MarkLogic Server

Content Processing Framework Guide

MarkLogic 8
February, 2015

Last Revised: 8.0-6, August, 2016

Copyright © 2016 MarkLogic Corporation. All rights reserved.

Table of Contents

Content Processing Framework Guide

1.0 Overview of the Content Processing Framework 5
- 1.1 Making Content More Useful 5
  - 1.1.1 Getting Your Content Into XML Format 5
  - 1.1.2 Striving For Clean, Well-Structured XML 6
  - 1.1.3 Enriching Content With Semantic Tagging, Metadata, etc. 6
- 1.2 Access Internal and External Web Services 6
- 1.3 Components of the Content Processing Framework 7
  - 1.3.1 Domains 7
  - 1.3.2 Pipelines 7
  - 1.3.3 XQuery Functions and Modules 7
  - 1.3.4 Pre-Commit and Post-Commit Triggers 9
  - 1.3.5 Creating Custom Applications With the Content Processing Framework . 9
- 1.4 Default Conversion Option 9
  - 1.4.1 Microsoft Office and Adobe PDF Conversion 9
  - 1.4.2 HTML Conversion and Enrichment 10

2.0 Getting Started with a Simple CPF Application 11
- 2.1 Overview of Example CPF Application 12
- 2.2 Create the Action Modules 13
  - 2.2.1 add-copyright.xqy 13
  - 2.2.2 add-last-updated.xqy 14
- 2.3 Insert the Action Modules into the Modules Database 14
- 2.4 Create the Pipeline 15
- 2.5 Configure a Database for Content Processing 17
- 2.6 Insert and Update a Document in the Database 20
- 2.7 View the Properties Document 21
- 2.8 Extend the CPF Application 22

3.0 Understanding and Using Domains 26
- 3.1 Overview of Domains 26
- 3.2 Domain Scope and Code Evaluation Context 26
  - 3.2.1 Domain Scope 27
  - 3.2.2 Evaluation Context 27
  - 3.2.3 Domain Scope Can Encapsulate Processing Logic 28
- 3.3 Rules for Domains 28
  - 3.3.1 Do Not Overlap Domains 28
  - 3.3.2 Collection Domain Scope Notes 29
- 3.4 Creating and Modifying Domains 29

3.5 Attaching and Detaching Pipelines to Domains 30

4.0 Understanding and Using Pipelines 31
4.1 Pipeline Architecture 31
4.1.1 Automatic Status (Event) Handling 31
4.1.2 Transitioning Between States 32
4.1.3 Pipelines Can Flow Through Other Pipelines 33
4.2 Viewing Pipelines in the Admin Interface 34
4.3 Loading Pipelines With the Admin Interface 35
4.4 XML Format of a Pipeline 36
4.4.1 Sample Pipeline XML Document 37
4.4.2 Success-Action and Failure-Action 38
4.4.3 Status Transitions 38
4.4.3.1 Status 39
4.4.3.2 On Success and On Failure 39
4.4.3.3 Default Action 39
4.4.3.4 Priority 39
4.4.3.5 Execute 40
4.4.4 State Transitions 40
4.4.4.1 State 40
4.4.4.2 On Success and On Failure 41
4.4.4.3 Default Action 41
4.4.4.4 Priority 41
4.4.4.5 Execute 42
4.5 XQuery Functions to Manage Pipelines 42

5.0 Developing Modules to Process Content 43
5.1 Overview of Modules 43
5.2 Loading Modules Into the Database 43
5.3 External Variables Available to Modules 44
5.4 Design Patterns and Rules 45
5.4.1 Condition Modules Must Return a Boolean 45
5.4.2 Condition Modules Should Not Update Documents 45
5.4.3 Action Modules Use try/catch With cpf:success and cpf:failure 45
5.4.4 Action Modules Operate On a Single Document 46
5.4.5 Action Modules Must Be a Single Transaction 47
5.4.6 Using XSLT Stylesheets Instead of Action Modules 47

6.0 Using the Framework to Create Custom Applications 48
6.1 Install Content Processing Framework in Database 48
6.2 Decide on the Stages and Logic for Your Processing 49
6.3 Create and Load Your Pipelines 50
6.4 Create and Load Your Modules 50
6.5 Design Patterns For Content Applications 50
6.6 Microsoft Office 2007 Documents 51

6.7 Other CPF Applications Included With MarkLogic Server 51

7.0 Security Considerations With Content Processing 52
7.1 Security Requirements for Users Who Create or Modify Documents 52
7.2 Security Requirements When Modules Perform Privileged Operations 52
7.3 Security Roles for Managing Content Processing 53

8.0 Debugging and Recovering from Error Conditions 54
8.1 Database Online Events 54
8.2 Disabling Content Processing Triggers 54
8.3 Content Processing Framework Trace Events 56
8.3.1 List of Trace Events 56
8.3.2 Using the Server Trace Events 57
8.3.3 Sample Scenario for Trace Events 58
8.3.4 Creating Your Own Trace Events 58
8.4 Examining the Host and Task Server Status Pages For Tasks in the Queue 59
8.5 Find Errors in the ErrorLog.txt Log File 60
8.6 Examining Properties Documents 60
8.7 Find Documents in the Error State 61

9.0 The Default Conversion Option 62
9.1 Installing the Conversion Pipelines and Framework 62
9.2 Simple Drag-and-Drop Conversion 64
9.3 What the Conversion Pipeline Generates 65
9.4 Understanding and Using the Default Conversion Option 66
9.4.1 Components of the Default Conversion Option 66
9.4.2 Steps in the Conversion Process 67
9.4.3 Default Conversion Option States 67
9.4.4 Errors, Troubleshooting, Debugging, and Recovery 67
9.4.4.1 Microsoft Word 95 and Other Microsoft Office Errors 68
9.4.4.2 Set the Needed Permissions on the Root Directory 68
9.4.4.3 Default or Inherited Collections and Permissions 69
9.4.4.4 Enable Debugging Capabilities 69
9.4.4.5 Create Your Own Error Handling Pipeline 70
9.5 Modifying the Default Conversion Option 70
9.5.1 Copy Defaults and Modify 70
9.5.2 PDF Alternate Pipelines 71
9.5.3 Modifying the Options for Default Conversion 72

10.0 Technical Support 74

11.0 Copyright 75
11.0 COPYRIGHT 75

1.0 Overview of the Content Processing Framework

Content processing applications often require multi-step processing. Each step in the process performs a particular task or set of tasks. The Content Processing Framework in MarkLogic Server supports these types of multi-step conversion processes. This chapter provides an overview of the framework. This chapter includes the following topics:

- Making Content More Useful
- Access Internal and External Web Services
- Components of the Content Processing Framework
- Default Conversion Option

For instructions on installing the Default Conversion Option, which converts HTML, Adobe PDF, and Microsoft Office documents to XML, see “Installing the Conversion Pipelines and Framework” on page 62.

1.1 Making Content More Useful

Content is information stored in a variety of forms. It could be stored on paper, in various proprietary electronic formats, or in XML format. For many businesses, content is their most valuable asset. In many cases, however, it is very difficult to extract information out of that content because of a variety of reasons, including:

- The content is difficult to access.
- It is difficult to compare with other content.
- You might not know the content exists.

In short, it is difficult to get the most use out of the content that already exists. Content processing is a way to programmatically add value to your content by helping to alleviate some of these difficulties.

The overall goal of content processing is to make your content more useful. Sometimes that means adding value to it by combining it with other content, and sometimes it means making it more accessible to more people. By moving the content through a series of content processing steps, you can add value to it at each step of the process.

1.1.1 Getting Your Content Into XML Format

In order to efficiently add value to your content, the first step is to get the content into a marked up XML format. In many cases the content is already in XML format, but sometimes it needs to undergo a conversion process to become XML. For example, if your content is in HTML, Microsoft Word, or Adobe PDF format, you can convert it to XML using the Default Conversion Option, as described in “The Default Conversion Option” on page 62.

1.1.2 Striving For Clean, Well-Structured XML

One of the important requirements for building rich content applications is to have content that is in a well-structured XML format and is free from errors. Having well-structured XML allows you to easily build applications that can combine content from many documents to produce new and unique documents that are based on some dynamic criteria. In order to get well-structured XML, however, you typically have to perform some processing on the source content to transform it to XML, and then transform the XML to the structure you need.

The requirements for content conversion and transformation are often very specific to an organization, and can require a custom solution. MarkLogic Server allows you to build custom pipelines that exactly match your specific requirements. The Content Processing Framework provides the tools needed to build your own custom pipeline.

1.1.3 Enriching Content With Semantic Tagging, Metadata, etc.

One way to make your content more useful is to add information to it. For example, you can add semantic tagging, which identifies and tags meaningful concepts within the content. The markup from the semantic tagging creates a more uniform target for search and retrieval applications.

There are many web services designed to add semantic tagging to XML documents. For example, a web service might perform entity extraction on an XML document, which identifies people, places, and other entities within keywords in documents. You can write custom applications to add this type of functionality or you can use one of many third-party products and services, often available as web services, to supplement your existing content.

You can also add metadata to the content. Metadata is any kind of information about the document. It can be simple, such as the date the document was created, or it can be more involved, such as information on how the document came into being. For example, MarkLogic Server includes options to maintain metadata (such as the last modified date) on documents, and the Default Conversion Option maintains metadata for all of the content it processes.

The Content Processing Framework makes it easy to add these services as steps in your content processing application.

1.2 Access Internal and External Web Services

You can access web services, both within an intranet and anywhere across the internet, with the XQuery-level HTTP functions built into MarkLogic Server. The HTTP functions allow you to perform HTTP operations such as GET, PUT, POST, and DELETE. You can access these functions directly through XQuery, thus allowing you to post or get content from any HTTP server, including the ability to communicate with web services. The web services that you communicate with can perform external processing on your content, such as entity extraction, language translation, or some other custom processing. Combined with the conversion and HTML Tidy functions, the HTTP functions make it very easy to process any content you can get to on the web within MarkLogic Server.

The XQuery-level HTTP functions can also be used directly with `xdmp:document-load`, `xdmp:document-get`, and all of the conversion functions. You can then, for example, directly process content extracted via HTTP from the web and process it with HTML Tidy (`xdmp:tidy`), load it into the database, or do anything you need to do with any content available via HTTP.

> **Note:** MarkLogic Server requires all data to be in UTF-8 format. Therefore, any web services or HTTP services you access must return data UTF-8 format, or an error occurs.

1.3 Components of the Content Processing Framework

This section describes the components of the Content Processing Framework and includes the following sections:

- Domains
- Pipelines
- XQuery Functions and Modules
- Pre-Commit and Post-Commit Triggers
- Creating Custom Applications With the Content Processing Framework

1.3.1 Domains

A *domain* defines a scope of documents to process. With domains, you can organize your content so that some documents are processed in one way and others are processed in another way. Domains provide flexibility in your content processing applications, making it easy to demarcate sets of documents to which you want to apply the same processing. For more details about domains, see “Understanding and Using Domains” on page 26.

1.3.2 Pipelines

A *pipeline* is an XML document that describes a set of content processing steps. A pipeline defines the steps that occur during the processing of documents and defines actions that occur at each step. After each step, the document being processed is committed to the database, and the Content Processing Framework then catches this document change event with a trigger, which in turn can execute some more processing. This process continues for as many steps as you have defined in the pipeline. There are XQuery functions to help you define and install pipelines. For more details on pipelines, see “Understanding and Using Pipelines” on page 31.

1.3.3 XQuery Functions and Modules

As part of the Content Processing Framework, MarkLogic Server includes many XQuery functions and supporting XQuery modules. The following functions are built into MarkLogic Server; they support document conversion and other applications you can build with the Content Processing Framework:

- `xdmp:pdf-convert`
- `xdmp:word-convert`
- `xdmp:excel-convert`
- `xdmp:powerpoint-convert`
- `xdmp:tidy`
- `xdmp:document-load`
- `xdmp:document-get`
- `xdmp:http-get`
- `xdmp:http-post`
- `xdmp:http-put`
- `xdmp:http-delete`

Additionally, there are XQuery modules to support the Content Processing Framework. Each module has XQuery functions designed to use in content processing applications.

The following modules support the Content Processing Framework:

- Triggers
- Content Processing Framework
- Links
- Domains
- Pipelines

These XQuery modules include the XQuery source code, so you can analyze them and use their functions in your own applications. The XQuery modules are installed into the following directory:

```
<install_dir>/Modules/MarkLogic
```

For details on these functions, see the *MarkLogic XQuery and XSLT Function Reference*.

Note: The convert functions and the conversion XQuery modules require the Default Conversion Option to run.

1.3.4 Pre-Commit and Post-Commit Triggers

MarkLogic Server and the Content Processing Framework use triggers to automate processes that are described by a pipeline. Triggers allow you to capture document events (create, update, delete, or property change) and system events (database online), and then perform some tasks after the event occurs. The tasks you perform are defined by an XQuery module, and can therefore be anything you can do in XQuery. The triggers capture the document change events for documents under a domain. Some of the triggers used by the Content Processing Framework are pre-commit triggers (that is, they execute *before* the transaction completes) and some and are post-commit triggers (that is, they execute *after* the transaction commits). When you construct pipelines for processing your content, triggers automate the transitions between stages of the pipeline. For more details about triggers, see the chapter on triggers in the *Application Developer's Guide*.

1.3.5 Creating Custom Applications With the Content Processing Framework

The Content Processing Framework is designed for creating your own content processing applications, with your own content processing code, and with pipelines that follow your own logical and business processes.

For details on creating custom applications with the Content Processing Framework, see "Using the Framework to Create Custom Applications" on page 48.

1.4 Default Conversion Option

The Default Conversion Option includes features to perform the following types of document conversion:

- Microsoft Office and Adobe PDF Conversion
- HTML Conversion and Enrichment

For details on the Default Conversion Option, see "The Default Conversion Option" on page 62.

1.4.1 Microsoft Office and Adobe PDF Conversion

Included with the Default Conversion Option is the ability to convert Microsoft Word, Excel, and Powerpoint documents, as well as Adobe PDF documents, to XML. You can access this functionality directly through XQuery. The Default Conversion Option, which is written in XQuery, uses the conversion functions to do the initial conversion to XML from Adobe PDF and Microsoft Office documents.

> **Note:** The Default Conversion Option requires Microsoft Office documents to be Office 97 or newer format; it does not convert Office 95 documents or older documents.

1.4.2 HTML Conversion and Enrichment

HTML documents contain HTML markup, but are not required to be well-formed and are not XML. XHTML is an XML version of HTML that requires well-formed tags, is case-sensitive for the element and attribute names, and is more strict in its structured requirements than HTML.

MarkLogic Server includes an implementation of HTML Tidy, which is an open source project (http://tidy.sourceforge.net/) that allows you to take any HTML document and convert it to well-formed XHTML. This provides XQuery-level access to Tidy, with the processing done natively within MarkLogic Server.

In addition to simply converting the HTML documents to XHTML, the XHTML is significantly enriched, improving its structure. The enriched structure makes it easier to create robust applications with the content.

2.0 Getting Started with a Simple CPF Application

This chapter describes the basic procedures for creating a simple CPF application. The procedures include creating two action modules and a pipeline, and how to configure a database to make use of the pipeline.

