MarkLogic Server

Flexible Replication Guide

MarkLogic 9
May, 2017

Last Revised: 9.0-1, May, 2017

Copyright © 2019 MarkLogic Corporation. All rights reserved.

Table of Contents

Flexible Replication Guide

1.0 Flexible Replication in MarkLogic Server 5
- 1.1 Terms Used in this Guide 5
- 1.2 Understanding Flexible Replication 6

2.0 Flexible Replication Quick Start 11
- 2.1 Creating Master and Replica Databases 11
- 2.2 Configuring a Flexible Replication Pipeline for the Master Database 12
- 2.3 Creating a Replication App Server 15
- 2.4 Configuring Push Replication on the Master Database 17
- 2.5 Scheduling Replication Push Task 19
- 2.6 Loading Documents into the Master Database and Checking Replication 21

3.0 Configuring Replication 23
- 3.1 Replication Security 23
- 3.2 Defining Replicated Domains 25
- 3.3 Configuring Replication App Servers 28
- 3.4 Configuring Push Replication 29
- 3.5 Creating a Scheduled Replication Task 32
  - 3.5.1 Creating a Push-Local-Forests Replication Task 36
  - 3.5.2 Creating a Pull Replication Task 38
- 3.6 Configuring Pull Replication 39
- 3.7 Configuring Alerting With Flexible Replication 43
  - 3.7.1 Configuring Alerts 43
    - 3.7.1.1 Create an Alerting Domain 44
    - 3.7.1.2 Create an Alerting Action 44
    - 3.7.1.3 Associate Flexible Replication with the Alerting Configuration .. 45
    - 3.7.1.4 Associate the Target With a User 45
    - 3.7.1.5 Create an Alerting Rule 46
  - 3.7.2 Using QBFR 46
- 3.8 Backing Up, Restoring, and Clearing the Master and Replica Databases 47
  - 3.8.1 Interrupted replication 48

4.0 Filtering Replicated Documents 49
- 4.1 Creating a Filter Module 49
  - 4.1.1 Outbound Filters 49
  - 4.1.2 Inbound Filters 51

4.2 Configuring MarkLogic Server to use a Replication Filter Module 52
4.2.1 Configuring a Master Database to Use an Outbound Filter 52
4.2.2 Configuring a Replica Database to Use an Inbound Filter 53
4.3 Example Outbound Filter Modules 53
4.3.1 Adding a Collection 54
4.3.2 Changing the Document Quality 56
4.3.3 Adding Document Permissions 58
4.3.4 Adding a Forest Name 60
4.3.5 Changing the Document URI 61
4.3.6 Changing a Document Element 62
4.3.7 Prohibiting Replication on Select Documents 63
4.4 Example Inbound Filter Modules 64
4.4.1 Adding a Collection 64
4.4.2 Changing the Document URI 65
4.5 Setting Outbound Filter Options 66

5.0 Checking Replication Status 69
5.1 Special Circumstances that Impact Replication 69
5.2 Checking the Replication Status of a Master Database 69
5.3 Checking the Replication Status of a Domain 72
5.4 Checking the Replication Status of a Target 73

6.0 Technical Support 77

7.0 Copyright 79
7.0 COPYRIGHT 79

1.0 Flexible Replication in MarkLogic Server

This chapter describes Flexible Replication in MarkLogic Server in general terms, and includes the following sections:

- Terms Used in this Guide
- Understanding Flexible Replication

Note: To enable Flexible Replication, a license key that includes Flexible Replication is required. For details on purchasing a Flexible Replication license, contact your MarkLogic sales representative.

1.1 Terms Used in this Guide

The following are the definitions for the replication terms used in this guide:

