFile);
mbp.setDataHandler(new DataHandler(fds));
mbp.setFileName(zipFile.getName());

Multipart mp = new MimeMultipart();
mp.addBodyPart(mbp);

msg.setContent(mp);
msg.setSentDate(new Date());

javax.mail.Transport.send(msg);

catch (MessagingException me) {
    throw new TransportException("Could not complete SMTP transport :",
        me);
} catch (IOException ioe) {
    throw new TransportException("Could not complete SMTP transport :",
        ioe);
}
}
```

This custom class can then be used to configure the SMTP Sender in the transport configuration file (`cd_transport_conf.xml`).

For example, if you created an SMTP custom class, you can then add the class to the configuration file as follows:

```
<Senders>
  <Sender Type="SMTP" Class="com.tridion.examples.SMTPSender"/>
</Senders>
```

Create a protocol Schema for the type SMTP that contains the following fields:

- SMTPServer
- UserName
- Password
- EmailAddress

If content is published to the SMTP destination specified in a Publication Target, the Transport Service retrieves the values for the SMTP Sender from the Publication Target.

## Appendix B Example: Custom POP3 Receiver

The following POP3 custom class defines the POP3 Receiver:

```
package com.tridion.examples;

import com.tridion.configuration.*;
import com.tridion.logging.*;
import com.tridion.transport.transportpackage.TransportPackage;
import com.tridion.util.ZipUtils;
import java.io.*;
import java.util.*;
import javax.mail.*;

public class POP3Receiver extends Receiver {
    private static final String INTERVAL_CONFIG = "Interval";
    private static final String USERNAME_CONFIG = "User";
    private static final String PASSWORD_CONFIG = "Password";
    private static final String FOLDER_CONFIG = "Folder";
    private static final String SERVER_CONFIG = "Server";
    private String password;
    private String server;
    private String userName;
    private String folder;
    private float interval;

    int lastMessageCount = 0;

    private List packageQueue = Collections.synchronizedList(new
    LinkedList());
    private static Logger log = LogFactory.getDefaultLogger();

    public POP3Receiver(Configuration config) throws ConfigurationException {
        configure(config);
        try {
            Properties props = System.getProperties();

            // Get a Session object
            Session session = Session.getDefaultInstance(props, null);

            // session.setDebug(true);
            // Get a Store object

            Store store = session.getStore("pop3");
            // Connect
            store.connect(server, userName, password);
            // Open a Folder Folder folder = store.getFolder(folder);
            if (folder == null || !folder.exists()) {
                throw new ConfigurationException("Invalid folder for POP3Receiver");
            }
            folder.open(Folder.READ_WRITE);
            lastMessageCount = folder.getMessageCount();
            POP3Monitor monitor = new POP3Monitor(this, folder);
            new Thread(monitor).start();
            log.info("POP3Receiver " + this + " started monitoring");
        } catch (MessagingException me) {
```

```

throw new ConfigurationException("Could not connect to mailserver: " +
me.getMessage());
} catch (Throwable t) {
throw new ConfigurationException(t.getMessage());
}
public void configure(Configuration configuration)
throws ConfigurationException {
interval = configuration.getFloatValue(INTERVAL_CONFIG, (float)0.5);
userName = configuration.getStringValue(USERNAME_CONFIG, "transport");
password = configuration.getStringValue(PASSWORD_CONFIG, "transport");
server = configuration.getStringValue(SERVER_CONFIG, "localhost");
folder = configuration.getStringValue(FOLDER_CONFIG, "Inbox");
}
public TransportPackage listen() throws TransportException {
TransportPackage result = null;
try {
while (packageQueue.isEmpty()) {
//sleep for half the timer interval
Thread.sleep((long)(interval * 1000 / 2));
}
Object item = packageQueue.remove(0);
if (item instanceof Throwable) {
throw new TransportException("Error happened while waiting for incoming
package ",
(Throwable)item);
}
if (item instanceof TransportPackage) {
result = (TransportPackage)item;
}
}
}
catch (InterruptedException ie) {
}
return result;
}
void handleIncomingMessage(Message msg) {
File packageFile = null;
try {
// Unpack file to workFolder.
Object messageContent = msg.getContent();
if (messageContent instanceof Multipart) {
Multipart mp = (Multipart)messageContent;
int count = mp.getCount();
for (int i = 0; i < count; i++) {
Part part = mp.getBodyPart(i);
log.info("Receiving " + part.toString());
Object partContent = part.getContent();
if (partContent instanceof InputStream) {
InputStream is = (InputStream)partContent;
packageFile = new File(workingFolder.getFolder(), part.getFileName());
FileOutputStream fos = new FileOutputStream(packageFile);
byte[] buffer = new byte[1024];
int read = 0;
for ( ; read != -1; read = is.read(buffer)) {
log.info("Read bytes from attachment:" + read);
fos.write(buffer, 0, read);
}
}
}
}
}
}

```

```

}
fos.close();
is.close();
log.info("Copied attachment into file: " + packageFile);

}
}
//mark the message as deleted
msg.setFlag(Flags.Flag.DELETED, true);
msg.setFlag(Flags.Flag.SEEN, true);
try {
msg.getFolder().expunge();
} catch (MessagingException e) {
//not all providers support expunging...
}
}
ZipUtils.unzip(packageFile, workingFolder.getFolder());
// Create package messageContent.
int namePos = packageFile.getName().indexOf(".zip");
String packageName = packageFile.getName().substring(0, namePos);
TransportPackage data = new TransportPackage(
new File(workingFolder.getFolder(), packageName));
// Remove zip archive.
packageFile.delete();
// Insert into queue.
packageQueue.add(data);
} catch (IOException ioe) {
log.error("POP3Receiver:handleIncommingMessage got exception: " +
ioe.getMessage());
packageQueue.add(ioe);
} catch (MessagingException me) {
log.error("POP3Receiver:handleIncommingMessage got exception: " +
me.getMessage());
packageQueue.add(me);
}}class POP3Monitor implements Runnable {
Folder folder = null;
POP3Receiver Receiver = null;
POP3Monitor(POP3Receiver Receiver, Folder folder) {
this.folder = folder;
this.Receiver = Receiver;
}// The Monitor runs in it's own thread to receive messages independent of
// the Deployer processing loop
public void run() {
while (true) {
try {
int newMessageCount = folder.getMessageCount();
if (newMessageCount > 0) {
Message[] msgs = folder.getMessages();
for (int i = 0; i < msgs.length; i++) {
Flags f = msgs[i].getFlags();
if (!f.contains(Flags.Flag.SEEN)) {
Receiver.handleIncomingMessage(msgs[i]);
}
}
}
}
Thread.sleep(POP_CHECK_INTERVAL);
} catch (InterruptedException e) {
} catch (MessagingException e) {
log.error("Could not get message", e);
}
}
}

```