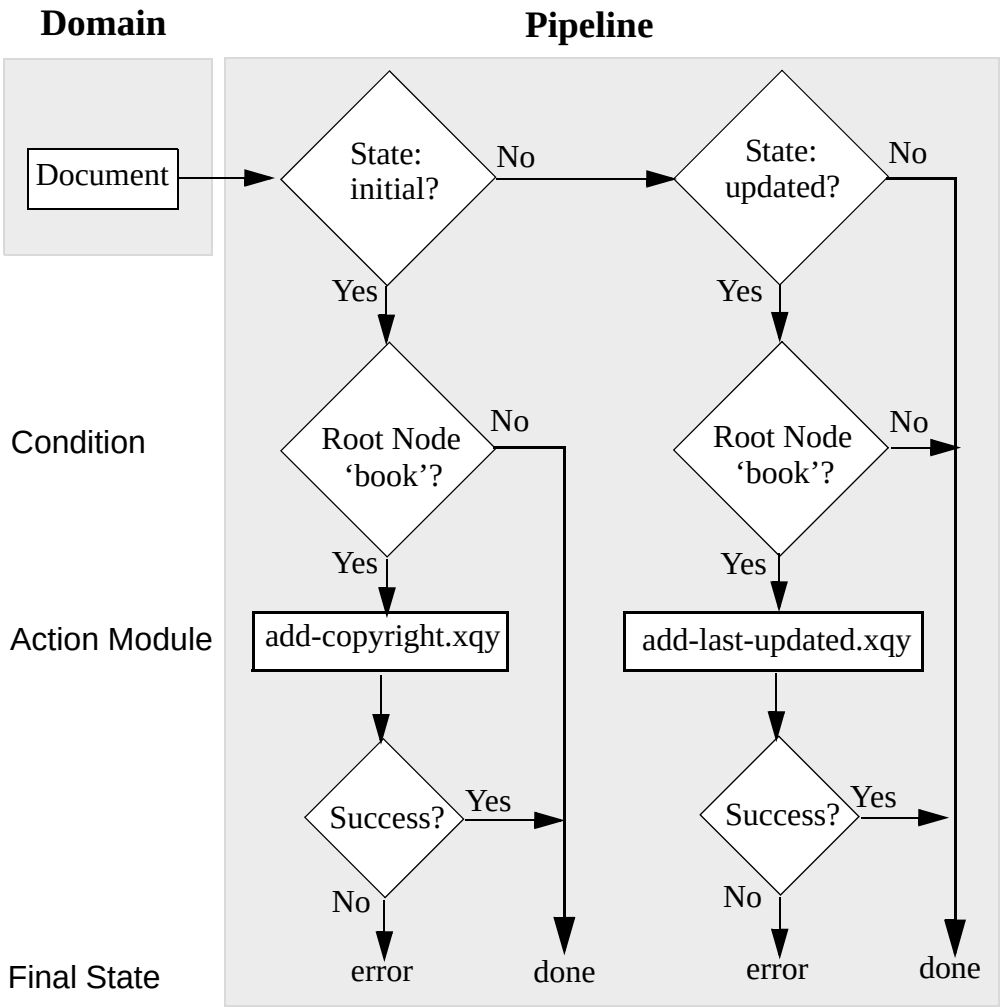
This chapter includes the following topics:

- Overview of Example CPF Application
- Create the Action Modules
- Insert the Action Modules into the Modules Database
- Create the Pipeline
- Configure a Database for Content Processing
- Insert and Update a Document in the Database
- View the Properties Document
- Extend the CPF Application

2.1 Overview of Example CPF Application

The pipeline portion of this CPF application detects when a document is initially inserted into the database and when a document is updated in the database and calls an action module to perform processing on the document. If a document is inserted into the database and has a root node of `book`, the pipeline calls an action module to insert a `copyright` node as a child of the root node. If a document is updated in the database and has a root node of `book`, the pipeline calls a different action module to insert a `last-updated` node as a child of the root node.

The following figure illustrates the logical flow of the example CPF application:



2.2 Create the Action Modules

The simple pipeline you create in “Create the Pipeline” on page 15 will call two action modules:

- add-copyright.xqy — inserts a `copyright` node into the document node.
- add-last-updated.xqy — inserts a `last-updated` node into the document node.

Create a directory, named `copyright`, somewhere in your file system. You will later use this location to load the CPF application files into MarkLogic Server.

2.2.1 add-copyright.xqy

The `add-copyright.xqy` action module inserts a `copyright` node as a child of the document’s `book` node. Copy the code below into a text editor and save as `add-copyright.xqy` in the `copyright` directory.

```
xquery version "1.0-ml";

import module namespace cpf = "http://marklogic.com/cpf"
  at "/MarkLogic/cpf/cpf.xqy";

declare variable $cpf:document-uri as xs:string external;
declare variable $cpf:transition as node() external;

if (cpf:check-transition($cpf:document-uri,$cpf:transition)) then try
{
  let $doc := fn:doc( $cpf:document-uri )
  return
      xdmp:node-insert-child(
        $doc/book,
        <copyright>
          <year>2010</year>
          <holder>The Publisher</holder>
        </copyright>),
      xdmp:log( "add copyright ran OK" ),
      cpf:success( $cpf:document-uri, $cpf:transition, () )
}
catch ($e) {
  cpf:failure( $cpf:document-uri, $cpf:transition, $e, () )
}
else ()
```

2.2.2 add-last-updated.xqy

The `add-last-updated.xqy` action module inserts a `last-updated` node as a child of the document's `book` node. Copy the code below into a text editor and save as `add-last-updated.xqy` in the `copyright` directory.

```
xquery version "1.0-ml";

import module namespace cpf="http://marklogic.com/cpf"
  at "/MarkLogic/cpf/cpf.xqy";

declare variable $cpf:document-uri as xs:string external;
declare variable $cpf:transition as node() external;

if (cpf:check-transition($cpf:document-uri,$cpf:transition)) then try
{
  let $doc := fn:doc($cpf:document-uri)
  return
      xdmp:node-insert-child(
        $doc/book,
        <last-updated>{fn:current-dateTime()}</last-updated>),
      ),
  xdmp:log( "add last-updated ran OK" ),
  cpf:success($cpf:document-uri, $cpf:transition, ())
} catch ($e) {
  cpf:failure($cpf:document-uri, $cpf:transition, $e, ())
}
else ()
```

2.3 Insert the Action Modules into the Modules Database

CPF locates the action modules in the modules database specified in the domain configuration set in Configure a Database for Content Processing, Step 6. In this example, the modules are placed in the Modules database.

Insert the `add-copyright.xqy` and `add-last-updated.xqy` modules in the `copyright` directory into the Modules database under the URI `/copyright/`*`module_name`*`.xqy`. For example, using Query Console, set the content-source to the Modules database and enter:

```
xquery version "1.0-ml";

xdmp:document-load("C:\copyright\add-copyright.xqy",
    <options xmlns="xdmp:document-load">
      <uri>/copyright/add-copyright.xqy</uri>
    </options>),

xdmp:document-load("C:\copyright\add-last-updated.xqy",
    <options xmlns="xdmp:document-load">
      <uri>/copyright/add-last-updated.xqy</uri>
    </options>)
```

2.4 Create the Pipeline

When CPF is enabled on a database, every document inserted into the database is given a CPF state. The CPF state is simply a label, stored as a document property, that identifies where the document is in relation to a set of processing steps. A pipeline manages the action modules applied to a document by transitioning the document from one state to another. A state transition says something like “whenever a document is moved into state A, do Y and then move the document into state B.”

CPF uses a special pipeline, called the Status Change Handling pipeline, to keep track of the status of a document during content processing and to set specific states on the document. For example, the Status Change Handling pipeline sets the state of a document to `initial` when it is first inserted into the database, to `updated` when it is updated, and cleans up links when it is deleted.

The example pipeline shown below detects when a document is in the `initial` state and calls the `add-copyright.xqy` module. When a document is in the `updated` state, the pipeline calls the `add-last-updated.xqy` module. If the document is successfully modified by an action module, the pipeline transitions the document to the `done` state; otherwise it transitions the document to the `error` state. Though it is a good practice to call an action module when an error occurs, this is omitted from this example for the sake of simplicity.

Copy the pipeline code below into a text editor and save as `copyright.xml` in the `copyright` directory.

```
<pipeline xmlns="http://marklogic.com/cpf/pipelines">

<pipeline-name>Copyright Pipeline</pipeline-name>
<pipeline-description>Pipeline to test CPF</pipeline-description>
<success-action>
  <module>/MarkLogic/cpf/actions/success-action.xqy</module>
</success-action>
<failure-action>
  <module>/MarkLogic/cpf/actions/failure-action.xqy</module>
</failure-action>
```

```
<state-transition>
  <annotation>
    When a document containing ‘book’ as a root element is created,
    add a ‘copyright’ statement.
  </annotation>
  <state>http://marklogic.com/states/initial</state>
  <on-success>http://marklogic.com/states/done</on-success>
  <on-failure>http://marklogic.com/states/error</on-failure>
  <execute>
    <condition>
      <module>/MarkLogic/cpf/actions/namespace-condition.xqy</module>
      <options xmlns="/MarkLogic/cpf/actions/namespace-condition.xqy">
        <root-element>book</root-element>
        <namespace/>
      </options>
    </condition>
    <action>
      <module>add-copyright.xqy</module>
    </action>
  </execute>
</state-transition>

<state-transition>
  <annotation>
    When a document containing ‘book’ as a root element is updated,
    add a ‘last-updated’ element
  </annotation>
  <state>http://marklogic.com/states/updated</state>
  <on-success>http://marklogic.com/states/done</on-success>
  <on-failure>http://marklogic.com/states/error</on-failure>
  <execute>
    <condition>
      <module>/MarkLogic/cpf/actions/namespace-condition.xqy</module>
      <options xmlns="/MarkLogic/cpf/actions/namespace-condition.xqy">
        <root-element>book</root-element>
        <namespace/>
      </options>
    </condition>
    <action>
      <module>add-last-updated.xqy</module>
    </action>
  </execute>
</state-transition>

</pipeline>
```

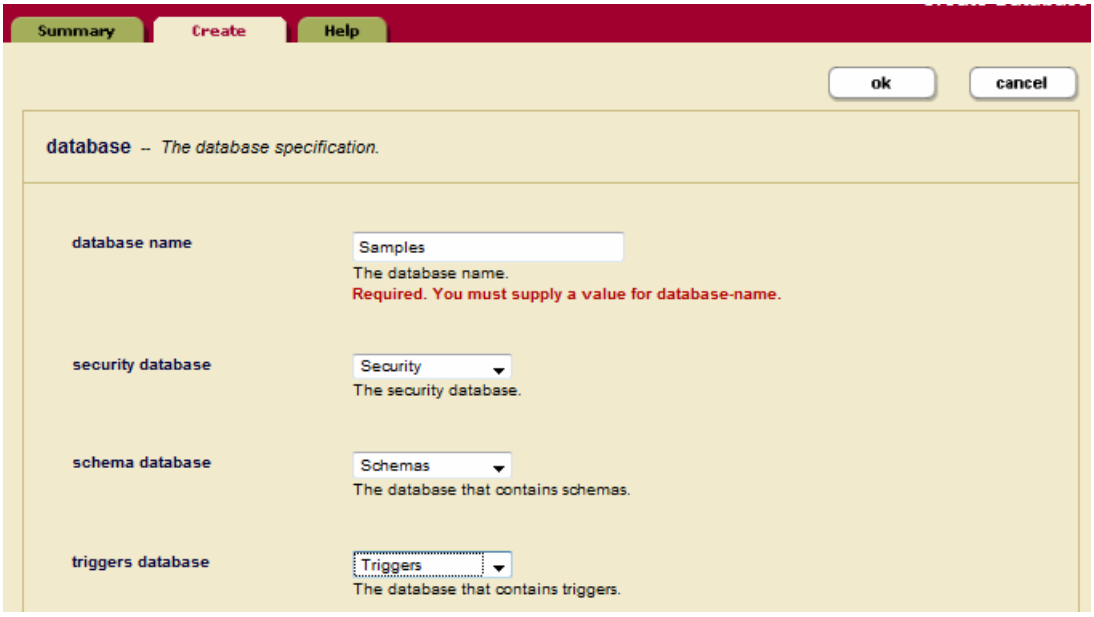
2.5 Configure a Database for Content Processing

This section describes how to create and configure a database that makes use of Content Processing. All of the basic procedures for creating and configuring a database are described in the Databases chapter in the *Administrator's Guide*.

Perform the following steps to create a database that uses CPF:

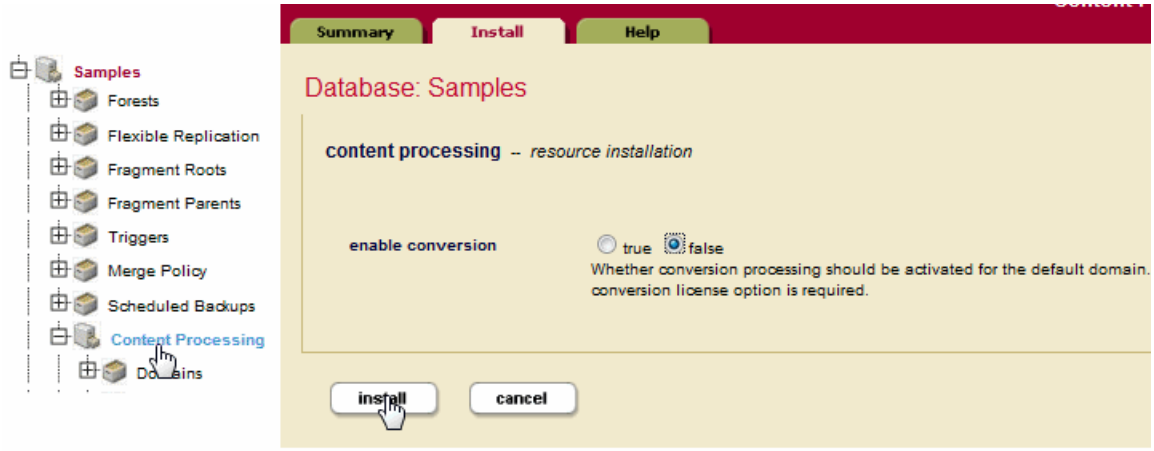
1. Create a forest named `Samples`.

2. Create a database named `Samples`, specifying `Triggers` as the triggers database, and attach the `Samples` forest to the `Samples` database.



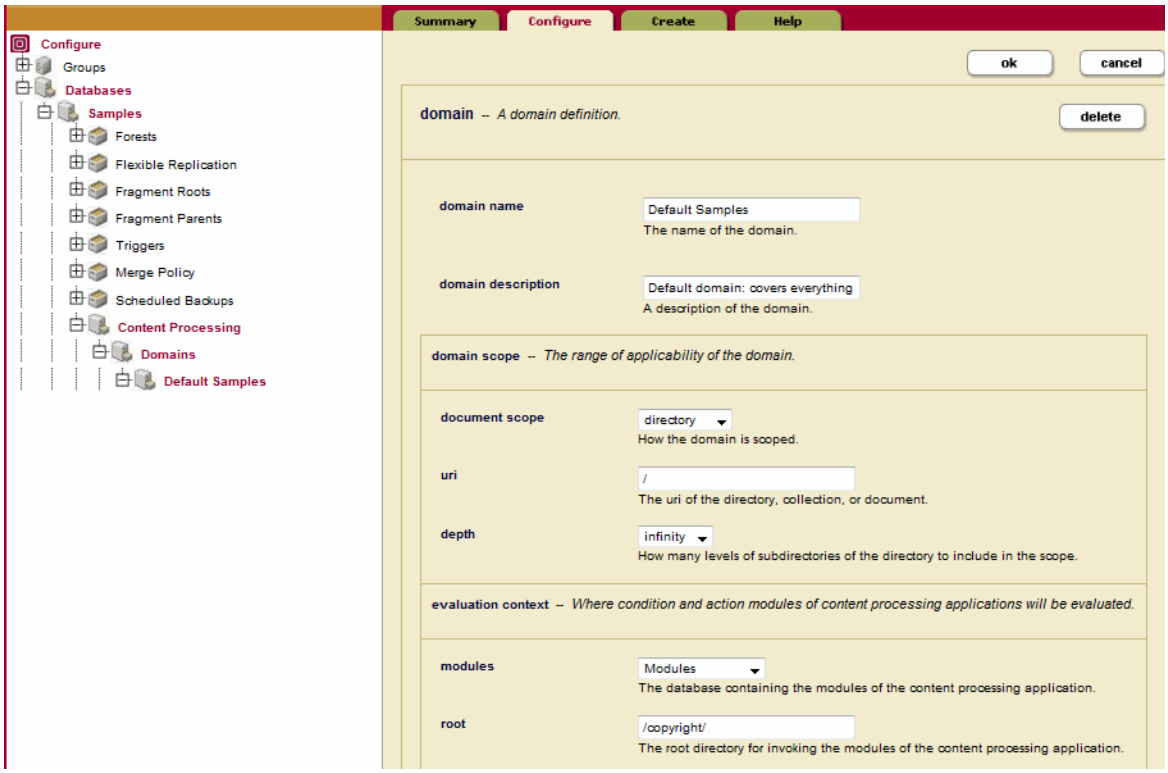
3. In the left tree menu, click `Content Processing` under the `Samples` database.