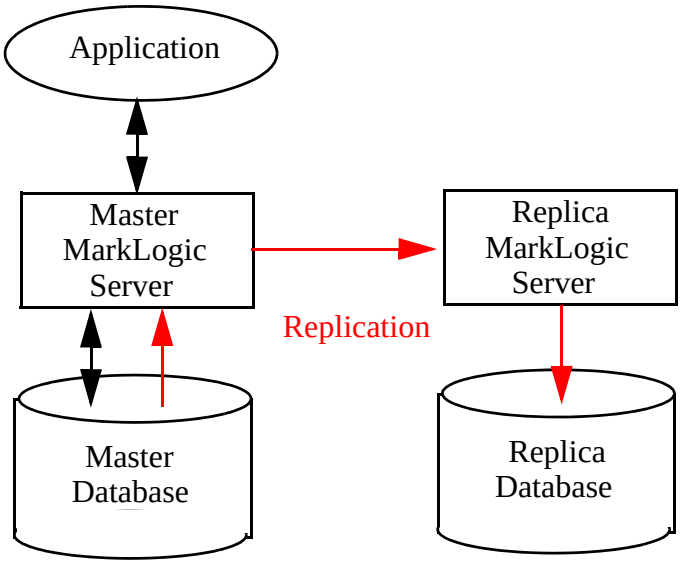
- To *Replicate* is to create a copy of a document in another database and to keep that copy in sync (possibly with some time-lag/latency) with the original.
- The *Master* is the repository that gets updated by the applications. The master, in turn, replicates the updates to other repositories, know as replicas.
- A *Replica* is a repository that receives replicated updates from the master.
- A *Master Copy* is the content being replicated. For any piece of replicated content there is a master and at least one copy.
- A *Master Cluster* is the cluster on which the replicated documents are updated by the applications.
- *Flexible Replication* is an implementation of replication based on the MarkLogic Server Content Processing Framework (CPF). Flexible replication is single-master, asynchronous, and provides a medium level of throughput and latency.
- *Replication Domain* is the specification of the set of documents to be replicated. This may be a collection or some other set definition.
- A *Filter* is an XQuery program that modifies the replicated documents in some manner, determines whether to replicate a change, or selects which parts of a document will be replicated.
- *Asynchronous Replication* refers to a configuration in which the Master does not wait for confirmation that the update has been received by the Replica before sending further updates. Flexible Replication is asynchronous.
- *Transaction-aware* refers to a configuration in which all updates that make up a transaction on the master are applied as a single transaction on the replica.
- *Zero-day Replication* refers to replicating the data in the replicated domains that existed before replication was configured.

1.2 Understanding Flexible Replication

Flexible Replication is the process of maintaining copies of data on multiple MarkLogic Servers. The purpose of replication is to make data continuously available to mission-critical applications and to enhance application performance. Some of the benefits of replication include:

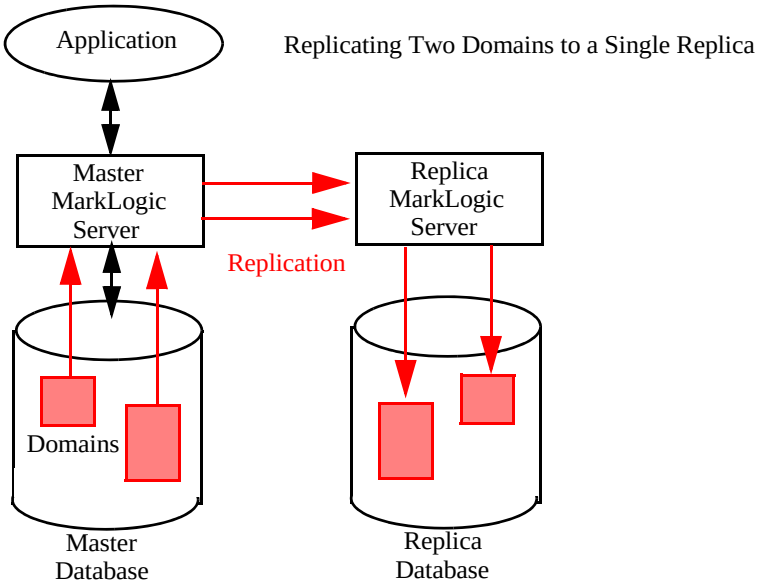
- High Availability: You can maintain duplicate data on two or more MarkLogic Servers. In the event of a software or hardware failure on one server, the data is available from another server.
- Disaster Recovery: In the event of some irreversible disaster on the Master server, duplicate data is preserved on its Replica.
- Performance: Companies with geographically dispersed clusters can use replication to maintain common data on each local cluster. Queries and updates done locally, are faster and the workload can be scaled across clusters, so that each cluster handles less of the query and update load.