```
}  
}  
}  
}  
}
```

Add the custom Receiver to the Deployer configuration file  
(cd\_deployer\_conf.xml):

```
<ReceiverBindings>  
  <Receiver Type="POP3Receiver" Class="com.tridion.examples.POP3Receiver">  
</ReceiverBindings>  
<Receivers>  
  <TypeReceiver User="transport" Password="transport" Folder="Inbox"  
  Server="pop.vistorsWeb.com" Interval="1" />  
</Receivers>
```

## Appendix C Sample Transport Service configuration file

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- The Tridion Content Distributor Transport Service configuration specifies all
      configuration values required to initiate and handle a publishing action. -->
<TransportService>
<!-- The WorkFolder element defines the location where temporary files are stored. -->
<WorkFolder Location="./transactions"/>
<!-- The Logging section specifies the behaviour and location of one or more loggers. -->
<Logging>
<!-- Defines the default logger. The Level attribute specifies the amount of information to
      log.
      Legal logging levels:
      info - Information messages.
      warning - Potential error conditions.
      error - Errors that prevent the software from functioning correctly.
      fatal - Unrecoverable errors. -->
<Logger Level="error">
<!-- A FileLogger logs to a log file specified by the 'Location' attribute. -->
<FileLogger Level="error" Location="./log/cd_transport.log"/>
<!-- Enable this Logger to print log messages to a console. -->
<!-- <ConsoleLogger Level="error" Trace="on"/> -->
</Logger>
</Logging>
<!-- The Senders section defines the installed Sender types. Senders are configured in
      the Management System interface as Publication Target Destinations. -->
<Senders>
<!-- Install custom Senders by providing a 'Type' that matches the root
      element name of a Management System Protocol Schema. The 'Class'
      attribute specifies the Java class that implements the functionality
      for a Sender. Make sure the class is registered on the system
      CLASSPATH environment variable. -->
<Sender Type="Local" Class="com.tridion.transport.LocalCopySender"/>
<Sender Type="HTTP" Class="com.tridion.transport.HTTPSSender"/>
<Sender Type="HTTPS" Class="com.tridion.transport.HTTPSSender"/>
<Sender Type="FTP" Class="com.tridion.transport.FTPSender"/>
<Sender Type="SFTP" Class="com.tridion.transport.SFTPSender"/>
</Senders>
</TransportService>

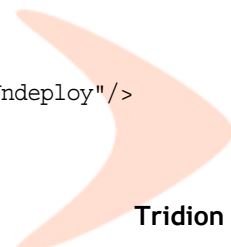
```



## Appendix D Sample Deployer configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- The Tridion Content Distributor Deployer configuration specifies all
      configuration values required to receive and deploy content to a delivery system. -->
<Deployer>
<!-- The WorkFolder element defines the location where temporary files are
      stored. Set 'Cleanup' to 'false' if TransportPackages should be
      archived or used for development purposes. -->
<WorkFolder Location="C:\Tridion\work" Cleanup="true"/>
<!-- The Logging section specifies the behaviour and location of one or more loggers. -->
<Logging>
<!-- Defines the default logger. The Level attribute specifies the amount of information to
log.
      Legal logging levels:
      info Information messages.
      warning Potential error conditions.
      error Errors that prevent the software from functioning correctly.
      fatal Unrecoverable errors. -->
<Logger Level="error">
<!-- A FileLogger logs to a log file specified by the 'Location' attribute. -->
<FileLogger Level="error" Location="./log/cd_deployer.log"/>
<!-- Enable this Logger to print log messages to a console. -->
<!-- <ConsoleLogger Level="error" Trace="on"/> -->
</Logger>
</Logging>
<!-- The Processors section defines what actions the Deployer is able to
      performed. The behavior of the Deployer is defined by the type of
      Processors the type and sequence of modules. The Processors section is
      where custom Deployer behavior can be configured. -->
<Processors>
<!-- A Processor is triggered by the Deployer to process an incoming
      TransportPackage based on the 'Action' command in the
      ProcessorInstructions. The default Processor triggers modules
      sequentially as they are defined in a Processor section. The 'Class'
      attribute defines the Processor class that will be used for processing an action. -->
<Processor Action="Deploy" Class="com.tridion.deployer.Processor">
<!-- A Module is triggered by a Processor to process incoming instructions.
      The 'Type' attribute needs to be unique within a Processor and serves
      as a symbolic identifier. The 'Class' attribute defines the
      implementation used for any type of Module. Replace or add modules to
      implement custom Deployer behavior. -->
<Module Type="SchemaDeploy" Class="com.tridion.deployer.modules.SchemaDeploy"/>
<Module Type="PageDeploy" Class="com.tridion.deployer.modules.PageDeploy"/>
<Module Type="BinaryDeploy" Class="com.tridion.deployer.modules.BinaryDeploy"/>
<Module Type="ComponentDeploy" Class="com.tridion.deployer.modules.ComponentDeploy"/>
<Module Type="TemplateDeploy" Class="com.tridion.deployer.modules.TemplateDeploy"/>
<Module Type="ComponentPresentationDeploy"
Class="com.tridion.deployer.modules.ComponentPresentationDeploy"/>
</Processor>
<Processor Action="Undeploy" Class="com.tridion.deployer.Processor">
<Module Type="PageUndeploy" Class="com.tridion.deployer.modules.PageUndeploy"/>
<Module Type="ComponentPresentationUndeploy"
Class="com.tridion.deployer.modules.ComponentPresentationUndeploy"/>

```