4. Click the `Install` tab, select `false` for enable conversion, and click the `Install` button to install content processing for the `Samples` database. This will enable content processing without the default conversion option.

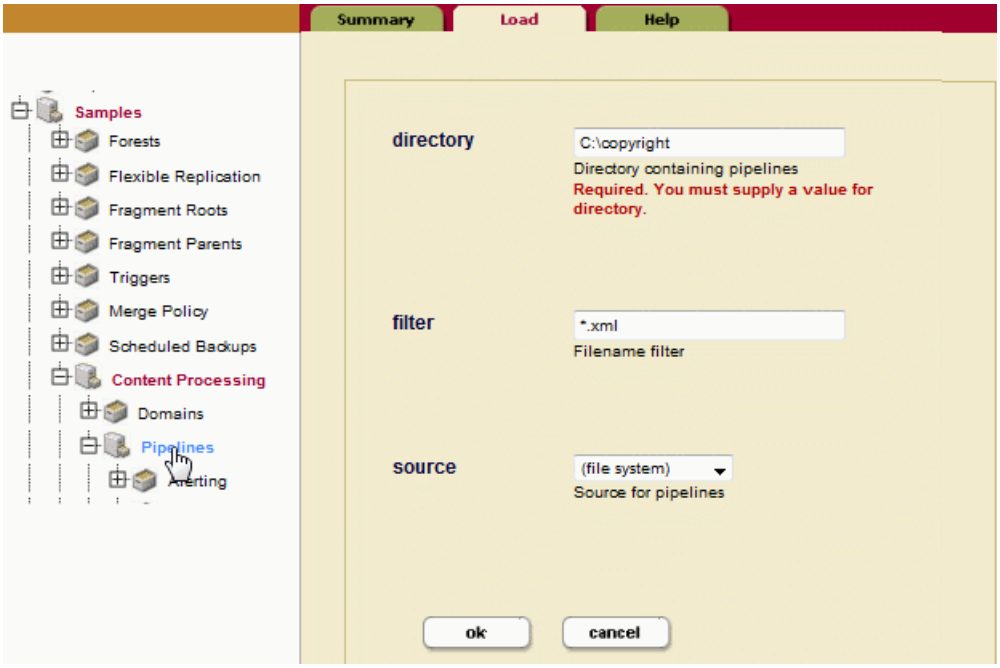


5. In the left tree menu, expand `Content Processing` under the `Samples` database, expand `Domains` and click `Default Samples`.

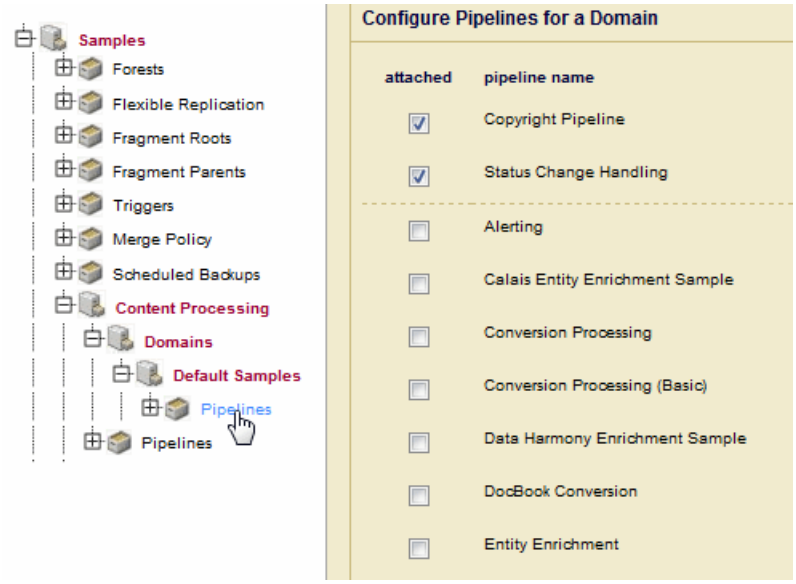
6. In the evaluation context section at the bottom, confirm that modules is set to Modules. In the root field, enter `/copyright/` to identify the base path of the action modules in the Modules database. Click `OK`.



7. In the left tree menu, under the `Samples` database, click `Pipelines`. In the directory field, identify the path to the directory in which you saved the `copyright.xml` file created in "Create the Pipeline" on page 15. Make sure the source is `(filesystem)`. Click `OK` to load the pipelines located in the `samples` directory into the `Triggers` database.



8. In the left tree menu, expand `Domains` and `Default Samples` under `Content Processing` and click `Pipelines`. Select the `Status Change Handling` and `Copyright Pipeline` to attach the pipelines to the `Default Samples` domain. Click `OK` when done.



Your CPF application is now configured and ready to respond to updates to the Samples database.

2.6 Insert and Update a Document in the Database

To see the results of the CPF pipeline, insert a document into the `Samples` database. For example, from Query Console, execute the following query against the `Samples` database.

> **Note:** The action modules will only enrich documents that have `book` as their root node.

```
xquery version "1.0-ml";

let $contents :=
<book>
  <bookTitle>All About George</bookTitle>
  <chapter1>
    <chapterTitle>Curious George</chapterTitle>
    <para>
       George Washington crossed the Delaware to see what was on the
other side.
    </para>
  </chapter1>
</book>

return
  xdmp:document-insert("/content/george.xml", $contents)
```

The pipeline detects that the document is in the `initial` state and calls the `add-copyright.xqy` action to insert a `copyright` node as a child of the `book` node. The `george.xml` document stored in the `Samples` database will look like the following:

```
<book>
  <bookTitle>All About George</bookTitle>
  <chapter1>
    <chapterTitle>Curious George</chapterTitle>
    <para>
       George Washington crossed the Delaware to see what was on the
other side.
    </para>
  </chapter1>
  <copyright>
    <year>2010</year>
    <holder>The Publisher</holder>
  </copyright>
</book>
```

Re-insert the document into the Samples database. The pipeline detects that the document is in the `updated` state and calls the `add-last-updated.xqy` action to insert a `last-updated` node as a child of the `book` node. The `george.xml` document stored in the Samples database will look like the following:

```
<book>
  <bookTitle>All About George</bookTitle>
  <chapter1>
    <chapterTitle>Curious George</chapterTitle>
    <para>
       George Washington crossed the Delaware to see what was on the
other side.
    </para>
  </chapter1>
  <last-updated>2009-11-10T13:28:20.144-08:00</last-updated>
</book>
```

2.7 View the Properties Document

Every document in MarkLogic Server is associated with a properties document. If a document has been processed by CPF, its properties document will hold the CPF status information for that document. If an error occurred while a document was being processed by CPF, the error information will be captured in the properties document.

For example, from Query Console, execute the following query against the `Samples` database to view the properties for the `george.xml` document:

```
xquery version "1.0-ml";

xdmp:document-properties("/content/george.xml")
```

If there were no CPF errors, the document properties will look like:

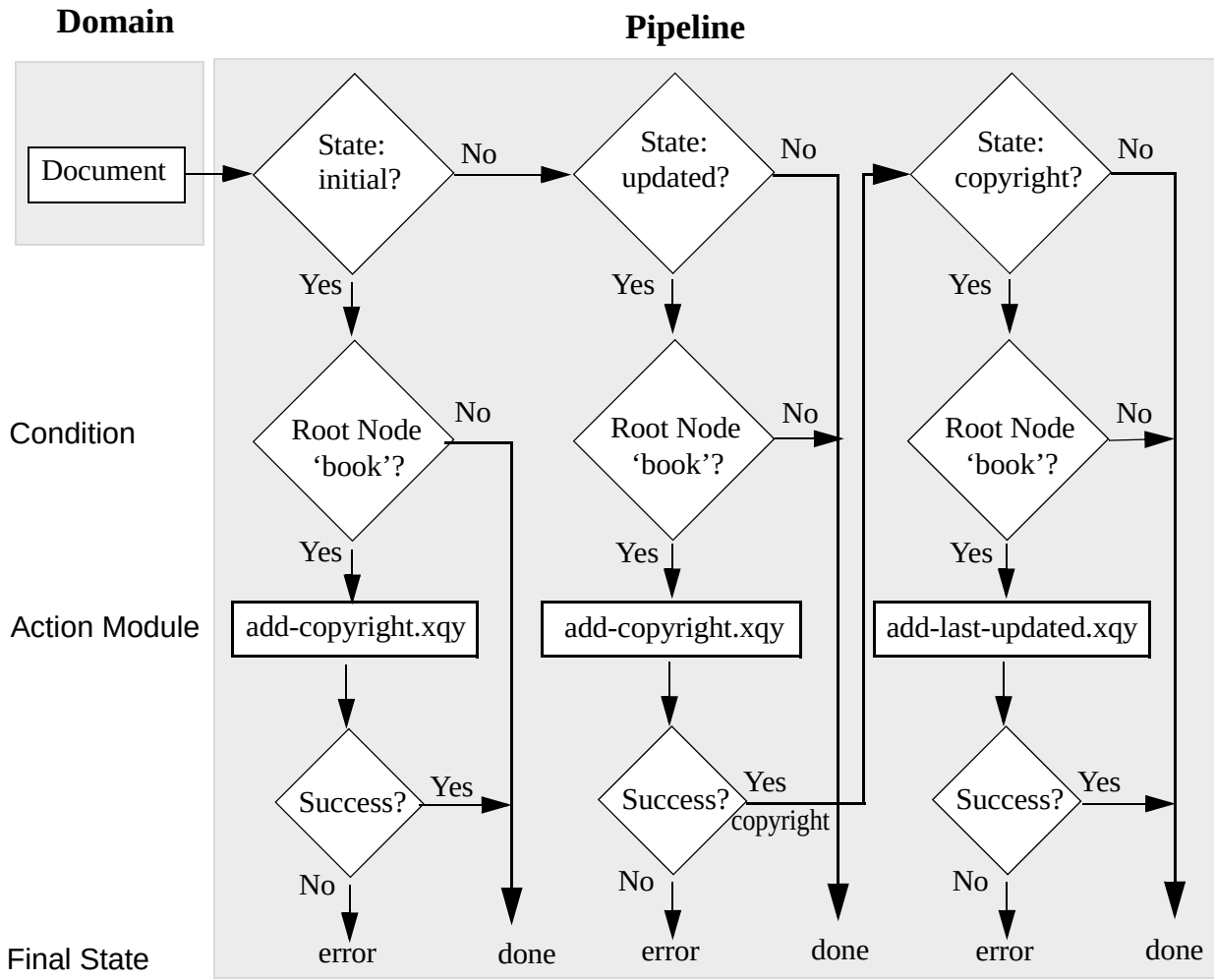
```
<?xml version="1.0" encoding="UTF-8"?>

<prop:properties xmlns:prop="http://marklogic.com/xdmp/property">
  <cpf:processing-status
xmlns:cpf="http://marklogic.com/cpf">done</cpf:processing-status>
   <cpf:property-hash
xmlns:cpf="http://marklogic.com/cpf">d41d8cd98f00b204e9800998ecf8427e<
/cpf:property-hash>
   <cpf:last-updated
xmlns:cpf="http://marklogic.com/cpf">2010-12-07T15:01:44.177-08:00</cp
f:last-updated>
   <cpf:state
xmlns:cpf="http://marklogic.com/cpf">http://marklogic.com/states/done<
/cpf:state>
   <prop:last-modified>2010-12-07T15:01:44-08:00</prop:last-modified>
</prop:properties>
```

2.8 Extend the CPF Application

A CPF pipeline often executes more than one action module to process a document. For example, you might want your CPF application to add both the `copyright` and `last-updated` nodes to documents when they are updated.

The example pipeline described in this section introduces a new state, `copyright`, which is set after the `updated` state-transition node successfully executes the `add-copyright.xqy` module. An additional state-transition node detects the `copyright` state and executes the `add-last-updated.xqy` module.

The following figure illustrates the logical flow of the extended CPF application:



Copy the pipeline code below into a text editor and save as `ex-copyright.xml` in the `copyright` directory.

Note: Changes to the previous pipeline example are highlighted in bold.

```
<pipeline xmlns="http://marklogic.com/cpf/pipelines">

<pipeline-name>Extended Copyright Pipeline</pipeline-name>
<pipeline-description>Pipeline to test CPF</pipeline-description>
<success-action>
  <module>/MarkLogic/cpf/actions/success-action.xqy</module>
</success-action>
<failure-action>
  <module>/MarkLogic/cpf/actions/failure-action.xqy</module>
</failure-action>

<state-transition>
  <annotation>
    When a document containing 'book' as a root element is created,
    add a 'copyright' statement.
  </annotation>
  <state>http://marklogic.com/states/initial</state>
  <on-success>http://marklogic.com/states/done</on-success>
  <on-failure>http://marklogic.com/states/error</on-failure>
  <execute>
    <condition>
      <module>/MarkLogic/cpf/actions/namespace-condition.xqy</module>
      <options xmlns="/MarkLogic/cpf/actions/namespace-condition.xqy">
        <root-element>book</root-element>
        <namespace/>
      </options>
    </condition>
    <action>
      <module>add-copyright.xqy</module>
    </action>
  </execute>
</state-transition>
```

```
<state-transition>
  <annotation>
    When a document containing 'book' as a root element is updated,
    add a 'copyright' element. When done, set the state to
    'copyright'.
  </annotation>
  <state>http://marklogic.com/states/updated</state>
  <on-success>http://marklogic.com/states/copyright</on-success>
  <on-failure>http://marklogic.com/states/error</on-failure>
  <execute>
    <condition>
      <module>/MarkLogic/cpf/actions/namespace-condition.xqy</module>
      <options xmlns="/MarkLogic/cpf/actions/namespace-condition.xqy">
        <root-element>book</root-element>
        <namespace/>
      </options>
    </condition>
    <action>
      <module>add-copyright.xqy</module>
    </action>
  </execute>
</state-transition>

<state-transition>
  <annotation>
    When the state is set to 'copyright', add a 'last-updated'
    element.
  </annotation>
  <state>http://marklogic.com/states/copyright</state>
  <on-success>http://marklogic.com/states/done</on-success>
  <on-failure>http://marklogic.com/states/error</on-failure>
  <execute>
    <condition>
      <module>/MarkLogic/cpf/actions/namespace-condition.xqy</module>
      <options xmlns="/MarkLogic/cpf/actions/namespace-condition.xqy">
        <root-element>book</root-element>
        <namespace/>
      </options>
    </condition>
    <action>
      <module>add-last-updated.xqy</module>
    </action>
  </execute>
</state-transition>

</pipeline>
```

To see the results of this new pipeline, do the following:

1. Load the `ex-copyright.xml` pipeline the same way you loaded the `copyright.xml` pipeline in Configure a Database for Content Processing, Step 7.

2. In the Default Samples domain Pipelines configuration page, un-attach the Copyright Pipeline and attach the Extended Copyright Pipeline, as described in Configure a Database for Content Processing, Step 8.

3. Insert and update the document as described in Insert and Update a Document in the Database and view the results.

3.0 Understanding and Using Domains

This chapter describes domains in the MarkLogic Server Content Processing Framework, and includes the following sections:

- Overview of Domains
- Domain Scope and Code Evaluation Context
- Rules for Domains
- Creating and Modifying Domains
- Attaching and Detaching Pipelines to Domains

3.1 Overview of Domains

Applications often perform the same content processing operations on a set of documents. For example, you might have a set of XML documents that come from one source, and you need to perform the same processing on each of those documents. The MarkLogic Server Content Processing Framework uses *domains* to describe a group of documents to which the same content processing operations should be applied.

You can use domains to partition sets of documents in meaningful ways, and apply the same content processing to all documents in a given set of documents. For example, if you have one set of documents that comes from Company X in a certain form, and another set of documents that comes from Company Y in a different form, you can define a different set of content processing for each set of documents. You can then create domains for each set of documents, running the appropriate sequence content processing operations on each type of document.