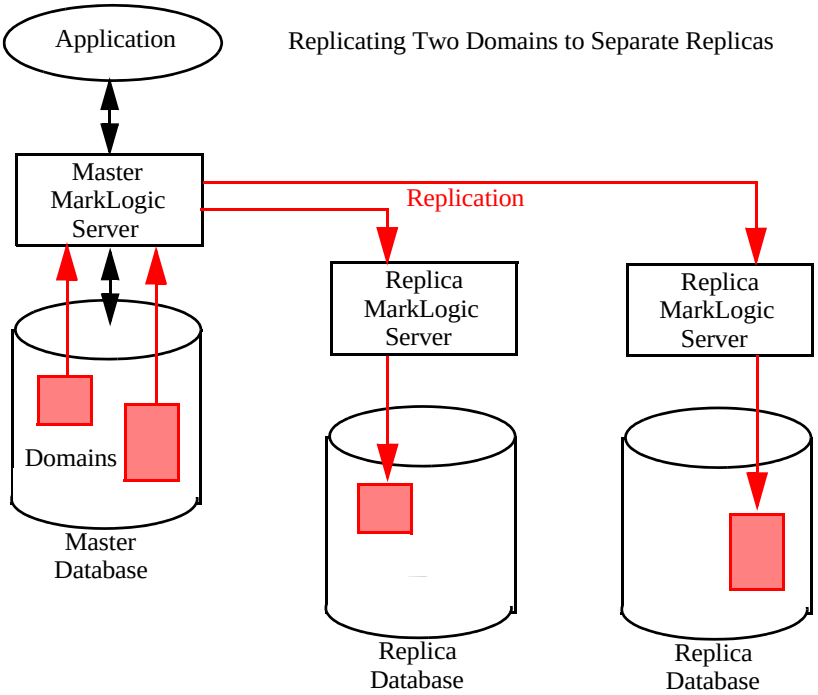
In a replicated environment, the original content is created by an application on the Master MarkLogic Server. Replication then copies the content to one or more Replica MarkLogic Servers. The Master and Replica servers are typically in different clusters, which may be in the same location or in different locations.



Flexible Replication is asynchronous, which means that the Master does not wait for confirmation that the update has been received by the Replica before sending further updates. Replication from the Master to the Replica occurs as soon as possible after the document is added or updated by the application.

MarkLogic Server uses the Content Processing Framework (CPF) as the underlying replication mechanism. The documents to be replicated are defined by a CPF domain. The scope of a domain may be a document, a collection of documents, or a directory. For more details about domains, see Understanding and Using Domains in the *Content Processing Framework Guide* Guide. You can replicate multiple domains either to the same Replica or to different Replicas, as shown is the illustrations below.