```
</Processor>
</Processors>
<!-- The ReceiverBindings section define the types of Receivers that can be used by the
Deployer. -->
<ReceiverBindings>
  <!-- A Receiver can be identified by the 'Type' attribute. The 'Class'
       attribute defines the implementation used for receiving TransportPackages. -->
  <Receiver Type="HTTPSReceiver" Class="com.tridion.transport.HTTPSReceiver"/>
  <Receiver Type="FileReceiver" Class="com.tridion.transport.FileReceiver"/>
</ReceiverBindings>
<!-- The Receivers section specifies Receiver instances that will be started by the
Deployer. -->
<Receivers>
  <!-- Element names for Receivers that are configured in this section must
       match the 'Type' attribute as defined in the ReceiverBindings section. -->
  <FileReceiver Location="C:\Tridion\incoming" Interval="1"/>
  <!-- Enable this Receiver to handle the HTTP transport method. Change the
       'MaxSize' attribute to limit the maximum size of accepted TransportPackages. -->
  <!--<HTTPSReceiver MaxSize="10000000"/>-->
</Receivers>
</Deployer>
```



## Appendix E Sample Broker configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- The Tridion Content Broker configuration specifies all configuration values to enable
data storage. -->
<Configuration Version="5.1">
  <!-- The Global section specifies configuration details that apply to all functionality. -->
  <Global>
    <!-- The Logging section specifies the behaviour and location of one or more loggers. -->
    <Logging>
      <!-- Defines the default logger. The Level attribute specifies the amount of information
to log.
Legal logging levels:
info Information messages.
warning Potential error conditions.
error Errors that prevent the software from functioning correctly.
fatal Unrecoverable errors. -->
      <Logger Level="error">
        <!-- A FileLogger logs to a log file specified by the 'Location' attribute. -->
        <FileLogger Level="error" Location="./log/cd_broker.log"/>
        <!-- Enable this Logger to print log messages to a console. -->
        <!-- <ConsoleLogger Level="error" Trace="on"/> -->
      </Logger>
    </Logging>
    <Storage>
      <!-- Database elements configure settings for a specific database. Multiple
Database elements can be used. Each Database element has a type,
either sql or tamino. Only one element can be specified for a certain
type. Because of this, you can specify a maximum of two Database elements.
The following attributes can be specified:
Type The type of the database, this can be sql or tamino.
Username The user name used to connect to the database.
Password The password used to connect to the database.
Url The url the driver should use.
Driver The fully qualified classname of the driver to use.
JNDIName The JNDI name of a Datasource to use (optional, other attributes are then
ignored).

For Database elements of type sql, you can specify the pooling that
should be used with a Pool element. This element has the following
attributes:
Type The type should be either jdbc or tridion.
Size Maximum number of connections to open.
MonitorInterval Number of seconds between checks on the pool.
IdleTimeout Number of seconds a connection can be idle before it is closed.
CheckoutTimeout Number of seconds a connection can be checked out before it is
returned to pool. -->
      <!-- For MS SQL Server, the following structure should be used: -->
      <!-- <Database Type="sql" Username="USERNAME" Password="PASSWORD"
Url="jdbc:microsoft:sqlserver://DATABASE_HOST:PORT"
Driver="com.microsoft.jdbc.sqlserver.SQLServerDriver">
<Pool Type="jdbc" Size="10"/>
</Database> -->
      <!-- For DB2, the following structure should be used: -->
      <!-- <Database Type="sql" Username="USERNAME" Password="PASSWORD"
Url="jdbc:db2://DATABASE_HOST:PORT/DATABASE_NAME"
```



```

        Driver="COM.ibm.db2.jdbc.net.DB2Driver">
        <Pool Type="jdbc" Size="10"/>
    </Database> -->
<!-- For Oracle, the following structure should be used: -->
<!-- <Database Type="sql" Username="USERNAME" Password="PASSWORD"
        Url="jdbc:oracle:thin:@DATABASE_HOST:PORT:SID"
        Driver="oracle.jdbc.driver.OracleDriver">
        <Pool Type="jdbc" Size="10"/>
    </Database> -->
<!-- For a JNDI datasource, the following structure should be used: -->
<!-- <Database Type="sql" JNDIName="java:comp/env/jdbc/DATA_SOURCE_NAME"/> -->
<!-- For Tamino, the following structure should be used: -->
<!-- <Database Type="tamino" Username="USERNAME" Password="PASSWORD"
        Url="http://HOST_NAME/HOST_PATH/DATABASE_NAME" /> -->
<Database Type="sql" Username="USERNAME" Password="PASSWORD"
Url="jdbc:oracle:thin:@DATABASE_HOST:PORT:SID" Driver="oracle.jdbc.driver.OracleDriver">
    <Pool Type="jdbc" Size="10"/>
</Database>
</Storage>
<!-- The Bindings element contains mappings from symbolic names to classes in the form of
Binding elements.
    A Binding element has two attributes:
    Name The symbolic name for an item that needs to be stored.
    Class The class implementing the interface that is expected for a certain symbolic name.
    By default, the following bindings are loaded: -->
<!--
    <Binding Name="LinkInfo" Class="com.tridion.broker.linking.FSCSVLinkInfoHome"/>
    <Binding Name="ComponentMeta"
Class="com.tridion.broker.components.meta.XMLFileComponentMetaHome"/>
    <Binding Name="PageMeta" Class="com.tridion.broker.pages.meta.XMLFilePageMetaHome"/>
    <Binding Name="BinaryMeta"
Class="com.tridion.broker.binaries.meta.XMLFileBinaryMetaHome"/>
    <Binding Name="Schema" Class="com.tridion.broker.schemas.FSSchemaHome"/>
    <Binding Name="Template" Class="com.tridion.broker.xslt.FSXSLTHome"/>
    <Binding Name="XSLT" Class="com.tridion.broker.xslt.FSXSLTHome"/>
    <Binding Name="ComponentPresentationMeta"
Class="com.tridion.broker.componentpresentations.meta.XMLFileComponentPresentationMetaHome"/>
>
    <Binding Name="TextComponentPresentation"
Class="com.tridion.broker.componentpresentations.FSTextComponentPresentationHome"/>
    <Binding Name="ASPComponentPresentation"
Class="com.tridion.broker.componentpresentations.FSASPComponentPresentationHome"/>
    <Binding Name="JSPComponentPresentation"
Class="com.tridion.broker.componentpresentations.FSJSPComponentPresentationHome"/>
    <Binding Name="XMLComponentPresentation"
Class="com.tridion.broker.componentpresentations.FSXMLComponentPresentationHome"/>
    <Binding Name="Reference" Class="com.tridion.broker.references.FileReferenceHome"/>
-->
<!-- The default bindings (which use the file system) can be overridden for
different storage types. Note that a storage specific implementation class are
delivered for all bindings.
    Below combinations of Bindings are given for the default supported. -->
<!-- MS SQL Server:
<Binding Name="LinkInfo" Class="com.tridion.broker.linking.SQLLinkInfoHome"/>
<Binding Name="PageMeta" Class="com.tridion.broker.pages.meta.SQLPageMetaHome"/>
<Binding Name="ComponentMeta"
Class="com.tridion.broker.components.meta.MsSqlComponentMetaHome"/>
<Binding Name="BinaryMeta" Class="com.tridion.broker.binaries.meta.SQLBinaryMetaHome"/>