3.2 Domain Scope and Code Evaluation Context

You can view a content processing application as having two parts:

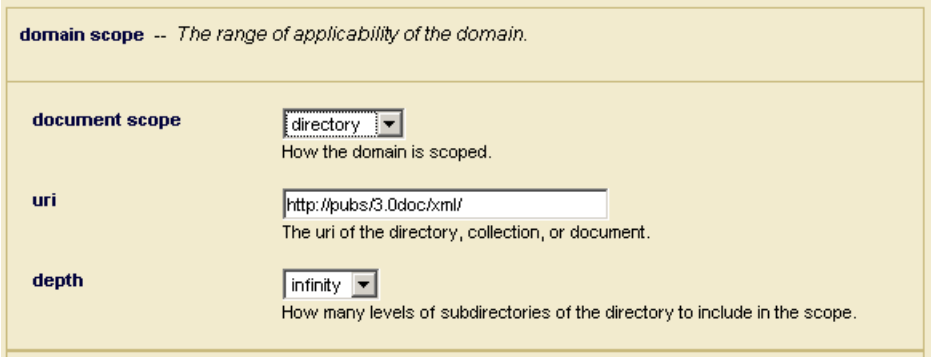
- Documents (content) processed by the application.
- Code that makes up the application.

Each domain includes configuration information for documents (answering the question: *which documents will be processed by this application?*) as well as configuration information for code (answering the question: *where is the code that makes up this application?*). This section describes both of these configurations associated with a domain and includes the following sections:

- Domain Scope
- Evaluation Context
- Domain Scope Can Encapsulate Processing Logic

3.2.1 Domain Scope

The *domain scope* specifies the documents to which this domain applies. The domain scope is defined in the Domain Configuration page of the Admin Interface.



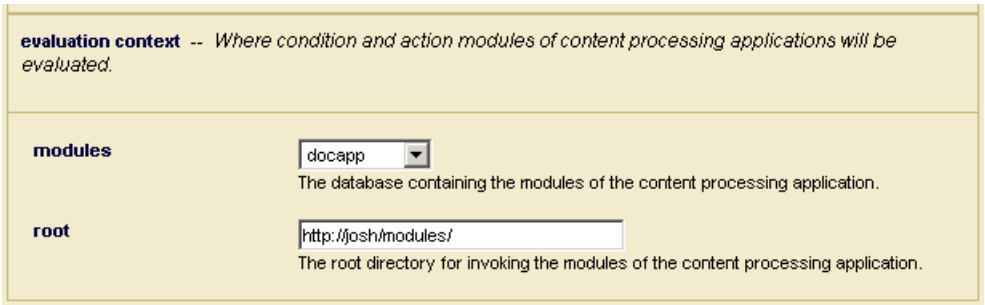
In the Admin Interface, the `document scope` drop-down list specifies whether the domain applies to a single document, a directory, or a collection. Each domain can only have one of these document scopes; if you need more than one of these document scopes, you can create multiple domains.

The `uri` field specifies the URI for the document, directory, or collection specified in the `document scope`.

The `depth` drop-down list applies only if you specify a `document scope` of `directory`, and you specify either `0` to indicate only documents in the immediate directory, or `infinity` to indicate documents in any directory that is a descendant of the specified directory URI.

3.2.2 Evaluation Context

When you create a domain, the Content Processing Framework automatically creates a set of triggers to listen for events (create, update, delete, property changes, and database online). The queries that the triggers execute run in the specified *evaluation context*. This is important because any content processing code that uses this domain also ends up evaluating its modules in this context.



Because the content processing code executes in the specified context, any module imports in the content processing code (for your condition and action modules, for example) are resolved relative to the specified database and URI root.

3.2.3 Domain Scope Can Encapsulate Processing Logic

Because content processing only occurs on documents within the scope of a domain, any content processing code can work under the assumption that all documents it sees require processing. This fact simplifies the processing code, as it does not need to include complex logic to determine if a document needs processing. The fact that a document is in the scope of a particular domain provides all the logic needed to determine that it needs processing.

3.3 Rules for Domains

This section describes the following rules for domains:

- Do Not Overlap Domains
- Collection Domain Scope Notes

3.3.1 Do Not Overlap Domains

If you use multiple domains, ensure that no two domains overlap; that is, a domain should not include any documents that are included in another domain. If you have overlapping domains, then it is possible for documents to be processed twice, which can cause unexpected results. For example, if you have a domain defined with infinite directory scope on the directory `/myDomain`, and if you have another domain with infinite directory scope on the directory `/myDomain/docs`, then any documents under the directory `/myDomain/docs` apply to both domains, so they would get processed twice. If you create an overlapping domain, the Admin Interface issues a warning. While it is possible to create them, MarkLogic recommends that you do not use overlapping domains.

> **Note:** If you are using collection scope in your domains, it might not always be obvious if your domains overlap. Documents can belong to multiple collections, and you can add or remove documents from a collection. Also, a new document can be created in a collection based on the default collections of the user who created the document. Be careful of unexpected overlapping domains when using collection scope domains.

3.3.2 Collection Domain Scope Notes

If you are using a collection-scope domain to specify which documents to convert, any new documents created by the conversion process must also be created as part of the collection specified in the domain. If they are not part of the collection, they will not be recognized by the domain for further processing.

The following are some of the ways you can ensure that documents are part of one or more collections:

- Set the `inherit collections` option at the database level to `true` and make sure the parent directory belongs to the collection.
- The user who initiates content processing (that is, the user who originally creates the documents to be processed, whether by drag and dropping into a WebDAV folder or by some other means) can have the collection specified as a default collection (or have the default collection attached to a role to which the user is assigned).
- You can explicitly set the collection on a document (for example, in your XQuery module code or through XDBC).

Collection domain scope is appropriate for some types of applications, particularly when you cannot always control the URI of the document.

> **Note:** Because domains with a collection scope can only continue the next phase of processing if the new or modified document is part of that collection, you can use collections as a way of moving documents in and out of different sets of processing.

3.4 Creating and Modifying Domains

You use the Admin Interface to create and modify domains. Perform the following steps to create a domain:

1. In the Admin Interface menu, click the Databases link and then click the name of the database to which you want to add a domain.

2. Under the database name, click Content Processing.

3. If content processing is already installed for your database, you will see links in the navigation tree for Domains and Pipelines. Click Domains.

   If content processing is not installed, install it as described in “Install Content Processing Framework in Database” on page 48.

4. Click the Create tab. The Domain Create page appears.

5. Enter a `domain name` and a `domain description`.

6. Specify the `domain scope`. For details on the domain scope, see “Domain Scope” on page 27.

7. Specify the `evaluation context`. For details on the evaluation context, see “Evaluation Context” on page 27.

8. Click OK.

The domain is created. To use the domain, you must attach a pipeline.

Similarly, you can use the Admin Interface to select an existing domain and modify its configuration information.

3.5 Attaching and Detaching Pipelines to Domains

To execute a pipeline, it must first be attached to a domain. For details about pipelines, see “Understanding and Using Pipelines” on page 31.

Perform the following steps to attach or detach pipelines to a domain:

1. In the Admin Interface, select the domain to which you want to add a pipeline (for example, Databases > myDatabase > myDomain).

2. Under the domain, select Pipelines. The Domain Pipeline screen appears

3. Check all the pipelines you want to attach to the domain and uncheck all the pipelines you want to detach from the domain. For most domains, you should select the `Status Change Handling` pipeline to attach, as well as any other pipelines you are using (including any custom pipelines you have created).

4. If you want to attach multiple pipelines to the domain, click the checkbox for others.

5. Click OK.

The attached pipelines appear at the top of the Domain Pipeline Configuration list. Note that a pipeline can be attached to multiple domains simultaneously.

4.0 Understanding and Using Pipelines

This chapter describes pipelines in the MarkLogic Server Content Processing Framework, and includes the following sections:

- Pipeline Architecture
- Viewing Pipelines in the Admin Interface
- Loading Pipelines With the Admin Interface
- XML Format of a Pipeline
- XQuery Functions to Manage Pipelines

4.1 Pipeline Architecture

A core component of the Content Processing Framework is the *pipeline*. A pipeline is an XML document that defines document states as a document moves through stages of content processing. In addition to defining document states, a pipeline specifies *actions* that occur under certain *conditions*. A condition is an XQuery module or an XSLT stylesheet that evaluates to `true` or `false`. An action is an XQuery module or an XSLT stylesheet that is called when the condition associated with an action is either `true` or if there is no condition.

This section includes the following topics about pipelines:

- Automatic Status (Event) Handling
- Transitioning Between States
- Pipelines Can Flow Through Other Pipelines

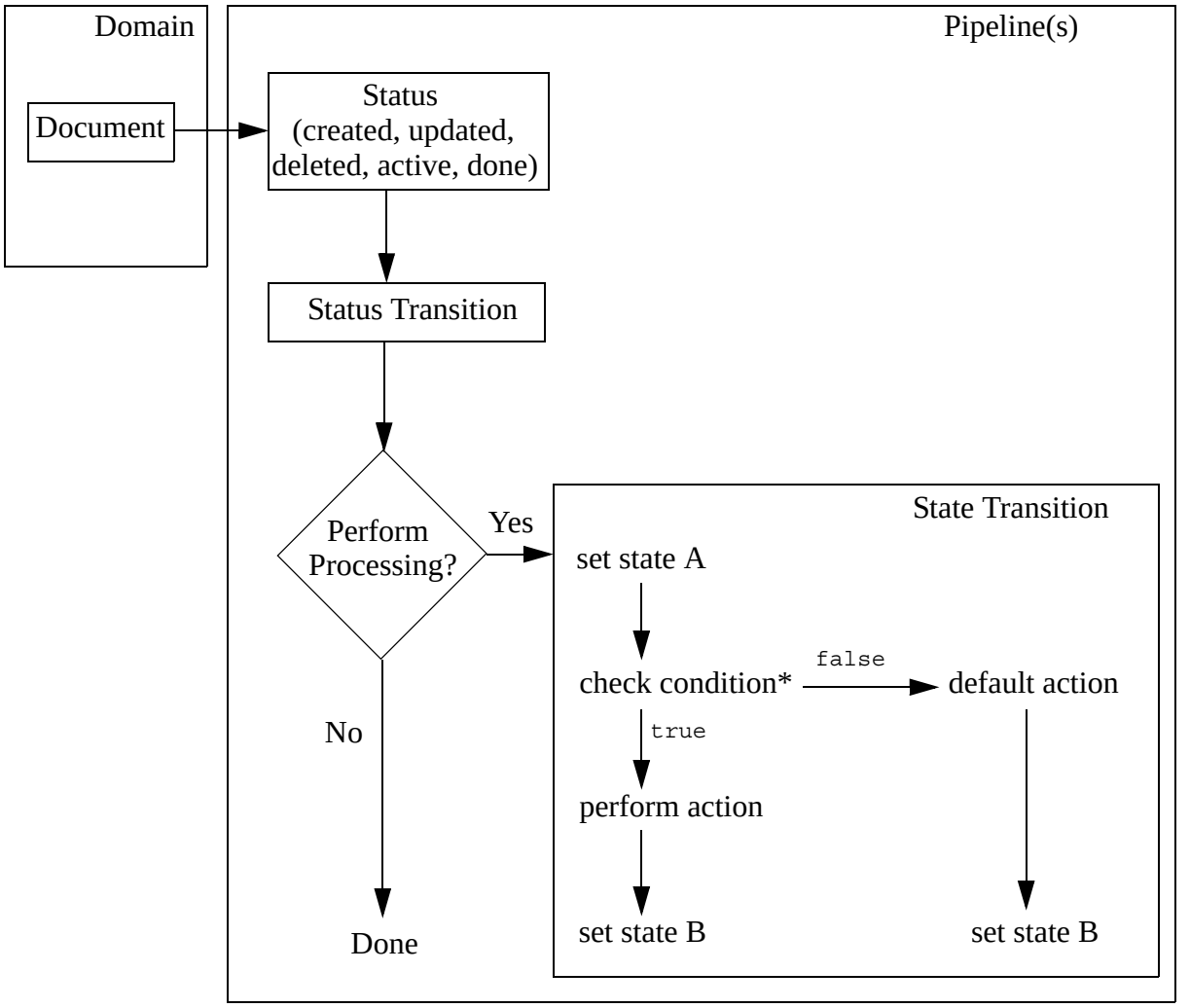
4.1.1 Automatic Status (Event) Handling

Robust content processing applications must be able to gracefully move documents in and out of active processing, and they must be able to cleanly recover from failures. Common reasons to move a document in or out of active processing are when a document is first created or when it changes, for example upon a create, update, or delete event. When you want the result of these changes to move a document in or out of document processing, it is known as a *status change*.

When a document that is in a content processing domain is created, updated, or deleted, the Content Processing Framework automatically handles the document status change events (create, update, and delete) and sets a state (or cleans up in the case of delete) for the document. This has the effect of activating content processing for that document. Similarly, if there is a failure that causes the system to go down, the content processing must recover gracefully. The Status Change Handling pipeline, installed when you install content processing in a database, performs these tasks automatically.

4.1.2 Transitioning Between States

You attach pipelines to domains, and the domains determine the documents on which a pipeline acts. The pipeline then facilitates the transitioning of the document from one state to another, calling XQuery modules to perform the content processing between states. The following figure shows how a simplified pipeline can move a document from one state to another.

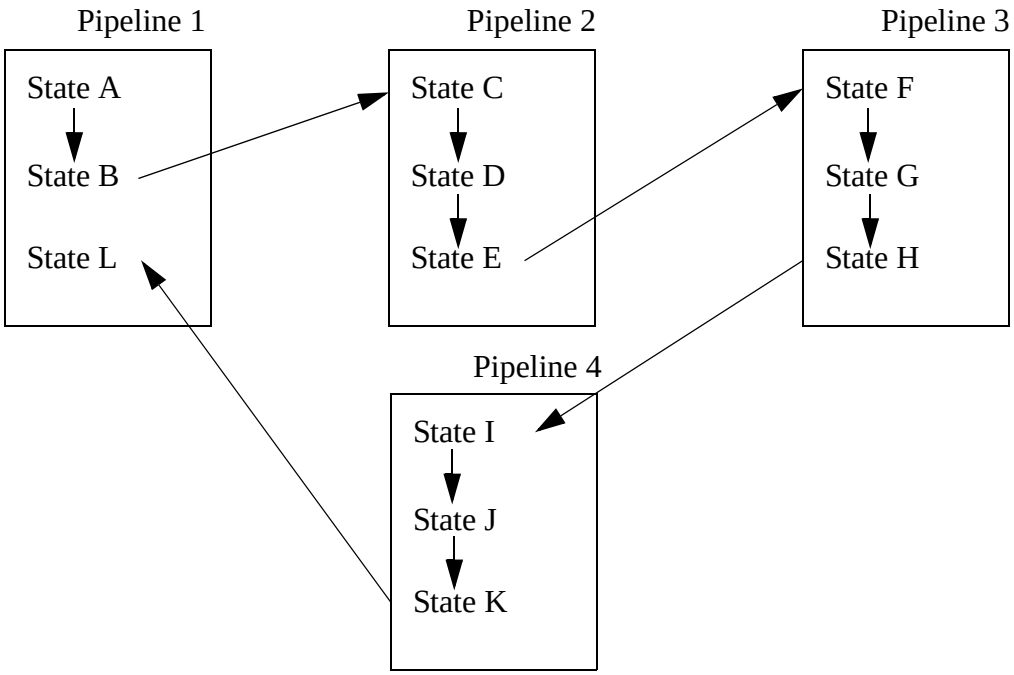


* pipelines can have multiple conditions

Note: While setting a state on a document is a common outcome of a status transition or a state transition, it is not a requirement.

4.1.3 Pipelines Can Flow Through Other Pipelines

All of your content processing states need not be defined by a single pipeline; document processing can flow from one pipeline to another. The following figure shows how states can flow in and out of different pipelines.



Having the processing flow through multiple pipelines allows for flexibility and modularity in the way you design your pipelines. For example, you might have a pipeline that defines states that are common to several kinds of processing. Factoring out the common processing stages into a separate pipeline allows you to have different pipelines transition into a common pipeline used by multiple applications. There is enough flexibility to set up your pipelines to be very simple or very complex.