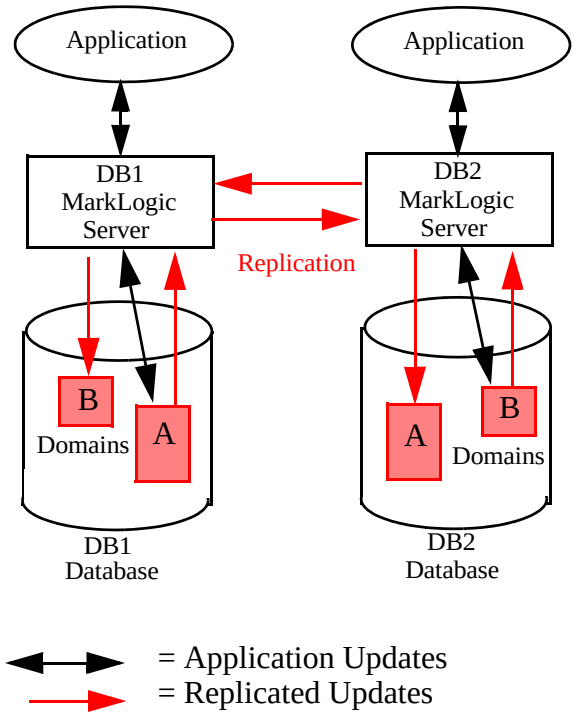




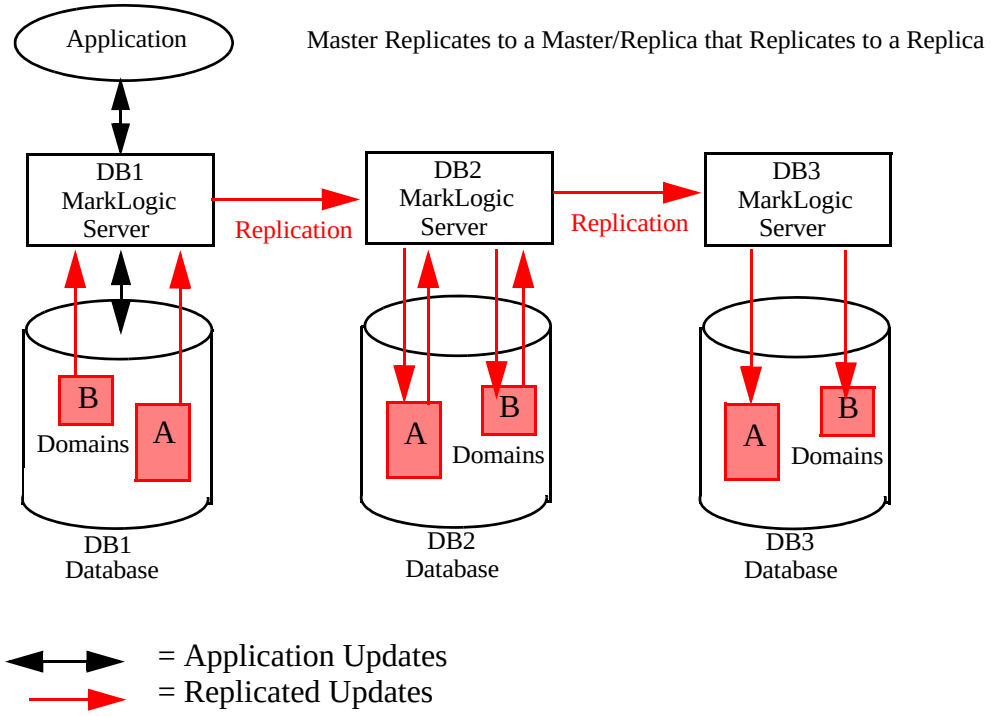
Replicated databases do not necessarily need to be configured as entirely a Master or a Replica in the replication scheme. For example, you may have two databases, DB1 and DB2, where DB1 replicates updates to the documents in Domain A to DB2 and DB2 replicates updates to the documents in Domain B to DB1.

Note: This is not a multi-master replication configuration, as the documents updated by each application must be in different domains. Any overlap between the replicated domains may result in unpredictable behavior.

Replicating Different Domains Between Databases



Another possible Master/Replica configuration is where a Master replicates updates to a Replica/Master that replicates the updates to another Replica. For example, you may have three databases, DB1, DB2, and DB3, where DB1 replicates updates to DB2 and DB2 replicates updates to DB3.



You can set up filters that narrow the scope of what documents and what parts of the documents are replicated within a domain. For example, you can set up a filter to replicate only XML documents, or you can create filters to only replicate inserts and updates (not deletes), or only replicate a particular node or element within each document.

Replication can be configured to either *push* or *pull* updates from the Master to the Replica. Push replication means that the Master pushes updates to the Replica. Pull replication means the Replica pulls updates from the Master. Push replication is triggered whenever an update is made on the Master database. Pull replication can only be configured as a scheduled task, as described in “Configuring Pull Replication” on page 39. Typically, you should use push replication, unless the Master and Replica are separated by a firewall through which a Replica server can only pull content from a Master server outside the firewall.

2.0 Flexible Replication Quick Start

This chapter provides the quick-start procedures for creating a simple flexible replication configuration on a single MarkLogic Server. Typical flexible replication configurations will have the Master and Replica databases on different MarkLogic Servers located on different hosts. Replicated MarkLogic Servers may reside in the same cluster or in different clusters. In order to keep the hardware requirements to a minimum, this quick start describes how to set up replication between two databases on the same MarkLogic Server.

This chapter includes the following sections:

- Creating Master and Replica Databases
- Configuring a Flexible Replication Pipeline for the Master Database
- Creating a Replication App Server
- Configuring Push Replication on the Master Database
- Scheduling Replication Push Task
- Loading Documents into the Master Database and Checking Replication

All of the procedures described in this chapter are done using the Admin Interface described in the Administrative Interface chapter in the *Administrator's Guide*.

2.1 Creating Master and Replica Databases

Before attempting to configure replication, create a Master and a Replica database. In these examples, the databases are named Master and Replica. However, you can use any name you wish for either the master or its replica.