```

```

    <Binding Name="Reference" Class="com.tridion.broker.references.SQLReferenceHome"/>
    <Binding Name="ComponentPresentationMeta"
Class="com.tridion.broker.componentpresentations.meta.SQLComponentPresentationMetaHome"/>
    <Binding Name="TextComponentPresentation"
Class="com.tridion.broker.componentpresentations.MsSqlTextComponentPresentationHome"/>
    <Binding Name="XMLComponentPresentation"
Class="com.tridion.broker.componentpresentations.MsSqlXMLComponentPresentationHome"/>
    -->
<!-- DB2:
    <Binding Name="LinkInfo" Class="com.tridion.broker.linking.SQLLinkInfoHome"/>
    <Binding Name="PageMeta" Class="com.tridion.broker.pages.meta.SQLPageMetaHome"/>
    <Binding Name="ComponentMeta"
Class="com.tridion.broker.components.meta.DB2ComponentMetaHome"/>
    <Binding Name="BinaryMeta" Class="com.tridion.broker.binaries.meta.SQLBinaryMetaHome"/>
    <Binding Name="Reference" Class="com.tridion.broker.references.SQLReferenceHome"/>
    <Binding Name="ComponentPresentationMeta"
Class="com.tridion.broker.componentpresentations.meta.SQLComponentPresentationMetaHome"/>
    <Binding Name="TextComponentPresentation"
Class="com.tridion.broker.componentpresentations.DB2TextComponentPresentationHome"/>
    <Binding Name="XMLComponentPresentation"
Class="com.tridion.broker.componentpresentations.DB2XMLComponentPresentationHome"/>
    -->
<!-- Oracle:
    <Binding Name="LinkInfo" Class="com.tridion.broker.linking.SQLLinkInfoHome"/>
    <Binding Name="PageMeta" Class="com.tridion.broker.pages.meta.SQLPageMetaHome"/>
    <Binding Name="ComponentMeta"
Class="com.tridion.broker.components.meta.OracleComponentMetaHome"/>
    <Binding Name="BinaryMeta" Class="com.tridion.broker.binaries.meta.SQLBinaryMetaHome"/>
    <Binding Name="Reference" Class="com.tridion.broker.references.SQLReferenceHome"/>
    <Binding Name="ComponentPresentationMeta"
Class="com.tridion.broker.componentpresentations.meta.SQLComponentPresentationMetaHome"/>
    <Binding Name="TextComponentPresentation"
Class="com.tridion.broker.componentpresentations.OracleTextComponentPresentationHome"/>
    <Binding Name="XMLComponentPresentation"
Class="com.tridion.broker.componentpresentations.OracleXMLComponentPresentationHome"/>
    -->
<!-- Tamino:
    <Binding Name="PageMeta" Class="com.tridion.broker.pages.meta.TaminoPageMetaHome"/>
    <Binding Name="ComponentMeta"
Class="com.tridion.broker.components.meta.TaminoComponentMetaHome"/>
    <Binding Name="ComponentPresentationMeta"
Class="com.tridion.broker.componentpresentations.meta.TaminoComponentPresentationMetaHome"/>
    <Binding Name="Schema" Class="com.tridion.broker.schemas.TaminoSchemaHome"/>
    <Binding Name="XMLComponentPresentation"
Class="com.tridion.broker.componentpresentations.TaminoXMLComponentPresentationHome"/>
    -->
<!-- When the WAI is installed with Profiling and Tracking are enabled, the
    following bindings should be added: -->
<!-- MS SQL Server:
    <Binding Name="User" Class="com.tridion.user.MsSqlUserHome"/>
    <Binding Name="CustomerCharacteristic"
Class="com.tridion.personalization.SQLCustomerCharacteristicHome"/>
    <Binding Name="TrackingKey" Class="com.tridion.personalization.SQLTrackingKeyHome"/>
    <Binding Name="Timeframe" Class="com.tridion.timeframes.MsSqlTimeframeHome"/>
    <Binding Name="TrackedPage" Class="com.tridion.tracking.pages.SQLTrackedPageHome"/>
    <Binding Name="TrackedComponent"
Class="com.tridion.tracking.components.SQLTrackedComponentHome"/>