4.2 Viewing Pipelines in the Admin Interface

You can use the Admin Interface to view pipelines already loaded in a database. To view an existing pipeline, navigate to Databases > *database_name* > Content Processing > Pipelines > *pipeline_name* in the Admin Interface tree menu. The Admin Interface displays the following information for each pipeline:

Display Item	Description
Pipeline name	The name of the pipeline.
Pipeline description	A description of the pipeline, if one exists in the pipeline.
If document *created*	The logic to follow after the status of a document is set to `created` (typically after a document create event occurs).
If document *updated*	The logic to follow after the status of a document is set to `updated` (typically after a document update event occurs).
If document *deleted*	The logic to follow after the status of a document is set to `deleted` (typically after a document delete event occurs).
on-success action	The path to the XQuery module or XSLT stylesheet called after any stage of the pipeline completes successfully (after it calls `cpf:success`). The on-success action should always call the `success-action.xqy` module.
on-failure action	The path to the XQuery module or XSLT stylesheet called after any stage of the pipeline does not complete successfully (after it calls `cpf:failure`). The on-failure action should always call the `failure-action.xqy` module.
when	The path to the XQuery module or XSLT stylesheet called to test a condition for a pipeline stage.
do	The path to the XQuery module or XSLT stylesheet called when the above “when” condition returns `true`.
otherwise	The path to the XQuery module or XSLT stylesheet to execute if no other action is called.
always	The path to the XQuery module or XSLT stylesheet to execute if no other conditions or actions are specified.

Note: The URI paths to XQuery modules in a pipeline are relative to the module root specified in the domain to which the pipeline is attached. You should load the XQuery modules into the specified database as a stored module, with execute permissions for users that need to run it. It is also possible to reference modules stored in the modules directory on the filesystem, but MarkLogic recommends loading your modules under the database root specified in the domain.

4.3 Loading Pipelines With the Admin Interface

You can use the Admin Interface to load XML files into the database as pipelines. Pipelines are stored in the triggers database, therefore any pipelines you load for a given database through the Admin Interface are loaded into the triggers database for that database. Note the following about loading pipelines:

- If there is no triggers database configured, then an error will occur when you try to load the pipeline.
- Pipeline names must be unique.
- If you load a pipeline with the same name as an existing pipeline, the new pipeline definition will replace the old one.

Perform the following steps to load a pipeline into the database:

1. Create a valid pipeline XML document and save it to a filesystem accessible from the machine on which the Admin Interface is running. For details on the XML format of a pipeline, see "XML Format of a Pipeline" on page 36.

2. In the Admin Interface menu, click the Databases link and then click the name of the database to which you want to load a pipeline.

3. Under the database name, click Content Processing.

4. If content processing is already installed for your database, you will see links for Domains and Pipelines. Click Pipelines.

   If content processing is not installed, install it as described in "Install Content Processing Framework in Database" on page 48.

5. Click the Load tab. The Pipeline Load page appears.

6. Enter the directory where your pipeline XML file is stored.

7. Enter a wildcard filter to search on. For example, entering `*.xml` will look for all pipeline files in the directory with filenames ending with `.xml`.

8. Click OK.

9. On the Pipeline Load confirmation page, examine the name of the pipeline(s) displayed. Only XML documents that are in the `http://marklogic.com/cpf/pipelines` namespace are displayed. If the pipeline displayed is correct, click OK to load the pipeline into the database.

The pipeline is loaded into the triggers database associated with the database in which you are defining content processing. If the pipeline does not conform to the `pipelines.xsd` schema, the load will fail.

4.4 XML Format of a Pipeline

A pipeline is an XML document, and it must conform to the `pipelines.xsd` schema, located as follows:

```
<install_dir>/Config/pipelines.xsd
```

The pipeline document defines the properties of the pipeline, including the pipeline name, the success and failure actions, any status transitions, any state transitions, and all of the actions associated with the various stages of the pipeline. Once you create the XML pipeline document, you must load it into the database to use it, as described in “Loading Pipelines With the Admin Interface” on page 35.

This section describes the pipeline XML format and includes the following topics:

- Sample Pipeline XML Document
- Success-Action and Failure-Action
- Status Transitions
- State Transitions

4.4.1 Sample Pipeline XML Document

The following is a sample pipeline XML document.

```
<!-- Copyright 2002-2010 MarkLogic Corporation.  All Rights Reserved.
-->
<?xml-stylesheet href="/cpf/pipelines.css" type="text/css"?>
<pipeline xmlns="http://marklogic.com/cpf/pipelines"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://marklogic.com/cpf/pipelines pipelines.xsd">
  <pipeline-name>HTML Conversion</pipeline-name>
  <pipeline-description>Additional conversion rules for HTML.
   This pipeline should be used in conjunction with the basic
   conversion pipeline and the status change handling pipeline.
  </pipeline-description>
  <success-action>
    <module>/MarkLogic/cpf/actions/success-action.xqy</module>
  </success-action>
  <failure-action>
    <module>/MarkLogic/cpf/actions/failure-action.xqy</module>
  </failure-action>
  <state-transition>
    <annotation> Convert HTML documents and only HTML documents.
    </annotation>
    <state>http://marklogic.com/states/initial</state>
    <on-success>http://marklogic.com/states/converted</on-success>
    <on-failure>http://marklogic.com/states/error</on-failure>
    <priority>9200</priority>
    <execute>
      <condition>
       <module>/MarkLogic/cpf/actions/mimetype-condition.xqy</module>
       <options xmlns="/MarkLogic/cpf/actions/mimetype-condition.xqy">
         <mime-type>text/html</mime-type>
       </options>
       </condition>
       <action>
       <module>/MarkLogic/conversion/actions/convert-html-action.xqy</module>
       <options
        xmlns="/MarkLogic/conversion/actions/convert-html-action.xqy">
          <destination-root/>
          <destination-collection/>
       </options>
       </action>
    </execute>
  </state-transition>
    <!-- States converted and error not handled here -->
</pipeline>
```

4.4.2 Success-Action and Failure-Action

The success-action and failure-action elements in the pipeline are where you specify the clean-up activities to perform when the processing for a state or status transition action succeeds or when it fails. The failure-action is only called if a transition fails; when a status or state transition action succeeds, it is not called. The success-action is called only if no success action is specified in a status or state transition, and if the transition has no default action. The `success-action.xqy` and `failure-action.xqy` XQuery modules are designed to handle these actions, calling the functions `cpf:success` or `cpf:failure` to advance the state as appropriate.

Your XQuery action modules or XSLT stylesheets for state transitions should call `cpf:success` or `cpf:failure` to advance the state, as described in "Action Modules Use try/catch With cpf:success and cpf:failure" on page 45. The default success-action and failure-action modules are only called either if there is no action specified in a state transition or if the pipeline catches an exception.

You should use the default actions in all of your pipelines. While it is possible to create your own success/failure actions, MarkLogic recommends using the default `success-action.xqy` and `failure-action.xqy` XQuery modules for these actions.

4.4.3 Status Transitions

A *status transition* is an action that happens after a document has a status change (for example, document create, update, or delete). The Status Change Handling pipeline, installed when you install content processing in a database, keeps track of the status during content processing.

The Status Change Handling pipeline sets the state of a document to `http://marklogic.com/states/initial` on creation (unless the document is created with an initial state), sets the state to `http://marklogic.com/states/updated` on update, and cleans up links on delete.

Warning You should not need to create any of your own status transitions. While it is possible to create your own status transitions, MarkLogic recommends using the status transitions supplied in the Status Change Handling pipeline. Changing the Status Change Handling pipeline can cause compatibility problems in future upgrades and releases of MarkLogic Server.

Status transitions have the following parts:

- Status
- On Success and On Failure
- Default Action
- Priority
- Execute

4.4.3.1 Status

The status is defined by the Status Change Handling pipeline, and has the following possible values:

- created
- updated
- deleted
- active
- done

The Status Change Handling Pipeline and corresponding XQuery modules automatically handle status changes. Setting the status in your application code, especially on a document whose status is anything except `done`, can cause unexpected behavior; do not set the status in your application code.

4.4.3.2 On Success and On Failure

The on-success and on-failure part of the status transition defines the next state if the status transition is successful (in the case of success) and defines the next state if the status transition is not successful (in the case of failure). In each case, they reference an XQuery module that is called when the status transition succeeds or fails. If you do not specify an on-success or on-failure state, the document remains in its current state after the status transition success or failure.

4.4.3.3 Default Action

The default action references an XQuery module that is called if no other actions are activated. The status transition will execute the first of the following that occurs:

- The action whose condition in the status transition returns `true` or is absent.
- The default action for any transition for this status.
- The success-action.

4.4.3.4 Priority

The priority is used to determine which pipeline should be executed first in the event that there are multiple pipelines attached to a domain that act on the same status. Status transitions with a higher priority number execute before ones with a lower number. In the event of a tie (that is, if two priorities are the same number), it is indeterminate which one will execute first. For the pipelines supplied with MarkLogic Server, the order is set such that conversion executes first, then entity enrichment, then modular documents (xinclude), then alerting, and finally schema validation.

4.4.3.5 Execute

The execute part of the status transition runs the XQuery module referenced in the condition, and then runs the XQuery module referenced by the action if the condition returns `true` or if there is no condition specified. If there are multiple execute elements in a single status transition, you should design them so that at most one execute element has a condition that returns `true`. If multiple conditions return `true`, then the first one to return `true` has its action executed. The reason for this is that all of the XQuery modules execute in separate transactions, and it is non-deterministic which one will return first. Therefore, if you have multiple conditions that return `true`, either the first one will come first (in the case of a single pipeline) or you cannot guarantee which one will come first and execute its action (in the case of multiple pipelines).

> **Note:** If a condition fails (for example, throws an exception), the condition is handled as if it returns `false`.

Execute nodes for status transitions can also include options nodes. The options nodes are the same as the ones for state transitions, described in "Execute" on page 42.

4.4.4 State Transitions

Pipelines that define your content processing are generally made up of one or more *state transitions*. A state transition performs some work and then moves a document from one state to another. Each state is stored as a property in the properties document corresponding to the document URI. You define success or failure states in the on-success or on-failure part of the transition definition.

The state transitions have the following parts:

- State
- On Success and On Failure
- Default Action
- Priority
- Execute

4.4.4.1 State

The *state* is a string that is stored in a properties document. A state can be any valid URI. States are used by pipelines to begin a state transition. When the transition is complete, the pipeline specifies a new state for the document. The new state, in turn, is caught by another state transition, and so on until there are no state transitions for the new document state.

4.4.4.2 On Success and On Failure

The on-success and on-failure part of the state transition is where you specify the state to which the document is set if the state transition returns successfully (on-success) or if it fails (on-failure). If you do not specify an on-success or on-failure state, the document remains in its current state after the state transition success or failure, thereby completing processing for that document.

4.4.4.3 Default Action

The default action references an XQuery module that is called if no other actions are activated. The state transition will execute the first of the following that occurs:

- Any action whose condition in the state transition returns `true` or is absent.
- The default action for any transition for this state.
- The success-action.

You can use the default action to move a document to the next state if the processing you want to perform is the default action. The following is a sample state-transition node that contains a default action:

```
<state-transition>
  <annotation>
    Default action example
  </annotation>
  <state>http://marklogic.com/states/initial</state>
  <on-success>http://marklogic.com/states/collected</on-success>
  <on-failure>http://marklogic.com/states/error</on-failure>
  <priority>5000</priority>
  <default-action>
    <module>/pipeline/mypipes/default-action.xqy</module>
  </default-action>
 </state-transition>
```

If you want the move the state to one that is different from the state transition's on-success state, you can use the `$override-state` parameter to the `cpf:success` function in your default action XQuery module. You should move the state to a different state from the document's current state. An example of a module that does this is the `/MarkLogic/cpf/actions/state-setting-action.xqy` under the `Modules` directory.

4.4.4.4 Priority

The priority is used to determine which pipeline state should be executed first in the event that there are multiple pipelines states attached to a domain that act on the same state. State transitions with a higher priority number execute before ones with a lower number. In the event of a tie (that is, if two priorities are the same number), it is indeterminate which one will execute first. For the pipelines supplied with MarkLogic Server, the order is set such that conversion executes first, then entity enrichment, then modular documents (xinclude), then alerting, and finally schema validation.

4.4.4.5 Execute

The execute part of the state transition runs the XQuery module or XSLT stylesheet referenced in the condition, and then runs the XQuery module or XSLT stylesheet referenced by the action if the condition returns `true` or if there is no condition specified. If there are multiple execute elements in a single state transition, you should design them so that at most one execute element has a condition that returns `true`. If multiple conditions return `true`, then the first one to return `true` has its action executed. The reason for this is that all of the XQuery modules or XSLT stylesheets execute in separate transactions, and it is non-deterministic which one will return first. Therefore, if you have multiple conditions that return `true`, either the first one will come first (in the case of a single pipeline) or you cannot guarantee which one will come first and execute its action (in the case of multiple pipelines).

> **Note:** If a condition fails (for example, throws an exception), the condition is handled as if it returns `false`.

You can also include an options node, which allows you to pass an external variable and/or an options node to code in the action XQuery moduleor XSLT stylesheet. The options node should have the namespace of the action module, or the namespace of the options node called in the XQuery function (`xdmp:tidy`, for example). The following is a sample options node within a condition:

```
<condition>
   <module>/MarkLogic/cpf/actions/mimetype-condition.xqy</module>
   <options xmlns="/MarkLogic/cpf/actions/mimetype-condition.xqy">
      <mime-type>text/html</mime-type>
   </options>
</condition>
```

The pipeline passes the options node as an external variable (`$cpf:options`) to the module that tests for the condition (`mimetypes-condition.xqy`, in this sample).

4.5 XQuery Functions to Manage Pipelines

The Admin Interface provides all of the functionality for loading pipelines. However, if you want to load and manage pipelines without using the Admin Interface, the `pipelines.xqy` XQuery module contains functions to manage pipelines. This XQuery file is installed into the following location:

```
<install_dir>/Modules/MarkLogic/cpf/pipelines.xqy
```

For details on the functions in this module, see the *MarkLogic XQuery and XSLT Function Reference*.

5.0 Developing Modules to Process Content

This chapter describes modules in the MarkLogic Server Content Processing Framework, and includes the following sections:

- Overview of Modules
- Loading Modules Into the Database
- External Variables Available to Modules
- Design Patterns and Rules

5.1 Overview of Modules

While domains and pipelines provide a framework to build content processing applications, the actual work of transforming or enhancing content is done through XQuery modules. An XQuery module can do an arbitrary amount of work; it can be very simple, very complex, or somewhere in between.

There are two types of XQuery modules used in content processing applications:

- Condition modules
- Action modules

Condition modules test for a condition and return a boolean value (`true` or `false`). Action modules typically perform some processing on the document and then call `cpf:success` or `cpf:failure` to advance the state, according to the definitions in the current pipeline.

Action modules can call out to web services inside or outside of MarkLogic Server, they can perform transformations on the document within MarkLogic Server, or they can perform any work needed in the current phase of the pipeline.

> **Note:** Instead of using XQuery modules for conditions and actions, you can also use XSLT modules. If you have a path to an XSLT stylesheet in a pipeline, the action or condition is run with the specified XSLT code. For details about using XSLT stylesheets in CPF actions, see “Using XSLT Stylesheets Instead of Action Modules” on page 47.

5.2 Loading Modules Into the Database

XQuery modules used as action and condition modules should be stored in the database. Load the XQuery modules into the database and root specified in the evaluation context for the domain configuration. For details about where to load the XQuery modules, see “Domain Scope and Code Evaluation Context” on page 26.

5.3 External Variables Available to Modules

Each CPF action condition and action module has the following external variables available:

- `$cpf:document-uri`: The URI of the document being processed. In Server-Side JavaScript, the name of this variable is `uri`.
- `$cpf:transition`: The name of the transition being executed. Every action should use this external variable so it can pass the value into `cpf:success` and `cpf:failure` to advance the state of the document. In Server-Side JavaScript, the name of this variable is `transition`.
- `$cpf:options`: The options XML node from the pipeline action. You can use this to pass in options that the module uses, so you can use the same module with different pipelines and get different behavior. In Server-Side JavaScript, the name of this variable is `options`.