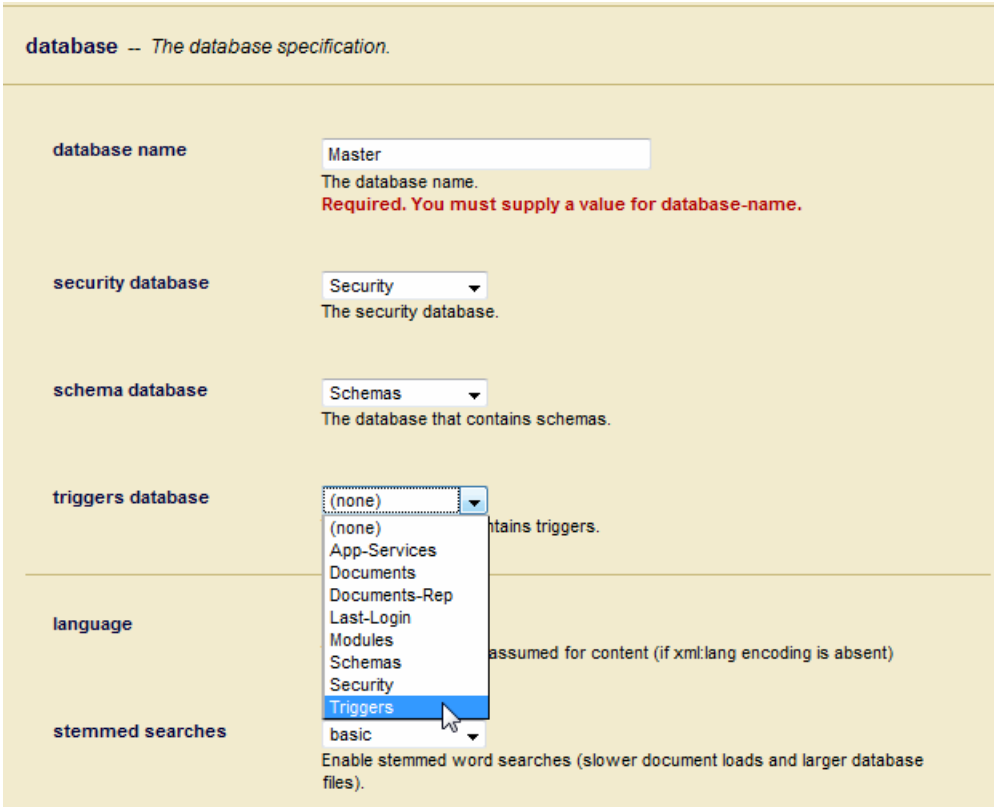
1. Create two forests, one for the Master database and one for the Replica database. For details on creating forests, see Creating a Forest in the *Administrator's Guide*.

2. Create two databases, Master and Replica, and attach the respective forest to each database. For details on creating databases and attaching forests, see Creating a New Database and Attaching and/or Detaching Forests to/from a Database in the *Administrator's Guide*.

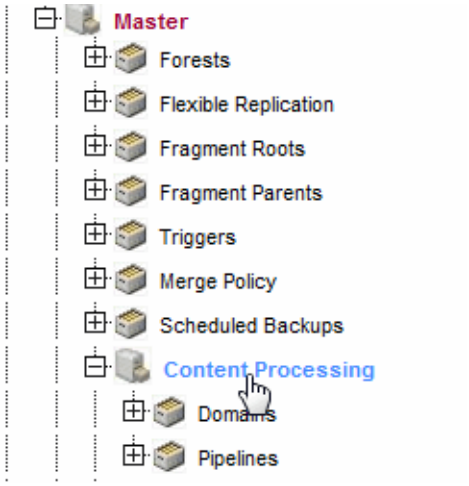
2.2 Configuring a Flexible Replication Pipeline for the Master Database

This section describes how to configure the Content Processing Framework (CPF) for the replicated domains on the Master database. In order to replicate data, CPF on the Master domains must be configured with the Status Change Handling and Flexible Replication pipelines. In this example, we use the existing Triggers database. However, if you are replicating more than one database, you should create a separate CPF database for each master database.

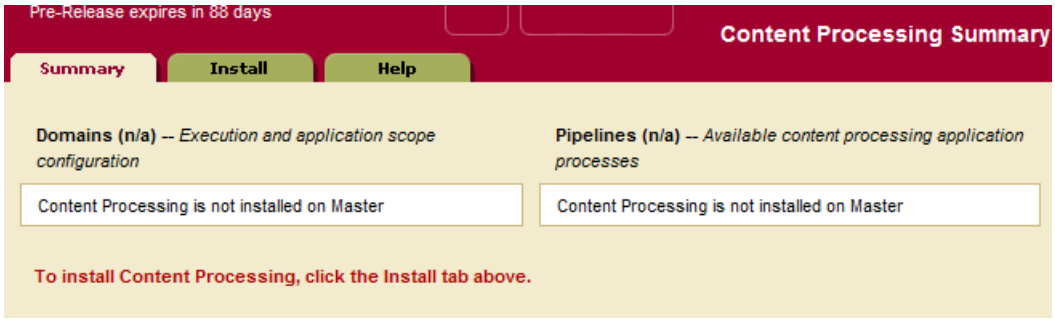
1. Configure the Master database to use a triggers database.