```

```

    <Binding Name="TrackedComponentLink"
Class="com.tridion.tracking.componentlinks.SQLTrackedComponentLinkHome"/>
-->
<!-- DB2:
    <Binding Name="User" Class="com.tridion.user.DB2UserHome"/>
    <Binding Name="CustomerCharacteristic"
Class="com.tridion.personalization.SQLCustomerCharacteristicHome"/>
    <Binding Name="TrackingKey" Class="com.tridion.personalization.SQLTrackingKeyHome"/>
    <Binding Name="Timeframe" Class="com.tridion.timeframes.DB2TimeframeHome"/>
    <Binding Name="TrackedPage" Class="com.tridion.tracking.pages.SQLTrackedPageHome"/>
    <Binding Name="TrackedComponent"
Class="com.tridion.tracking.components.SQLTrackedComponentHome"/>
    <Binding Name="TrackedComponentLink"
Class="com.tridion.tracking.componentlinks.SQLTrackedComponentLinkHome"/>
-->
<!-- Oracle:
    <Binding Name="User" Class="com.tridion.user.OracleUserHome"/>
    <Binding Name="CustomerCharacteristic"
Class="com.tridion.personalization.SQLCustomerCharacteristicHome"/>
    <Binding Name="TrackingKey" Class="com.tridion.personalization.SQLTrackingKeyHome"/>
    <Binding Name="Timeframe" Class="com.tridion.timeframes.OracleTimeframeHome"/>
    <Binding Name="TrackedPage" Class="com.tridion.tracking.pages.SQLTrackedPageHome"/>
    <Binding Name="TrackedComponent"
Class="com.tridion.tracking.components.SQLTrackedComponentHome"/>
    <Binding Name="TrackedComponentLink"
Class="com.tridion.tracking.componentlinks.SQLTrackedComponentLinkHome"/>
-->
<Bindings>
    <Binding Name="User" Class="com.tridion.user.MsSqlUserHome"/>
    <Binding Name="CustomerCharacteristic"
Class="com.tridion.personalization.SQLCustomerCharacteristicHome"/>
    <Binding Name="TrackingKey" Class="com.tridion.personalization.SQLTrackingKeyHome"/>
    <Binding Name="Timeframe" Class="com.tridion.timeframes.MsSqlTimeframeHome"/>
    <Binding Name="TrackedPage" Class="com.tridion.tracking.pages.SQLTrackedPageHome"/>
    <Binding Name="TrackedComponent"
Class="com.tridion.tracking.components.SQLTrackedComponentHome"/>
    <Binding Name="TrackedComponentLink"
Class="com.tridion.tracking.componentlinks.SQLTrackedComponentLinkHome"/>
    <Binding Name="ComponentMeta"
Class="com.tridion.broker.components.meta.XMLFileComponentMetaHome"/>
    <Binding Name="PageMeta" Class="com.tridion.broker.pages.meta.XMLFilePageMetaHome"/>
    <Binding Name="XSLT" Class="com.tridion.broker.xslt.FSXSLTHome"/>
    <Binding Name="LinkInfo" Class="com.tridion.broker.linking.FSCSVLinkInfoHome"/>
</Bindings>
</Global>
<!-- The Publications element specifies the default setting for all
Publications. This information can be overridden in Publication elements.

```

The `DefaultRootLocation` specifies a directory for both pages and binaries. By default, this folder is also used to find the meta-data. Note that any directory information defined in this `Publications` or lower `Publication` elements override configuration information defined by the `Linking` configuration. -->

```
<Publications DefaultRootLocation="c:/inetpub/wwwroot">
```

<!-- The `publication` element allows you to specify `Publication` specific settings. This element is optional.

The `Id` attribute indicates the id of the publication. This should be a number.

A publication can optionally contain a Dcp element.

The following optional attributes override storage settings:

**DataRoot** The directory in which meta-data should be stored and retrieved on the filesystem. If this is set it will override any value set in the Publications element's DefaultRootLocation.

**DocumentRoot** The base-directory in which pages and binaries are located. -->

```
<Publication Id="1" DataRoot="c:/inetpub/wwwroot/pub1"
DocumentRoot="c:/inetpub/data/pub1">
  <!-- The Dcp element contains one or more file type specific locations to store
  files of that type. -->
  <Dcp>
    <!-- The Jsp element allows you to specify settings for JSP pages.
    The required Location attribute overrides storage settings:
      Location The directory in which JSP pages should be stored
      and retrieved on the filesystem. This overrides
      any values set in the Publications element's
      DefaultRootLocation and the Publication element's DataRoot. -->
    <Jsp Location="c:/AbsoluteLocation(1)"/>
    <!-- The Asp element allows you to specify settings for ASP pages.
    The required Location attribute overrides storage settings:
      Location The directory in which ASP pages should be stored
      and retrieved on the filesystem. This overrides
      any values set in the Publications element's
      DefaultRootLocation and the Publication element's DataRoot. -->
    <Asp Location="c:/AbsoluteLocation(1)"/>
    <!-- The Xml element allows you to specify settings for XML files.
    The required Location attribute overrides storage settings:
      Location The directory in which XML files should be stored
      and retrieved on the filesystem. This overrides
      any values set in the Publications element's
      DefaultRootLocation and the Publication element's DataRoot. -->
    <Xml Location="c:/AbsoluteLocation(1)"/>
    <!-- The Txt element allows you to specify settings for text files.
    The required Location attribute overrides storage settings:
      Location The directory in which test files should be stored
      and retrieved on the filesystem. This overrides
      any values set in the Publications element's
      DefaultRootLocation and the Publication element's DataRoot. -->
    <Txt Location="c:/AbsoluteLocation(1)"/>
  </Dcp>
</Publication>
</Publications>
</Configuration>
```



## Appendix F Broker bindings for MS SQL, Oracle, Tamino and DB2 databases

### MS SQL

```
<Binding Name="LinkInfo"
Class="com.tridion.broker.linking.SQLLinkInfoHome"/>
<Binding Name="PageMeta"
Class="com.tridion.broker.pages.meta.SQLPageMetaHome"/>
<Binding Name="ComponentMeta"
Class="com.tridion.broker.components.meta.MsSqlComponent
MetaHome"/>
<Binding Name="BinaryMeta"
Class="com.tridion.broker.binaries.meta.SQLBinaryMetaHome"/>
<Binding Name="Reference"
Class="com.tridion.broker.references.SQLReferenceHome"/>
<Binding Name="ComponentPresentationMeta"
Class="com.tridion.broker.componentpresentations.meta.SQL
ComponentPresentationMetaHome" />
```

### Oracle

```
<Binding Name="LinkInfo"
Class="com.tridion.broker.linking.SQLLinkInfoHome"/>
<Binding Name="PageMeta"
Class="com.tridion.broker.pages.meta.SQLPageMetaHome"/>
<Binding Name="ComponentMeta"
Class="com.tridion.broker.components.meta.OracleComponent
MetaHome"/>
<Binding Name="BinaryMeta"
Class="com.tridion.broker.binaries.meta.SQLBinaryMetaHome"/>
<Binding Name="Reference"
Class="com.tridion.broker.references.SQLReferenceHome"/>
<Binding Name="ComponentPresentationMeta"
Class="com.tridion.broker.componentpresentations.meta.SQL
ComponentPresentationMetaHome" />
```

### Tamino

```
<Binding Name="PageMeta"
Class="com.tridion.broker.pages.meta.TaminoPageMetaHome"/>
<Binding Name="ComponentMeta"
Class="com.tridion.broker.components.meta.TaminoComponentMetaHome"/>
<Binding Name="ComponentPresentationMeta"
Class="com.tridion.broker.componentpresentations.meta.TaminoComponent
PresentationMetaHome"/>
<Binding Name="XMLComponentPresentation"
Class="com.tridion.broker.componentpresentations.TaminoXMLComponent
PresentationHome"/>
```

Refer to Chapter 11 "*Tamino*" on page 89 for more information about configuring and using the Tamino XML database with the Content Distributor.