You use these external variables to get the URI of the document being processed, the name of the transition being executed, and any options that the pipeline passes through to the module. To use these external variables in your XQuery modules, import them as external variables to your modules, as in the following code snippet from an XQuery `1.0-ml` prolog:

```
declare variable $cpf:document-uri as xs:string external;
declare variable $cpf:transition as element() external;
declare variable $cpf:options as element() external;
```

To use the external variables in your Server-Side JavaScript code, you need to require the `cpf.xqy` module and declare the variables in your module, as in the following example:

```
declareUpdate();
var cpf = require("/MarkLogic/cpf/cpf.xqy");
var uri;
var transition;
if (cpf["check-transition"](uri,transition)) {
  try {
    xdmp.nodeInsertChild(cts.doc(uri).xpath("/book"),
       xdmp.unquote('<copyright><year>2010</year><holder>The
Publisher</holder></copyright>').next().value.root);
    xdmp.log("add copyright ran OK");
    cpf["success"](uri,transition,null);
  } catch (e)
  {
    cpf["failure"](uri,transition,e,null);
  }

} else {
}
```

5.4 Design Patterns and Rules

The Content Processing Framework is designed with certain assumptions about what the modules called in a pipeline will do. This section describes these rules and provides XQuery design patterns to help you follow the rules in your XQuery code. It is important to follow these rules in your XQuery modules; not following these rules can lead to unexpected results. The following topics are included:

- Condition Modules Must Return a Boolean
- Condition Modules Should Not Update Documents
- Action Modules Use try/catch With cpf:success and cpf:failure
- Action Modules Operate On a Single Document
- Action Modules Must Be a Single Transaction
- Using XSLT Stylesheets Instead of Action Modules

5.4.1 Condition Modules Must Return a Boolean

Condition modules must return a boolean value (`true` or `false`).

In one common scenario, a condition module checks either the existence or the value of an element in the document or in its properties document. If it exists, then the module returns `true` and the document needs processing for the current phase of the pipeline. Another scenario is that the condition performs some specialized logic based on some part of the document contents. The logic does not even need to pertain to the document, as long as the module returns `true` or `false`.

5.4.2 Condition Modules Should Not Update Documents

A condition module should only return a boolean value—it should not perform any other work. This is an assumption of the pipeline design. If you do perform updates in conditions, it will change the document from the state it was in when the event occurred (potentially causing an additional trigger to fire). Do not perform any document updates in a condition module; doing updates in conditions can cause non-deterministic behavior.

5.4.3 Action Modules Use try/catch With cpf:success and cpf:failure

The mechanism for transitioning a document from one state to the next is carried out by two functions in the `cpf.xqy` XQuery module: `cpf:success` and `cpf:failure`. These functions handle the logic to advance a document either to the on-success or to the on-failure state specified in the pipeline.

Action modules must call either `cpf:success` or `cpf:failure` exactly once. By using these functions in a `try`/`catch` expression in your XQuery code, it is easy to either advance the document to the success state when the code runs without exceptions, or to catch any exceptions in the XQuery code and then put the document in a failure state. This ensures that exactly one of these functions is called. In the XQuery code, you concatenate the `cpf:success` call following your action code, and return `cpf:failure` in the `catch` clause (which is run only if the `try` clause throws an exception).

The `try`/`catch` expression ensures that the state is advanced in the same transaction that performs the document processing. This way, if processing is interrupted, the state of the document always matches the actual state of the content processing.

The following sample code (from the `link-rename-action.xqy` module) shows how to use `cpf:success` and `cpf:failure` in a try/catch expression.

```
xquery version "1.0-ml";
import module namespace cpf = "http://marklogic.com/cpf"
   at "/MarkLogic/cpf/cpf.xqy";
import module namespace lnk = "http://marklogic.com/cpf/links"
   at "/MarkLogic/cpf/links.xqy";

declare variable $cpf:document-uri as xs:string external;
declare variable $cpf:transition as node() external;

try {
   lnk:propagate-rename( $cpf:document-uri )
   ,
   cpf:success( $cpf:document-uri, $cpf:transition, () )
}
catch ($e) {
   cpf:failure( $cpf:document-uri, $cpf:transition, $e, () )
}
```

5.4.4 Action Modules Operate On a Single Document

Use care if your action modules modify multiple documents. As a general rule, action modules should only modify the document being processed; they should not modify any other documents without fully understanding the implications. Creating side effects by modifying other documents within a single transaction can cause triggers to fire on updates, which can (potentially) cause multiple updates to the same document to be queued in the task server. Therefore, you should only modify other documents if you fully understand the consequences (or if you are sure there are no triggers on the other updates).

If your action modules modify multiple documents, you must design your application to handle the side effects. Each time a document that is in the scope of a domain is updated, a trigger fires, which can initiate a new set of processing. If your application must do this, make sure you carefully think through the side effects that will occur.

5.4.5 Action Modules Must Be a Single Transaction

An action module must execute as a single transaction; it should not update the document more than once. Avoid using `xdmp:eval` or `xdmp:invoke` to run other transactions from within an action module. The Content Processing Framework assumes that action modules will perform a single transaction.

If you do perform multiple update transactions in an action module, you should understand the implications. Transitions are initiated as transactions complete. Having multiple transactions complete in a single action module can lead to multiple transitions being initiated in parallel, leading to unpredictable results. While it is possible to do this, your application design must take this into account. For more details, see “Design Patterns For Content Applications” on page 50.

5.4.6 Using XSLT Stylesheets Instead of Action Modules

If you specify a path to an XSLT stylesheet instead of a path to an XQuery module in a pipeline, then the framework will invoke the stylesheet with the appropriate variables. For an action module, the stylesheet must return the new contents of the document. The framework takes care of invoking the stylesheet for you, and it takes care of the other mechanics (the `try/catch` and calling `cpf:success` or `cpf:failure`); your stylesheet should not directly perform any updates, as the framework will take care of that for you. Note that this is different from XQuery actions, which actually perform the update. If your stylesheet produces multiple result documents in an action module, the first result document is the content that updates the document under CPF control. Any subsequent result documents in the stylesheet output are saved to the database at the URIs specified in the stylesheet.

6.0 Using the Framework to Create Custom Applications

This chapter describes ways to use the MarkLogic Server Content Processing Framework to create custom applications that run your own modules. It includes the following sections:

- Install Content Processing Framework in Database
- Decide on the Stages and Logic for Your Processing
- Create and Load Your Pipelines
- Create and Load Your Modules
- Design Patterns For Content Applications
- Microsoft Office 2007 Documents
- Other CPF Applications Included With MarkLogic Server

6.1 Install Content Processing Framework in Database

You must install the Content Processing Framework in each database that contains documents you want to process. This section describes the procedure for installing the Status Change Handling Pipeline in a database, which enables you to create applications using the Content Processing Framework.

> **Note:** If you are using the Default Conversion Option, follow the installation steps in "The Default Conversion Option" on page 62 instead of the steps in this section. The installation process for the Default Conversion Option includes the installation of the Status Change Handling Pipeline, so the following procedure is not needed.

Perform the following steps to install the Content Processing Framework in a database:

1. If it is not already installed, install MarkLogic Server.

2. Open the Admin Interface to the database page for the database in which you want to install the Content Processing Framework.

3. On the database configuration page, select a `triggers database` to use with your database (for example, *Triggers*). You can use any database for the triggers database. It can be the same database as the one you are configuring (for example, you can set the *Documents* database as the triggers database for the *Documents* database) or it can be a different database (for example, the *Triggers* database created as part of the installation process).

4. Click OK to apply the changes to the database configuration.

5. Click the Content Processing link under the database to which you want to install the Content Processing Framework. The Content Processing Summary page appears.

6. On the Content Processing Summary page, click the Install tab. The Content Processing Installation page appears.

7. On the Content Processing Installation page, select `false` for the `enable conversion` option and click Install.

   Note: If you have purchased the Default Conversion Option and select `true` for the `enable conversion` option, the default conversion pipelines and domain will be installed (in addition to the Content Processing Framework). If you have not purchased the Default Conversion Option, the `enable conversion` option will be greyed out.

8. Click OK to confirm the installation of content processing in your database.

9. When the installation is complete, the Content Processing Summary page appears. It should show content processing installed in your database.

10. On Content Processing Summary page, click the default domain for your database (for example, *Default Documents* if you chose the *Documents* database).

11. On the Domain Configuration Page, modify any of the default values as needed to partition the data to which your processing will occur. The default values set the document scope to include any documents whose URI begins with a `/` and to use modules that are in the database defined as the modules database whose URI begins with a `/`. For details on setting the domain values, see "Creating and Modifying Domains" on page 29.

12. Attach any relavant pipelines to the domain, as described in "Attaching and Detaching Pipelines to Domains" on page 30.

The Content Processing Framework is now installed for the database.

6.2 Decide on the Stages and Logic for Your Processing

The Content Processing Framework makes it easy to set up content processing applications that have multiple processing stages. The number of stages you need depends on the processing you need to do. Also, the number of passes you need to make through a document might contribute to the number of stages needed in your application because each stage can only result in a single update transaction to the document (see "Action Modules Must Be a Single Transaction" on page 47).

Pipeline actions are typically called based on a condition, so you need to decide on the logic for your conditions as well as your actions.

6.3 Create and Load Your Pipelines

You must create and load any pipelines to define your content processing. Pipelines are XML documents that describe the conditions, actions, and states for your content processing application. For details on how to create and load pipelines, see “Understanding and Using Pipelines” on page 31.

6.4 Create and Load Your Modules

You must develop XQuery modules for any conditions or actions referenced in your pipelines. The XQuery modules are where the work of your content processing occurs. For the rules about condition and action modules, see “Developing Modules to Process Content” on page 43.

You should store your modules in the modules database and document root configured in the `evaluation context` for the domain to which the pipeline is attached.

6.5 Design Patterns For Content Applications

The Content Processing Framework uses post-commit triggers to move content processing from one stage to another. Consequently, applications with complex sets of pipelines process certain things asynchronously. When you are designing applications with the Content Processing Framework, it is important to think through the consequences of this.

The following are example scenarios which can cause asynchronous processing to occur in applications:

- Suppose you update documents B and C while you are processing document A. Because you cannot guarantee which update will finish first, and which trigger action will execute first, document B might end up being processed before document C, or it might end up being processed after document C.
- Suppose documents B and C are both processing, and each process ends up changing a property on document A to a different value. After both transactions complete, the value of the property on document A will depend on whether document B or document C committed its processing first. Also, each update to document A will trigger an update action on document A.
- Suppose a delete of document A triggers an action to occur which cleans up links to document A. Before the action transaction occurs, however, suppose document A is created again, triggering a different action. In this case, it would be a good idea for the delete action to check and make sure the document still does not exist before cleaning up links.

These scenarios are not necessarily bad, but they can cause unexpected behavior if you do not properly understand them at application design time. Your applications either need to avoid these types of scenarios or they need to be designed to handle them.

6.6 Microsoft Office 2007 Documents

Microsoft Office 2007 uses a zip format to package up documents, and inside the zip file is XML content. MarkLogic Server includes the ability to zip and unzip documents directly from XQuery. The zip APIs are `xdmp:zip-create`, `xdmp:zip-get`, and `xdmp:zip-manifest`. You can use these APIs to write applications that use Word 2007 (or other Office 2007) content in a MarkLogic Server database.

There is a pipeline installed with CPF called the `Office OpenXML Extract` pipeline. This pipeline unzips and extracts Word 2007 documents (with a `.docx` file extension) and then saves the extracted XML documents in the database.

There is another pipeline called `WordprocessingML Process` which takes the `document.xml` part of the extracted Word 2007 documents and does some processing on it to make it more searchable. The `document.xml` part of the extracted Word 2007 document sometimes breaks words into multiple elements, and this pipeline and its associated actions put the broken words back together, which makes them easier to search.

You can create custom pipelines to process other Office 2007 documents that perform similar actions to the other OpenXML pipelines. Because Office 2007 documents are already XML documents, you can do many things with them.

6.7 Other CPF Applications Included With MarkLogic Server

In addition to the conversion application (see "The Default Conversion Option" on page 62), there are several other CPF applications that ship with MarkLogic Server. The following are the pipelines for these applications:

- `Alerting` (for alerting applications, as described in Creating Alerting Applications in the *Search Developer's Guide*)
- `XInclude Processing` (for modular documents, as described in Reusing Content With Modular Document Applications in the *Application Developer's Guide*)
- `Entity Enrichment` (for finding entities in a document and enriching the XML with markup, as described in Marking Up Documents With Entity Enrichment in the *Search Developer's Guide*)

These applications are all designed to be used together, if you desire. To use these applications together, simply attach any of these the pipelines that you want to run (along with the `Status Change Handling` pipeline) to a domain. They will execute in the following order: conversion, entity enrichment, modular documents, and finally alerting.

Additionally, there are some sample applications that use CPF. The sample applications are for demonstration purposes only, and are not designed to be put into production; see the `samples-license.txt` file in the `<marklogic-dir>/Samples` directory for more information.

7.0 Security Considerations With Content Processing

This chapter describes security considerations to be aware of when using the MarkLogic Server Content Processing Framework, and includes the following sections:

- Security Requirements for Users Who Create or Modify Documents
- Security Requirements When Modules Perform Privileged Operations
- Security Roles for Managing Content Processing

7.1 Security Requirements for Users Who Create or Modify Documents

When a document is loaded into a database, the user who loads the document must possess the appropriate privileges to create and/or modify the document. This is true in any application, including a content processing application.

For example, if a content processing application has users who add documents to a database via a WebDAV client, then the user who accesses the WebDAV client must be granted a role with the necessary permissions and privileges to create documents in the context defined by the WebDAV configuration. For details on roles, permissions, and privileges, see *Understanding and Using Security Guide*.

7.2 Security Requirements When Modules Perform Privileged Operations

Any modules that are invoked by content processing applications will evaluate as the user who made the change to the document. If that user does not have the privilege to perform the operations in the invoked XQuery module, the module transaction will fail.

Certain MarkLogic Server operations require privileges to execute. For example, `xdmp:email`, `xdmp:eval`, `xdmp:eval-in`, `xdmp:invoke`, and `xdmp:invoke-in` all require that users possess their corresponding execute privileges. Therefore, if your modules perform any privileged operations, you either need to deal with the privileges in the XQuery code (with an amp, for example), grant your users a role with the required privileges, or handle the exceptions for unprivileged users in your XQuery code. The pipeline-execution role, predefined in the server, is used (with an amp) to allow users to run the `xdmp:eval` and `xdmp:invoke` functions in certain contexts in the content processing code.

7.3 Security Roles for Managing Content Processing

MarkLogic Server includes the following pre-defined roles for managing content processing applications:

- pipeline-execution

  Used in the XQuery code to allow any user (who can write a document to the domain) to execute code in the pipeline.

- pipeline-management

  Required to create, modify, or delete pipelines.

- domain-management

  Required to create, modify, or delete domains.

8.0 Debugging and Recovering from Error Conditions

This chapter describes application debugging and server trace events in the MarkLogic Server Content Processing Framework, and includes the following sections:

- Database Online Events
- Disabling Content Processing Triggers
- Content Processing Framework Trace Events
- Examining the Host and Task Server Status Pages For Tasks in the Queue
- Find Errors in the ErrorLog.txt Log File
- Examining Properties Documents
- Find Documents in the Error State

8.1 Database Online Events

The Content Processing Framework includes a mechanism for continuing processing in the event of a database becoming unavailable (from MarkLogic Server becoming unavailable, for example). When a database becomes available again, the Content Processing Framework catches the event and resumes processing where it left off. For example, if a pipeline defined five phases of processing and the database became unavailable during the processing, some documents might have completed their processing, some might be on phase two of processing, some might be on phase three, and so on. Because the state is stored in the properties document corresponding to each document, when the database starts back up, each document will continue from where it left off. This is the reason why you must call `cpf:success` and `cpf:failure` in your action modules, as describes in “Action Modules Use try/catch With cpf:success and cpf:failure” on page 45.

The database online events are part of the Status Change Pipeline, and the processing will automatically continue when the database becomes available again.

> **Note:** The database online event causes the Content Processing Framework to look for unprocessed documents in the domain scope when the database comes online (for example, when MarkLogic Server restarts). Therefore, if you set up a domain with a scope that includes existing, unprocessed documents, those documents will be processed the first time the database online event is triggered. For details on domains, see “Understanding and Using Domains” on page 26.