2. Select Content Processing on the Master database:



3. In the Content Processing Summary, click on the Install tab:

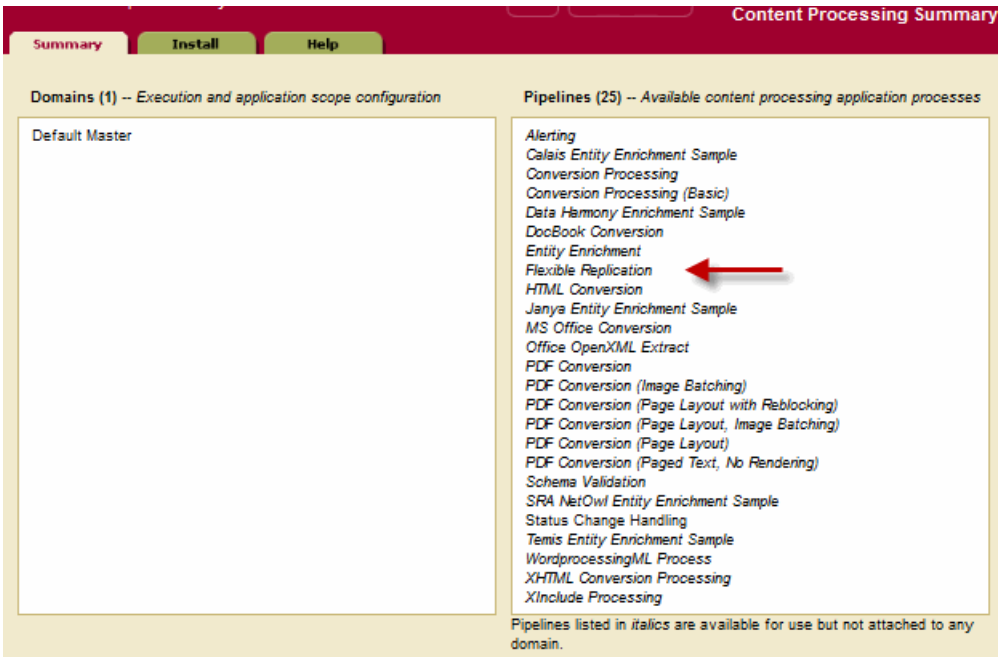


4. On the Content Processing Installation page, click Install:

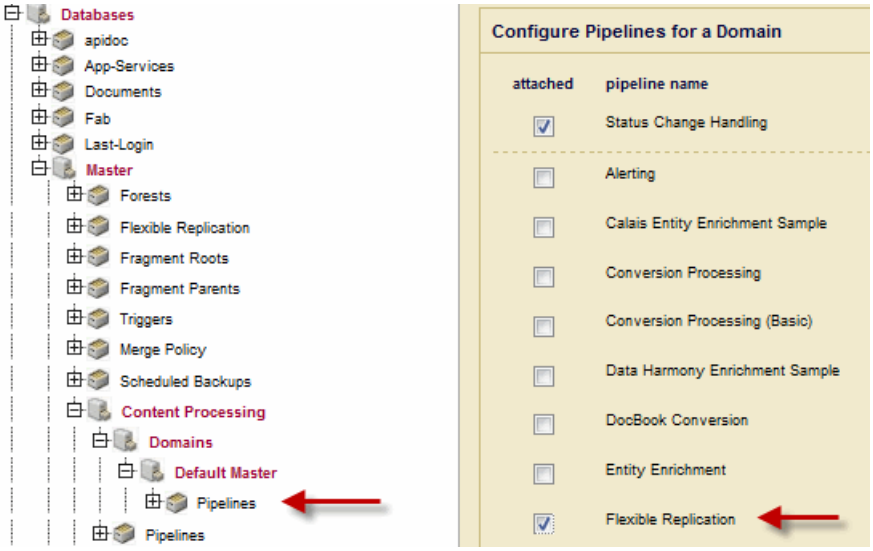


Then click OK.