### DB2

```
<Binding Name="LinkInfo"  
Class="com.tridion.broker.linking.SQLLinkInfoHome"/>  
<Binding Name="PageMeta"  
Class="com.tridion.broker.pages.meta.SQLPageMetaHome"/>  
<Binding Name="ComponentMeta"  
Class="com.tridion.broker.components.meta.DB2ComponentMetaHome"/>  
<Binding Name="BinaryMeta"  
Class="com.tridion.broker.binaries.meta.SQLBinaryMetaHome"/>  
<Binding Name="Reference"  
Class="com.tridion.broker.references.SQLReferenceHome"/>  
<Binding Name="ComponentPresentationMeta"  
Class="com.tridion.broker.componentpresentations.meta.SQLComponent  
PresentationMetaHome" />
```





## Appendix G WAI-Specific bindings

If the Web and Application Server Integration (WAI) is installed, and Profiling and Tracking is enabled, the following bindings must be added:

### MS SQL

```
<Binding Name="User" Class="com.tridion.user.MsSqlUserHome"/>
<Binding Name="CustomerCharacteristic"
Class="com.tridion.personalization.SQLCustomerCharacteristicHome"/>
<Binding Name="TrackingKey"
Class="com.tridion.personalization.SQLTrackingKeyHome"/>
<Binding Name="Timeframe"
Class="com.tridion.timeframes.MsSqlTimeframeHome"/>
<Binding Name="TrackedPage"
Class="com.tridion.tracking.pages.SQLTrackedPageHome"/>
<Binding Name="TrackedComponent"
Class="com.tridion.tracking.components.SQLTrackedComponentHome"/>
<Binding Name="TrackedComponentLink"
Class="com.tridion.tracking.componentlinks.SQLTrackedComponentLinkHome"/>
```

### Oracle

```
<Binding Name="User" Class="com.tridion.user.OracleUserHome"/>
<Binding Name="CustomerCharacteristic"
Class="com.tridion.personalization.SQLCustomerCharacteristicHome"/>
<Binding Name="TrackingKey"
Class="com.tridion.personalization.SQLTrackingKeyHome"/>
<Binding Name="Timeframe"
Class="com.tridion.timeframes.OracleTimeframeHome"/>
<Binding Name="TrackedPage"
Class="com.tridion.tracking.pages.SQLTrackedPageHome"/>
<Binding Name="TrackedComponent"
Class="com.tridion.tracking.components.SQLTrackedComponentHome"/>
<Binding Name="TrackedComponentLink"
Class="com.tridion.tracking.componentlinks.SQLTrackedComponentLinkHome"/>
```

### DB2

```
<Binding Name="User" Class="com.tridion.user.DB2UserHome"/>
<Binding Name="CustomerCharacteristic"
Class="com.tridion.personalization.SQLCustomerCharacteristicHome"/>
<Binding Name="TrackingKey"
Class="com.tridion.personalization.SQLTrackingKeyHome"/>
<Binding Name="Timeframe"
Class="com.tridion.timeframes.DB2TimeframeHome"/>
<Binding Name="TrackedPage"
Class="com.tridion.tracking.pages.SQLTrackedPageHome"/>
<Binding Name="TrackedComponent"
Class="com.tridion.tracking.components.SQLTrackedComponentHome"/>
<Binding Name="TrackedComponentLink"
Class="com.tridion.tracking.componentlinks.SQLTrackedComponentLinkHome"/>
```

## Appendix H System classpath

On Windows systems, the Java Virtual Machine (JVM) uses the system classpath to locate libraries and other resources (such as configuration files).

In previous versions of the Tridion Content Distributor, you were required to specify a system classpath to be used by the Transport Service, Content Deployer, Content Broker, Cache Service, WAI, and Linking. This involved specifying the `CLASSPATH` environment variable, and rebooting your system for changes to take effect.

The Content Distributor is now installed by default to `C:\Program Files\Tridion`. As a result, the length of the required environment variable for the classpath now exceeds the 1024 byte limit that Windows imposes on any environment variable.

Content Distributor 5.1 SP4 now dynamically builds this `CLASSPATH`. The `CLASSPATH` is generated based on the following:

- When you start your service, the Tridion Home is identified as follows:
  - If there is a registry entry for `TRIDION_HOME` for this specific service or for all Tridion Content Delivery Services, the value of that entry is Tridion Home.
  - If no such registry entry exists, Tridion Home is set to the value of the environment variable `TRIDION_HOME`, which by default is set to `C:\Program Files\Tridion`.
- If both the registry entry and the environment variable are not set, the parent directory of the folder in which `cd_*.exe` is located is identified as Tridion Home. Where `*` is `transport`, `deployer`, `broker`, `cacheservice`, `wai`, or `linking`.
- Once Tridion Home is determined, the system then adds `%TRIDION_HOME%\Config` to the `CLASSPATH`. The config directory contains the XML configuration files for the various products (e.g. `cd_transport_conf.xml` and `cd_license.xml`).
- The system iterates through the `*.jar` and `*.zip` files in the `%TRIDION_HOME%\Lib` directory and adds them to the `CLASSPATH`.

The system then adds the generated classpath to the system classpath.

## Appendix I Third-party libraries

The Content Distributor uses the following third-party libraries:

### Senders & Receivers

Senders and Receivers use the following third-party libraries.

- The FTP Sender-Receiver pair uses the following:
  - `ftp.jar` — implements FTP functionality.
- The SFTP Sender-Receiver pair uses the following:
  - `sftp.jar` — implements SFTP functionality.
- The HTTP(s) Sender-Receiver pair uses the following:
  - `cos.jar` — O'Reilly servlet utility library (`com.oreilly.servlet`).
  - `jsse.jar` — Sun library which implements the Java Secure Sockets Extension (JSSE).  
This library is dependent on:
    - `jnet.jar` — SUN library containing socket factories
    - `jcerc.jar` — SUN library for handling certificates

### XML

The following third-party libraries are used for XML formatting and interpretation:

- `xalan.jar` — XSLT processor from the Apache Software Foundation. See [www.apache.org](http://www.apache.org) for more information.
- `xercesImpl.jar` — XML Parser from the Apache Software Foundation. See [www.apache.org](http://www.apache.org) for more information.
- `xmlParserAPIs.jar` — Jar file containing all the standard APIs implemented by the Xerces parser. See [www.apache.org](http://www.apache.org) for more information.
- `jdom.jar` — open source library for Java-optimized XML data manipulations from [jdom.org](http://jdom.org).