8.2 Disabling Content Processing Triggers

If you want to temporarily disable content processing for a database, you can disable the triggers for that database. You can disable any or all of them. For example, if you want to disable only the restart triggers (which will make it so nothing happens after the database comes online, for example after a restart of the server), you can disable the `cpf:restart` trigger.

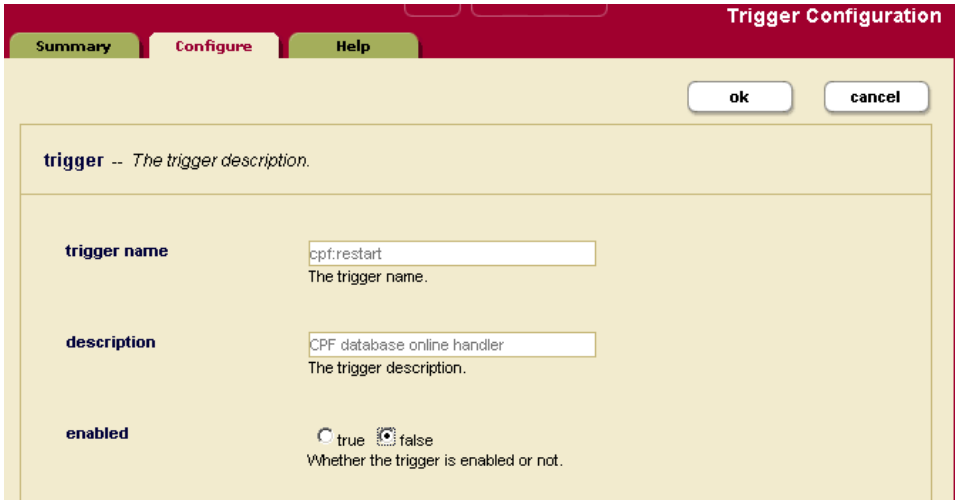
To disable content processing triggers, perform the following steps:

1. Open the Admin Interface to the database page for the database in which you want to disable content processing triggers.

2. In the Admin Interface menu, click Triggers for the database in which you want to disable content processing triggers.

3. On the Trigger Summary page, click the link corresponding to the content processing trigger you want to disable. For example, if you want to disable the `cpf:restart` trigger for the trigger with scope `/myDocuments/`, click that link.

4. Find the `enable` buttons on the Trigger Configuration page and click the `false` button.



5. Click OK.

This will disable the trigger, and will have the effect of stopping content processing for that event (in the example above, for the restart event).

To enable the trigger again (and enable content processing again for future events), go to the same Admin Interface page and select the `enable true` button.

8.3 Content Processing Framework Trace Events

There are trace events for the Content Processing Framework to help you debug your content processing applications. The trace events make it easy to see when documents are changed as a result of module actions from pipelines. This section describes the Content Processing Framework trace events and provides a procedure for how to configure them. The following sections are included:

- List of Trace Events
- Using the Server Trace Events
- Sample Scenario for Trace Events
- Creating Your Own Trace Events

8.3.1 List of Trace Events

This section lists the trace events to support debugging of content processing applications.

The following events cover the preconditions for the trigger events:

- `CPF on-create`

  This event is generated whenever the preconditions for the on-create trigger are satisfied.

- `CPF on-delete`

  This event is generated whenever the preconditions for the on-delete trigger are satisfied.

- `CPF on-update`

  This event is generated whenever the preconditions for the on-update trigger are satisfied.

- `CPF on-status-enter`

  This event is generated whenever the preconditions for the on-status-enter trigger are satisfied.

- `CPF on-state-enter`

  This event is generated whenever the preconditions for the on-state-enter trigger are satisfied.

- `CPF Condition Invoke`

  This event generates a trace for every condition CPF attempts. Note that this can event generate a lot of messages, so only use this if you need to debug your conditions.

- `CPF Condition Result`

  This event generates a trace for every result of an attempted CPF condition. Note that this event can generate a lot of messages, so only use this if you need to debug your conditions.

The preconditions include more than the conditions which cause a particular trigger to fire (although they recheck the trigger conditions as well, because there might be a lag between when the trigger fired and when the its module executed, and the triggering condition might no longer be true). For example, the `on-state-enter` trigger requires that the document also have an `active` processing status.

The following events cover action and state/status changes that occur during processing:

- `CPF Action Invoke`

  This event is generated whenever a Content Processing Framework trigger invokes a pipeline action.

- `CPF Action Complete`

  This event is generated whenever a Content Processing Framework trigger completes an invoked action.

- `CPF State Change`

  This event is generated whenever the state of a document is set (from a `cpf:document-set-state` operation).

- `CPF Status Change`

  This event is generated whenever the processing status of a document is set (from a `cpf:document-set-processing-status` operation).

- `CPF Link Change`

  This event is generated whenever a `lnk:link` property changes between documents by using the `lnk:insert`, `lnk:create`, or `lnk:remove` module functions.

- `CPF`

  This event enables all of the `CPF*` events except `CPF Condition Invoke` and `CPF Condition Result`. Note that this will generate a significant number of log messages, especially if you are processing a large number of documents.

8.3.2 Using the Server Trace Events

To use the trace events for content processing, you must enable tracing (at the group level) for your configuration and set events. Perform the following to enable and set trace events:

1. Log into the Admin Interface.

2. Select Groups > *group_name* > Diagnostics.

   The Diagnostics Configuration page appears.

3. Click the `true` button for `trace events activated`.

4. Enter the trace events (as described in "List of Trace Events" on page 56) you want to enable.

5. Click the OK button to activate the events.

After you configure the trace events, when any of the configured events occur, a line is added to the `ErrorLog.txt` file, indicating which document is involved the event.

> **Note:** The trace events are designed as development and debugging tools, and they might slow the overall performance of MarkLogic Server. Also, enabling many trace events will produce a large quantity of messages, especially if you are processing a high volume of documents. When you are not debugging, disable the trace event for maximum performance.

8.3.3 Sample Scenario for Trace Events

Suppose you are debugging a content processing application. You might enable `CPF Action Invoke` to verify that the actions you thought should take place did in fact take place. You might enable `CPF Action Complete` (which includes the elapsed time) to figure out which steps in your application are taking most of the time, so you can tune it.

But suppose you notice something is wrong; the application appears to skip processing for some documents. You can then enable `CPF on-state-enter` to see if the document is passing the preconditions of the trigger. Similarly, you can enable `CPF State Change` to follow the state changes defined in your pipeline.

The trace events allow you to follow the processing of your content processing application in as much detail as you need.

8.3.4 Creating Your Own Trace Events

You can add your own trace events to your code with the `xdmp:trace` function. When the `xdmp:trace` function is called and trace events are enabled, a message is logged to the `ErrorLog.txt` file. For the syntax of xdmp:trace, see the *MarkLogic XQuery and XSLT Function Reference*.

8.4 Examining the Host and Task Server Status Pages For Tasks in the Queue

The Host Status page in the Admin Interface shows information for any tasks that are in the task server queue for that host. The Task Server Status page also shows information about tasks in the task server queue. Tasks are added to the task queue during content processing, and you can use the Host Status page to monitor how many tasks are in the queue.

To view the Host Status page in the Admin Interface, click the Hosts menu item, then click the name of the host in which the content processing application is running. Then click the Status tab to view the Host Status page. The Task Server status appears in the second table, about two-thirds of the way down the page. The following screen shot shows the task Server portion of the Host status page:

Server	Current Tasks	Tasks Queued	Queue Size	Ratio	Task Rate	Oldest Task	Deepest Task
Task Server	14	1,458	10,000	14.6%	24.17	9.6 s	8

The following table shows the meaning of the fields in the Task Server portion of the Host Status page:

Task Server Status Field	Description
Current Tasks	The number of tasks currently being evaluated.
Tasks Queued	The number of tasks waiting to be evaluated.
Queue Size	The maximum size of the task queue. This limit is configurable on the Task Server Configuration page in the Admin Interface.
Ratio	The ratio of the size of the queue to the number of tasks in the queue.
Task Rate	A moving average that is the approximate number of tasks being executed per second.
Oldest Task	The longest time that a currently evaluating task has been running.
Deepest Task	The largest task depth of the currently evaluating tasks. The depth is determined based on a task that is spawned by another task that is in turn spawned by another task, and so on.

The Task Server Status page (Groups > *group_name* > Task Server > Status tab) also shows information about tasks that are currently running in the task server.

8.5 Find Errors in the ErrorLog.txt Log File

Errors that occur in content processing are logged to the server log file. Examine any errors in the `Logs/ErrorLog.txt` file.

8.6 Examining Properties Documents

The Content Processing Framework stores information about content processing in the properties document corresponding to the URI for each document. For details about properties documents, see the "Properties Documents and Directories" chapter in the *Application Developer's Guide*.

The following is a sample properties document for a document that has completed content processing:

```
<prop:properties>
  <cpf:processing-status>done</cpf:processing-status>
  <cpf:last-updated>2005-03-16T16:56:09.466262-08:00
    </cpf:last-updated>
  <cpf:state>http://marklogic.com/states/final</cpf:state>
  <lnk:link from="http://myDomainScope/myDocument_doc.xhtml"
     to="http://myDomainScope/myDocument.doc" rel="source"
     rev="conversion" strength="strong"/>
  <lnk:link from="http://myDomainScope/myDocument_doc_parts/css.xml"
    to="http://myDomainScope/myDocument.doc" rel="source"
    rev="stylesheet" strength="strong"/>
  <prop:last-modified>2005-03-16T18:34:40.71377-08:00
    </prop:last-modified>
</prop:properties>
```

Note: If a document fails a processing step (when `cpf:failure` is called), the error that caused the failure is stored in the properties document.

8.7 Find Documents in the Error State

When a document fails to complete a pipeline or enters some other error condition, the Content Processing Framework places the document in an error state. Because the states are stored as properties, you can easily query for documents in the error state. The following query finds all documents that are in the error state:

```
declare namespace cpf="http://marklogic.com/cpf"
declare namespace prop="http://marklogic.com/xdmp/property"

<errorReport>
{
(: set $dir to the document scope for your domain :)
let $dir := "http://myDomainScope/"
let $all :=
  for $x in xdmp:directory($dir)
  (: only find the documents in the error state :)
  where xdmp:document-properties(xdmp:node-uri($x))//cpf:state/text()
      eq "http://marklogic.com/states/error"
  return
  (: return the document uri and the properties document :)
  <errorState>{
    (<uri>{xdmp:node-uri($x)}</uri> ,
    xdmp:document-properties(xdmp:node-uri($x))/*)
  }</errorState>
return
(: count the number of documents in the error state :)
  (<countOfErrorStateDocuments>{count($all/prop:properties)
  }</countOfErrorStateDocuments>
,
$all)
}
</errorReport>
```

Note: This sample query works for the states defined in the Default Conversion Option. If you want to search only for `cpf:error` properties, you can write a query using the following expression:

```
declare namespace cpf="http://marklogic.com/cpf"

xdmp:document-properties()//cpf:error
```

9.0 The Default Conversion Option

This chapter describes the Default Conversion Option, which is designed to convert Microsoft Office, Adobe PDF, and HTML files to XHTML and DocBook. It includes the following sections:

- Installing the Conversion Pipelines and Framework
- Simple Drag-and-Drop Conversion
- What the Conversion Pipeline Generates
- Understanding and Using the Default Conversion Option
- Modifying the Default Conversion Option

9.1 Installing the Conversion Pipelines and Framework

The Default Conversion Option installation installs the Content Processing Framework for your database, sets up the domain for the pipeline, loads the needed triggers into the triggers database, and performs other pipeline initialization tasks. You need to install the Default Conversion Option for each database in which you plan on using conversion.

Complete the following steps to install the Default Conversion Option into a database.

1. If it is not already installed, install MarkLogic Server.

2. Open the Admin Interface to the database page for the database in which you want to install the Default Conversion Option. For example, if you want to install the pipeline into the *Documents* database, open the database page for the *Documents* database.

   Note: MarkLogic recommends creating a new database to use when testing the Default Conversion Option.

3. On the database configuration page, select a `triggers database` to use with your database (for example, *Triggers*). You can use any database for the triggers database. It can be the same database as the one you are configuring (for example, you can set the *Documents* database as the triggers database for the *Documents* database) or it can be a different database (for example, the *Triggers* database created as part of the installation process).

4. Click OK to apply the changes to the database configuration.

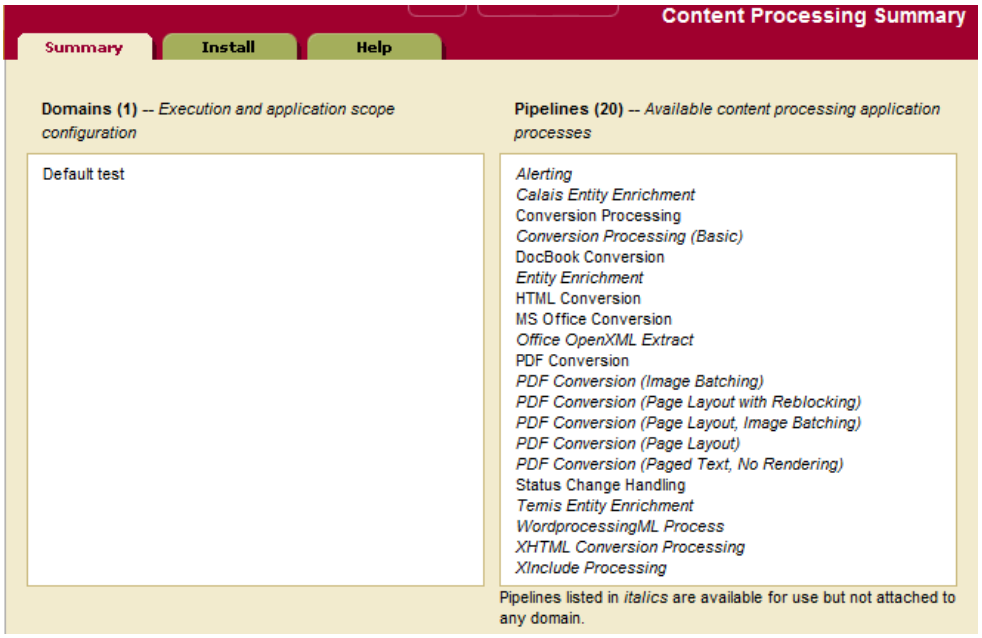
5. In the left tree menu, click the Content Processing link under the database to which you want to install the Default Conversion Option. The Content Processing Summary page appears.

6. On the Content Processing Summary page, click the Install tab. The Content Processing Installation page appears.

7. On the Content Processing Installation page, select `true` for the `enable conversion` option and click Install. Make sure `enable conversion` is set to `true`. If this is set to `false`, then you will only install the Content Processing Framework, not the Default Conversion Option.

8. Click OK to confirm the installation of content processing in your database.

9. When the installation is complete, the Content Processing Summary page appears. It should show content processing installed in your database.



The Default Conversion Option is now installed for the database. The default domain determines which documents are processed, and by default it has a document scope that applies to any document in the database with a URI starting with a slash (`/`).You can modify the domain settings if you want the Default Conversion Option to apply to a different set of documents. To modify the domain settings, click the default domain for your database (for example, *Default Documents* if you chose the *Documents* database) on the Content Processing Summary pages and make the needed modifications. For details on domains, see “Understanding and Using Domains” on page 26.

9.2 Simple Drag-and-Drop Conversion

To try out the pipeline, you need to load some Adobe PDF, Microsoft Office, and/or HTML files into the database. You can load the documents using any method you like. This section describes an easy way to load documents using a WebDAV server and client. You can then use this configuration to test document conversion with the Default Conversion Option. For more information on WebDAV servers, see "WebDAV Servers" in the *Administrator's Guide*.

Complete the following steps to load and process documents in a database.

1. Create a WebDAV server with root `/` that accesses the database in which you installed the Content Processing Pipeline.

   a. In the Admin Interface, go to the Groups > Default > AppServers page.

   b. Click the Create WebDAV tab.

   c. Enter a server name (for example, `CPF`).

   d. Enter `/` for the root.

   e. Enter a port number (for example, `9999`).

   f. Select the database in which you installed the content processing pipeline (for example, `Documents`).

   g. Click OK.