5. Confirm that Flexible Replication appears in the Pipelines section of the Content Processing Summary page:



6. On the left-hand navigation menu, navigate to the Content Processing > Domains > Default Master > Pipelines page. Select the Status Change Handling and Flexible Replication pipelines and click OK:

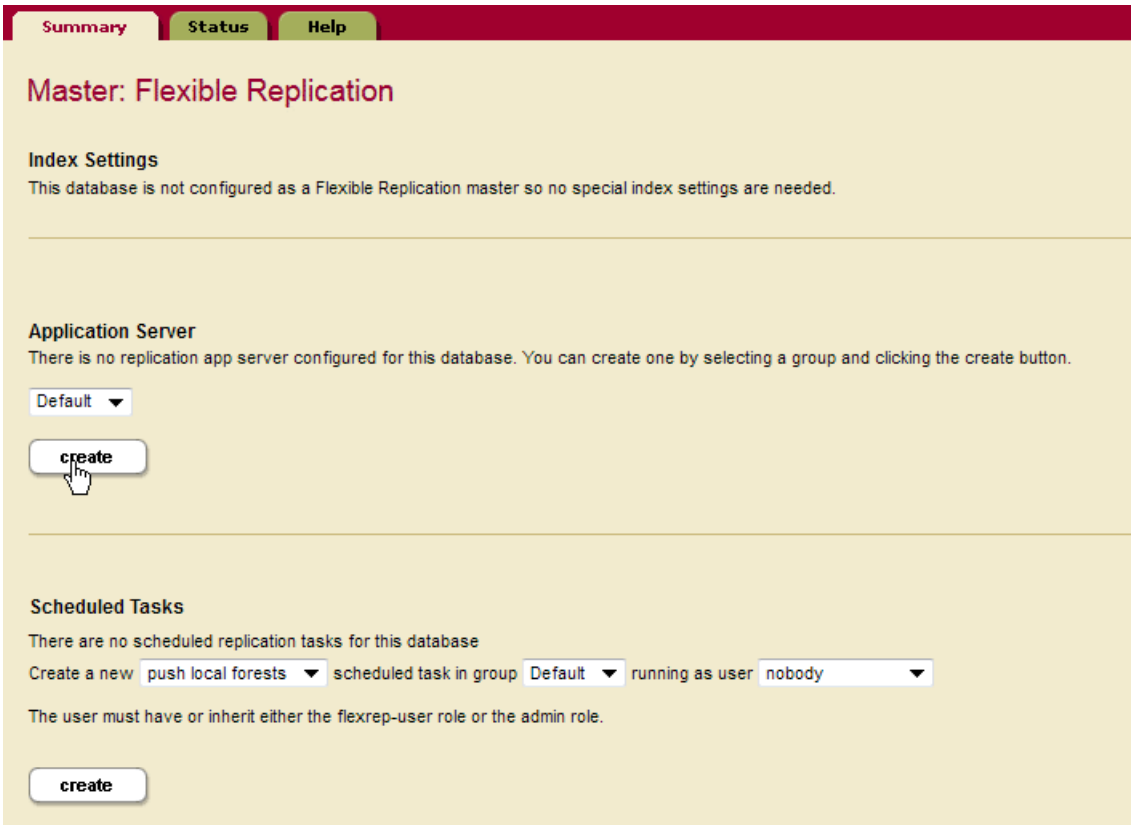


2.3 Creating a Replication App Server

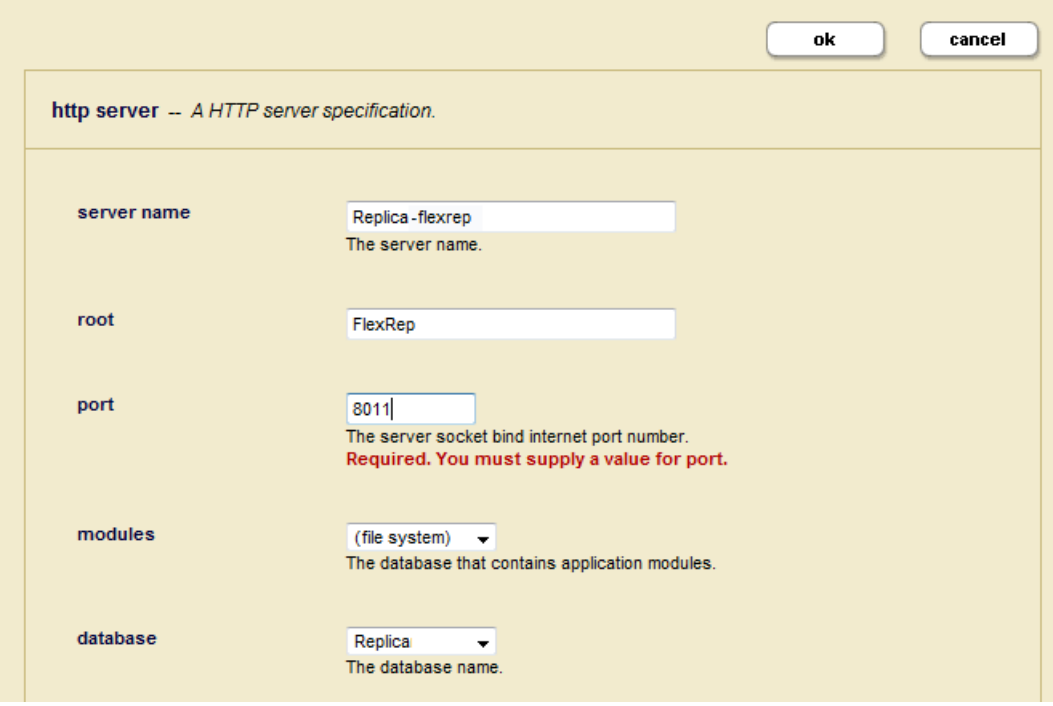
In a push replication configuration, the Replica database must be connected to a Replication App Server, an HTTP server used to communicate between the Master database and the Replica database. In a pull replication configuration, the Master database must be connected to a Replication App Server.