### Database

The following third-party libraries are used for database connectivity:

- `TaminoAPI4J.jar` — Java API to Software AG Tamino XML database.
- `classes12.zip` — Oracle driver classes for JDBC.
- `db2java.zip` — DB2 driver classes for JDBC.
- `jdbc2_0-stdext.jar` — javax.sql standard extension for SQL.
- `jdbcpool-0.99.jar` — JDBC connection pool by James Cooper of [www.bitmechanic.com](http://www.bitmechanic.com)
- `msbase.jar` — MS SQL driver classes for JDBC.
- `mssqlserver.jar` — MS SQL driver classes for JDBC.
- `msutil.jar` — MS SQL driver classes for JDBC.

### Miscellaneous

The following third-party libraries are used:

- `log4j.jar` — Logging library from Apache Software Foundation. See [www.apache.org](http://www.apache.org) for more information. This is only used for Tamino.
- `ezlicrun.jar` — This file must be present to use any licensed products.
- `jndi.jar` — Used to map objects to contexts.
- `jakarta-oro.jar` — Processes regular expressions.

### Differences between JRE 1.3 and JRE 1.4

JRE 1.3 does not contain some classes that are included in JRE 1.4.

If you use JRE 1.3, you need to ensure that the following jar files are installed:

- `jsse.jar` — Sun library which implements the Java Secure Sockets Extension (JSSE). This library is dependent on:
  - `jnet.jar` — SUN library containing socket factories
  - `jcert.jar` — SUN library for handling certificates

**Note** These files are only required if you are using the HTTPS sender.

## Appendix J    **Configuring the Java Virtual Machine for COM services**

---

**Important:**

---

The functionality described in this Appendix is available only in version 5.1 SP4 and higher of Tridion R5.

---

The Content Delivery COM services use a Java Virtual Machine (JVM). By default, these services start the JVM using system defaults. However, you might want to change these system defaults for one specific Content Delivery COM service, or perhaps for all services at the same time.

This appendix explains how you can modify the JVM call from any or all COM services by editing the Windows Registry. This appendix assumes that you are familiar with `regedit` or that you know how to manipulate the registry in some other way.

### **Content Delivery COM services that access a Java Virtual Machine**

The following COM services access a Java Virtual Machine:

- Tridion Cache Service
- Tridion Content Broker
- Tridion Content Distributor Deployer
- Tridion Content Distributor Transport Service
- Tridion Dynamic Linking
- Tridion Web and Application Server Integration (WAI)

### **To configure the Java Virtual Machine for one or all COM services:**

- 1** Start a registry editor, `regedit` for example, on the Content Delivery server.
- 2** Navigate to the existing Tridion registry subkey:  
`HKEY_LOCAL_MACHINE\SOFTWARE\Tridion\`
- 3** In the `HKEY_LOCAL_MACHINE\SOFTWARE\Tridion` subkey, create a subkey called `Content Delivery`.



### To configure the Java Virtual Machine for one or all COM services:

- 4 In the `HKEY_LOCAL_MACHINE\SOFTWARE\Tridion\Content Delivery` subkey, you can create subkeys for the various Content Delivery COM services with the following names:
  - `Tridion Cache Service`
  - `Tridion Content Broker`
  - `Tridion Content Distributor Deployer`
  - `Tridion Content Distributor Transport Service`
  - `Tridion Dynamic Linking`
  - `Tridion Web and Application Server Integration`

You can also add a subkey called `General`.

**Note** All of these subkeys are optional. You can add one, three or all of them.

- 5 You can now add one, more or all of the following entries to each of these subkeys:
  - `TRIDION_HOME`
  - `JREVersion`
  - `jvmarg1`
  - `jvmarg2`
  - `jvmarg3, etc`

These entries are explained below.

- 6 Close the registry editor.
- 7 Stop and start all Content Delivery COM services affected by these settings.

When you now start a Content Delivery COM service, Tridion Content Delivery will do the following:

- It looks for registry subkeys called `HKEY_LOCAL_MACHINE\SOFTWARE\Tridion\Content Delivery\General` and `HKEY_LOCAL_MACHINE\SOFTWARE\Tridion\Content Delivery\<service>` where `<service>` is the name of the service.
- It applies the settings it finds in both subkeys. If it finds the same entry in both subkeys, it always applies the entry found in the service-specific subkey.

**Note** The `jvmargX` keys are seen as one group. This means that, for example, one service-specific registry entry `jvmarg1` overrides all the `jvmarg1`, `jvmarg2`, and `jvmarg3` entries found in the `General` subkey.

What follows is an explanation of the various entries and their meanings.

### JVM configuration entries

For each or for all of the services listed above, you can modify the following settings:

- **The Tridion home directory (registry entry: `TRIDION_HOME`)**

Each COM service needs to know where to find certain Tridion files to include in its classpath. See Appendix H, "System classpath" for details.

If you specify a value for `TRIDION_HOME` for one or all services, the affected COM services will use that value as their starting point for finding files.

This is useful if you have Tridion Content Manager and Tridion Content Delivery installed on the same machine, and want to place the Content Delivery-related files in a different directory than the Content Manager.

- **The JRE version (registry entry: `JREVersion`)**

**Note** Setting this entry is only relevant if you have multiple versions of the Java Runtime Environment (JRE) installed.

By default, Tridion Content Delivery sets the JRE version to whatever it finds as the value of the following registry subkey:

```
HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment\  
CurrentVersion
```

If you specify a registry value for the JRE for one or all services, then that version will be used instead of the default one. The version must, of course, be installed on the Content Delivery server. To see which versions of the JRE are installed, look at the subkeys found in the following registry subkey:

```
HKEY_LOCAL_MACHINE\SOFTWARE\JavaSoft\Java Runtime Environment\  
CurrentVersion
```

Set the `JREVersion` entry to the name of one of the subkeys (that is, a valid version number such as "1.3.1\_13" or "1.4"). If you fill in a version that is not installed, you cannot start the affected COM services after you stop them.