2. If you will not be logging into the WebDAV client as a privileged user, set up the needed security requirements for your WebDAV root directory and your WebDAV user. For a sample of how to set this up, see "Set the Needed Permissions on the Root Directory" on page 68.

3. Create a WebDAV client that accesses the WebDAV server you just created. For example, the following procedure applies to Windows XP; other versions of Windows or other WebDAV clients have slightly different procedures:

   a. Double-click My Network Places from your desktop.

   b. Double-click Add Network Place.

   c. For the location of the network place, enter the address with the hostname and port number of the WebDAV server you created. For example, if your server is on port 9999 of the local machine, enter the following:

```
http://localhost:9999
```

   d. Click the Next button.

   e. If prompted, enter a username and password for your WebDAV server.

   f. Enter a name for your WebDAV folder (for example, `conversion`).

   g. Click Finish.

   h. If prompted, enter the username and password for your WebDAV server.

4. Drag-and-drop Microsoft Word, Excel, Powerpoint, and/or Adobe PDF files into the WebDAV folder. This loads the documents in the database.

5. After some time has passed, refresh the WebDAV folder (for example, View > Refresh). The amount of time it takes to convert depends on the number, size, and the complexity of the documents being converted. For simple and small documents, it will take just a few seconds. For larger documents, it might take significantly longer.

The converted documents, as well as the original documents and any parts generated as part of the conversion, will appear in the WebDAV folder. If you have large documents or if you load many documents into the database, the processing might continue for several minutes or longer.

9.3 What the Conversion Pipeline Generates

After the conversion process is finished, for each HTML, Word, Excel, Powerpoint, and PDF document you loaded, the Default Conversion Option produces the following:

- The original document
- An XHTML document (`*.xhtml`)
- A simplified DocBook XML document (*`.xml`)
- A directory (`*_parts`) containing various parts generated as part of the conversion process. The parts are typically any images that were in the original document, a cascading style sheet document (`conv.css`), and a document containing an analysis of the stylesheet (`css.xml`). PDF documents also include toc.xml, which is an analysis of the table of contents structure.

The generated XHTML and XML documents have a URI that includes the suffix of the original document. For example, a document called `word.doc` produces `word_doc.xml` and `word_doc.xhtml`.

9.4 Understanding and Using the Default Conversion Option

The Default Conversion Option uses the components of Content Processing Framework, as well as converters to create XML documents from Microsoft Office and Adobe PDF files, to create a unified conversion process which converts Microsoft Office, Adobe PDF, and HTML files to well-structured XHTML and simplified DocBook format XML documents. This section provides some background on how the default conversion process works, and includes the following sections:

- Components of the Default Conversion Option
- Steps in the Conversion Process
- Default Conversion Option States
- Errors, Troubleshooting, Debugging, and Recovery

9.4.1 Components of the Default Conversion Option

The Default Conversion Option includes the following components:

- Status Change Handling Pipeline
- Microsoft Office Pipeline
- PDF Pipelines
- HTML Pipeline
- Supporting XQuery modules
- Microsoft Office XML converter
- Adobe PDF XML converter
- The `xdmp:tidy` function built into MarkLogic Server

There are also supporting XQuery modules for the Default Conversion Option for the following:

- Generic Conversion
- PDF Conversion
- DocBook Conversion
- CSS Conversion
- XHTML Conversion
- Microsoft Office Conversion

These XQuery modules include the XQuery source code, so you can analyze them and use their functions in your own applications. The XQuery modules are installed into the following directory:

```
<install_dir>/Modules/MarkLogic/conversion
```

For details on these functions, see the *MarkLogic XQuery and XSLT Function Reference*.

9.4.2 Steps in the Conversion Process

The steps in the conversion process differ for the different document formats (Microsoft Office, Adobe PDF, and HTML). The steps are defined in the following pipelines:

- `html-pipeline.xml`
- `msoffice-pipeline.xml`
- `pdf-pipeline.xml`
- `pipeline.xml`

Generally, the conversion process perform the following tasks:

- Check to see what kind of document it is.
- Convert the document to XHTML based on its type.
- Cleans up the converted XHTML.
- Extract the style information into a CSS document.
- Transform the XHTML to infer the table of contents structure for the document.
- Transform the XHTML to create a simplified DocBook structured format for the document.

9.4.3 Default Conversion Option States

The conversion states are defined in the pipelines and are stored in the properties document for each document. The conversion process includes the following states:

- http://marklogic.com/states/initial
- http://marklogic.com/states/updated
- http://marklogic.com/states/xhtml
- http://marklogic.com/states/cleaned-xhtml
- http://marklogic.com/states/structured-xhtml
- http://marklogic.com/states/enhanced-xhtml
- http://marklogic.com/states/pdf-xhtml
- http://marklogic.com/states/analyzed-styles
- http://marklogic.com/states/final

9.4.4 Errors, Troubleshooting, Debugging, and Recovery

This section describes the following error and troubleshooting situations you might encounter with the Default Conversion Option:

- Microsoft Word 95 and Other Microsoft Office Errors
- Set the Needed Permissions on the Root Directory
- Default or Inherited Collections and Permissions
- Enable Debugging Capabilities
- Create Your Own Error Handling Pipeline

9.4.4.1 Microsoft Word 95 and Other Microsoft Office Errors

The Default Conversion Option only converts Microsoft Office 97 and newer documents; it cannot convert Microsoft Office 95 and earlier documents. If you try to convert Microsoft Word 95 or older documents (or other Microsoft Office 95 documents), the conversion will fail, putting the document in the `http://marklogic.com/states/error` state. If this happens, you can do the following:

- Find all of the documents that are in the error state (For details, see "Find Documents in the Error State" on page 61).
- Open the documents in a newer version of Microsoft Word and then re-save them (as newer Word documents, not Word 95 or older documents).
- Reload the saved files into the database.

Once you reload the documents into a database with content processing installed and configured, the new documents will be converted.

There are other types of errors you might get with Microsoft Office documents. For example, if a document is password protected, the conversion will fail because it needs the password to open the document. In general, you can address these types of issues by opening the document in the appropriate Microsoft Office application, changing the cause of the error (for example, removing the password protection), re-saving the document, and reloading the document into the conversion domain.

9.4.4.2 Set the Needed Permissions on the Root Directory

When you add documents to the database for conversion, the user who adds the documents must have the needed permissions to add and modify documents. If you are using WebDAV server to drag-and-drop documents into the database, the root directory of the WebDAV server must also have the needed permissions.

One simple way to accomplish these security requirements is to do the following:

- Create a URI privilege for the URI that is configured as the root directory of your WebDAV server.
- Create a role that has the URI privilege and has default permissions of read. insert, and update for the role.

- Set the permissions on the WebDAV root directory for the role you created. For example, if the role you created is named `webdav`, and the root directory has the URI `/webdav/root/`, run a query (as a privileged user) similar to the following:

  ```
  xdmp:document-set-permissions("/webdav/root/",
    ( xdmp:permission("webdav", "read"),
      xdmp:permission("webdav", "insert"),
      xdmp:permission("webdav", "update") ) )
  ```

  You can check the permissions with the following query:

  ```
  xdmp:document-get-permissions("/webdav/root/")
  ```

- Grant the new role (`webdav` in the example above) to the user who accesses the WebDAV server.

9.4.4.3 Default or Inherited Collections and Permissions

If you are using a collection in the domain to specify which documents to convert, the new documents created by the conversion process must be created as part of the collection specified in the domain. You can do this in the following ways:

- Set the `inherit collections` option at the database level to `true` and make sure the parent directory belongs to the collection.
- The user who runs the Default Conversion Option (that is, the user who originally creates the documents to be converted, whether by drag and dropping into a WebDAV folder or by some other means) can have the collection specified as a default collection (or a role to which the user is assigned).
- You can explicitly set the collection on a document (for example, in your XQuery module code or through XDBC).

Otherwise only the first phase of conversion will occur (because documents created during the conversion process will not be part of the collection specified in the domain). Similarly, you must have either the appropriate default permissions assigned to the user (or a role to which the user is assigned) or you should set the permissions to inherit at the database level.

For information on inherited collections and inherited permissions, see the *Administrator's Guide*. For information on permissions, see *Understanding and Using Security Guide*.

9.4.4.4 Enable Debugging Capabilities

If you need debugging capabilities, you can set trace events on the server for the Content Processing Framework. For details, see "Debugging and Recovering from Error Conditions" on page 54.

9.4.4.5 Create Your Own Error Handling Pipeline

If you have special error handling needs, you can always extend the Default Conversion Option application by adding your own custom error handling pipeline. For details on pipelines and creating custom code, see “Understanding and Using Pipelines” on page 31 and “Using the Framework to Create Custom Applications” on page 48.

9.5 Modifying the Default Conversion Option

This section describes ways to modify the Default Conversion Option, and includes the following subsections:

- Copy Defaults and Modify
- PDF Alternate Pipelines
- Modifying the Options for Default Conversion

9.5.1 Copy Defaults and Modify

All of the XQuery code and all of the pipelines for the Default Conversion Option are installed with MarkLogic Server. The pipelines are installed in the following directory:

```
<install_dir>/Installer
```

The XQuery modules are installed under the Modules directory in the following location:

```
<install_dir>/Modules/MarkLogic/conversion/actions
```

You can create your own pipelines by copying and modifying the Default Conversion Option code to suit your needs. Make sure you understand domains, pipelines, the concepts of the Content Processing Framework, and the rules for XQuery modules in content processing applications before modifying the pipelines. For information on these topics, see the rest of this document.

The modification possibilities are endless. You can add phases to the pipeline to do your own processing, add email notification to your application, add entity extraction from a semantic tagging service, and so on. For information on creating custom applications, see “Using the Framework to Create Custom Applications” on page 48.

9.5.2 PDF Alternate Pipelines

There are several alternate PDF pipelines available to attach to a domain instead of the default PDF pipeline. The Default Conversion Option is designed to have only one PDF pipeline attached to a domain at a time; do not attach several alternate PDF pipelines to the same domain. The following table lists the PDF pipelines with a description of each (choose the one that best matches your needs).

PDF Pipeline	Description
PDF Conversion	This is the default pipeline. It is set up to extract the most out of the text in PDF files, concentrating more on structure than page layout. It is not optimized for page layout fidelity.
PDF Conversion (Page Layout)	This pipeline preserves the page layout fidelity as closely as possible. It converts the PDF file on a page-by-page basis, producing a new element for each page. This can make it difficult to render logical parts of the document together, unless they correspond to page breaks, but makes it easy to process the document one page at a time. You cannot do the docbook processing (with the DocBook Conversion pipeline) when using this pipeline.
PDF Conversion (Page Layout with Reblocking)	This pipeline attempts to preserve more exact rendering of content then the default PDF pipeline, while making it more feasible to extract and render logical subunits instead of just pages. It is possible to lose some page-layout fidelity with this pipeline compared withe the Page Layout pipeline. This pipeline requires more processing time than the others.
PDF Conversion (Page Layout, Image Batching)	This pipeline preserves the page layout fidelity as closely as possible, and also batch-processes images. It converts the PDF file on a page-by-page basis, producing a new element for each page. This can make it difficult to render logical parts of the document together, unless they correspond to page breaks, but makes it easy to process the document one page at a time. You cannot do the docbook processing (with the DocBook Conversion pipeline) when using this pipeline. This pipeline is intended to minimize the memory requirements when converting PDF documents containing many large images (for example, scanned PDF files). The overall processing time will increase, however, because of the additional image extraction steps. You can adjust the `batch-size` parameter to tune memory needs against overall throughput. This pipeline is useful for page-layout conversion of very large PDF documents with many large images.

PDF Pipeline	Description
PDF Conversion (Image Batching)	This pipeline performs the extraction of the default pipeline and also extracts images in batches, increasing the overall processing time but decreasing the memory usage. You can adjust the `batch-size` parameter to tune memory needs against overall throughput. This pipeline is useful for default conversion of very large PDF documents with many large images.
PDF Conversion (Paged Text, No Rendering)	This pipeline produces a very simple XML structure that is suitable for word searches within pages, but not rendering. Paragraph and section structure is lost. By default images are not extracted. This variant emphasizes speed of extraction and word search. To make best use of the output of this pipeline, define the elements `a` and `page` in the `http://marklogic.com/cpf/paged-text` namespace as phrase-throughs in the database configuration. You cannot do the docbook processing (with the DocBook Conversion pipeline) when using this pipeline.

9.5.3 Modifying the Options for Default Conversion

The Default Conversion Option uses the built-in functions `xdmp:excel-convert`, `xdmp:pdf-convert`, `xdmp:powerpoint-convert`, `xdmp:tidy`, and `xdmp:word-convert`. The pipelines reference various XQuery modules that call these functions. Each of these functions takes an options node to control its behavior. The options are set to somewhat generic defaults that work well with a large variety of documents. Your own documents might have some more specific needs, however, and the pipelines are designed with the ability to pass in options nodes which specify conversion options.

Each condition and action step in the MarkLogic pipelines has an `options` node. The options node is defined in a namespace with a URI corresponding to the module path invoked by that step. In these option nodes, you can enter options from any of the Document Conversion functions (`xdmp:excel-convert`, `xdmp:pdf-convert`, `xdmp:powerpoint-convert`, `xdmp:tidy`, `xdmp:word-convert`) in the namespace corresponding to that conversion function.

For example, the action for the default PDF Pipeline (`pdf-pipeline.xml`) has the following options node:

```
<action>
<module>/MarkLogic/conversion/actions/convert-pdf-action.xqy</module>
<options xmlns="/MarkLogic/conversion/actions/convert-pdf-action.xqy">
  <destination-root/>
  <wrap xmlns="xdmp:tidy">0</wrap>
  <tidy-mark xmlns="xdmp:tidy">false</tidy-mark>
  <show-warnings xmlns="xdmp:tidy">false</show-warnings>
</options>
</action>
```

In this options node, the `destination-root` is an option for this pipeline step. The `wrap` element is an option passed into the `xdmp:tidy` built-in function (which the PDF Pipeline uses to clean the generated xhtml), and the `tidy-mark` and `show-warnings` elements are also options passed into tidy.

Suppose the PDF documents you want to convert all are password protected and all have the same password. You can then add the following to the `options` node to specify a password:

```
<password xmlns="xdmp:pdf-convert">your_password</password>
```

Notice the namespace of the `password` option is `xdmp:pdf-convert`, and this option will be passed in when the pipeline processing calls `xdmp:pdf-convert`.

All of the pipelines have options nodes, and you can pass in any option to each pipeline. You can change the default options or add other options that make sense for your content. See the *MarkLogic XQuery and XSLT Function Reference* for the options for each of the Document Conversion functions. Also, the format conversion steps in a pipeline have the following options:

- `destination-root`
- `default-language`

The `destination-root` option specifies an alternate directory URI where the output of the conversion processing is saved. The `default-language` option is only used on the Microsoft Office conversion pipelines, and it specifies the value of an `xml:lang` attribute to put on the root node of the converted Office documents.

10.0 Technical Support

MarkLogic provides technical support according to the terms detailed in your Software License Agreement or End User License Agreement.

We invite you to visit our support website at http://help.marklogic.com to access information on known and fixed issues, knowledge base articles, and more. For licensed customers with an active maintenance contract, see the Support Handbook for instructions on registering support contacts and on working with the MarkLogic Technical Support team.

Complete product documentation, the latest product release downloads, and other useful information is available for all developers at http://developer.marklogic.com. For general questions, join the general discussion mailing list, open to all MarkLogic developers.

11.0 Copyright

MarkLogic Server 8.0 and supporting products.
Last updated: October 13, 2016

COPYRIGHT

Copyright © 2016 MarkLogic Corporation. All rights reserved.
This technology is protected by U.S. Patent No. 7,127,469B2, U.S. Patent No. 7,171,404B2, U.S. Patent No. 7,756,858 B2, and U.S. Patent No 7,962,474 B2, US 8,892,599, and US 8,935,267.

The MarkLogic software is protected by United States and international copyright laws, and incorporates certain third party libraries and components which are subject to the attributions, terms, conditions and disclaimers set forth below.

For all copyright notices, including third-party copyright notices, see the Combined Product Notices.