- **Java Virtual Machine startup arguments (registry entries: `jvmarg1`, `jvmarg2`, etc)**

By default, Tridion Content Delivery starts up the Java Virtual Machine (JVM) with its own default parameters.

To add one parameter to one or to all COM services, create a registry entry `jvmarg1` in the appropriate subkey. To add two parameters, create two registry entries, `jvmarg1` and `jvmarg2`, and so on. Give each registry entry the value of a JVM parameter, including an initial dash (-). For example, to set the maximum amount of memory to 256 MB, add a `jvmarg1` registry entry and give it the value `-Xmx256M`.

Please note the following:

- ▶ The `jvmargX` registry entry are seen as one. That is, if your `General1` subkey contains a `jvmarg1` entry and a `jvmarg2` entry, while your `Tridion Content Broker` subkey contains only a `jvmarg1` entry, the COM service will be started with only one custom parameter (the service-specific `jvmarg1`).
- ▶ If you supply a `jvmargX` registry entry that sets the classpath (that is, with a value that starts with `-Djava.class.path` or with `-cp`, the service will not work, because your classpath will override the default one set by the COM service itself.







## Index

### B

- binary
  - mandatory metadata 77
- Binary links 104
- bindings
  - Broker 70
- Broker 61
  - bindings 70
  - cachebindings 66
  - configuring 62
  - configuring metadata storage 80
  - features 65
  - Linking 113
  - logging 63
  - object cache 64
  - object caching 64
  - policy 65
  - publications 71
  - remote synchronization 67
  - storage 68

### C

- cache channel service
  - object caching 67
- cachebindings
  - object caching 66
- Component
  - mandatory metadata 74
- Component links 105
  - Component Template priorities 107
  - created in templates 106
  - directory hierarchy 108
  - objects in a Page Template 107
  - resolving 107
  - to a different Component 106
  - to the same Component 105
- Component Presentation
  - mandatory metadata 76
- Component Presentations
  - accessing through filters 84
- Component Template priorities 107
- Component Templates
  - Tamino 91

- Components
    - Tamino 91
  - configuring
    - Broker 62
    - Deployer 48
    - Linking 111
    - Tamino 89
    - Transport Service 41
  - Content Delivery
    - product overview 1
  - Content Distributor 9
    - third-party libraries 175
  - creating
    - Protocol Schema 15
    - Protocol Schemas 16
    - Publication Target 19
    - Target Type 17
    - Target Types 18
  - custom metadata 77
    - multiple values 79
    - querying 87
  - custom modules 53
    - Deployer 53
  - custom processors 51
- ### D
- default modules
    - modifying 55
  - Deployer 47
    - configuring 48
    - custom modules 53
    - custom processors 51
    - logging 50
    - modifying default modules 55
    - modules 52
    - processors 51
    - receiver bindings 58
    - receivers 58
    - transformer 57
    - workfolder 50
  - Dynamic Component Presentations 140
    - ASP 149
    - assembling 144
    - JSP 145
    - publishing 143



- Dynamic Component Templates
  - output formats 142
- Dynamic content assembly 139
  - Component Templates 141
  - configuring the Content Broker 144
  - XML 152
- E**
- editing 18
  - Target Types 18
- examples
  - metadata 82
- F**
- features
  - Broker 65
  - object caching 65
- filters 84
  - accessing Component Presentations 84
  - metadata 84
  - presentation filters 86
  - result modifiers 86
  - search 85
- FTP 13
- H**
- HTTP(S) 15
  - receivers 33
  - senders 33
  - with Java Web and Application server 37
- HTTPS
  - with IIS 36
- I**
- installing
  - license 5
  - Transport Service license file 4
- L**
- libraries
  - third-party 175
- license
  - installing 5

- licenses 3
- link metadata 74
- Linking 101
  - Binary links 104
  - Broker 113
  - Component links 105
  - configuring 111
  - logging 112
  - Page links 103
  - publication 113
  - validation and resolution 102
- Local File System 13
- Logging
  - Transport Service 43
- logging
  - Broker 63
  - Deployer 50
  - Linking 112
- M**
- metadata 73
  - binary 77
  - Component 74
  - Component Presentation 76
  - configuring storage 80
  - custom 77, 78
  - database storage 80
  - examples 82
  - file system storage 79
  - filters 84
  - mandatory 74
  - mandatory link 74
  - Page 75
  - storage 79
  - storage in SQL databases 81
- modules
  - Deployer 52
- O**
- object cache
  - Broker 64
- object caching
  - Broker 64
  - cache channel service 67
  - cachebindings 66
  - features 65
  - policy 65
  - remote synchronization 67



## P

Page  
mandatory metadata 75

Page links 103

policy  
Broker 65

poller  
Transport Service 44

presentation filters 86

processors  
custom 51  
Deployer 51

Protocol Schema  
creating 15

Protocol Schemas  
creating 16

protocols 13  
FTP 13  
HTTP(S) 15  
Local File System 13

publication  
Linking 113

Publication Target  
creating 19

Publication Targets 11, 17  
Target Type 17

publications  
Broker 71

publishing 9

## Q

queries  
against SQL databases 87

querying  
custom metadata 87

## R

receiver bindings  
Deployer 58

receivers 29, 30  
custom 39  
Deployer 58  
FTP 31  
HTTP(S) 33  
Local Copy 31  
SFTP 31

remote synchronization  
object caching 67

## S

schemas  
Tamino 90

search filters 85

senders 29, 30  
custom 39  
FTP 31  
HTTP(S) 33  
Local Copy 31  
SFTP 31  
Transport Service 43

SFTP 13

SQL  
queries 87

SQL databases  
custom metadata 81

storage  
Broker 68  
metadata 79, 80

## T

Tamino 89  
Component Templates 91  
Components 91  
configuring 89  
Dynamic Content 92  
filter 97  
querying 93  
reference counting 93  
Schema mapping 92  
schemas 90  
unpublishing 92  
updating Components 92  
using 90  
XSLT 93

Target Type  
creating 17



Target Types 18  
  creating 18

transformer  
  Deployer 57

Transport Service 41  
  configuring 41  
  installing license file 4  
  Logging 43  
  poller 44  
  senders 43  
  Workfolder 43

## U

updating  
  Components in Tamino 92

using  
  Tamino 90

## W

Workfolder  
  Transport Service 43

workfolder  
  Deployer 50