The following procedure describes how to create an App Server for a Replica in a push replication configuration.

1. Under the Replica database, navigate to Flexible Replication. On the Flexible Replication Administration page, click Create under Application Server:

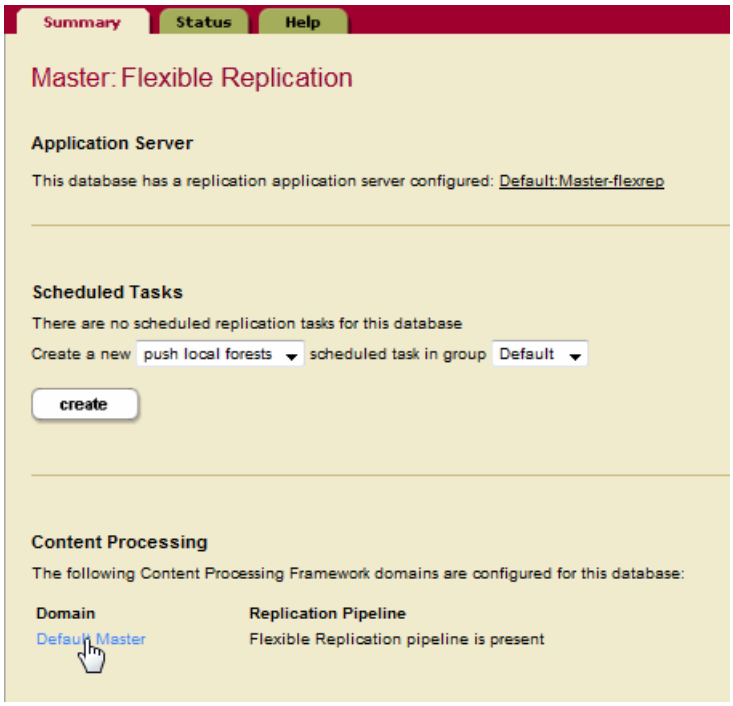


2. On the Create HTTP Server page, specify a port number of the App Server. Set the other fields as required, but do not change the server name, root directory, or database settings, which have been pre-set by MarkLogic Server. When finished, click OK.

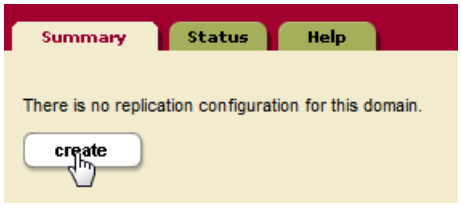


2.4 Configuring Push Replication on the Master Database

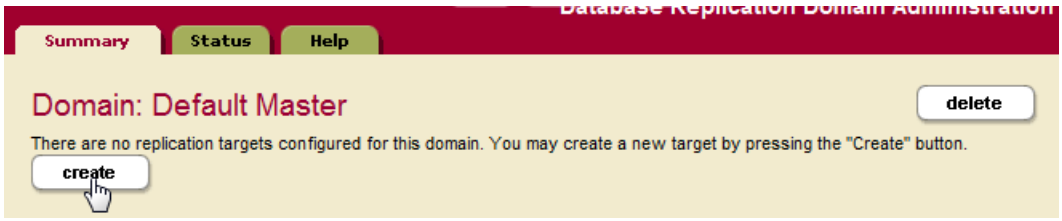
1. For the Master database, navigate to Flexible Replication on the Flexible Replication Administration page, and click Default Master under Content Processing:



2. Click Create to configure replication for the Default Master domain:

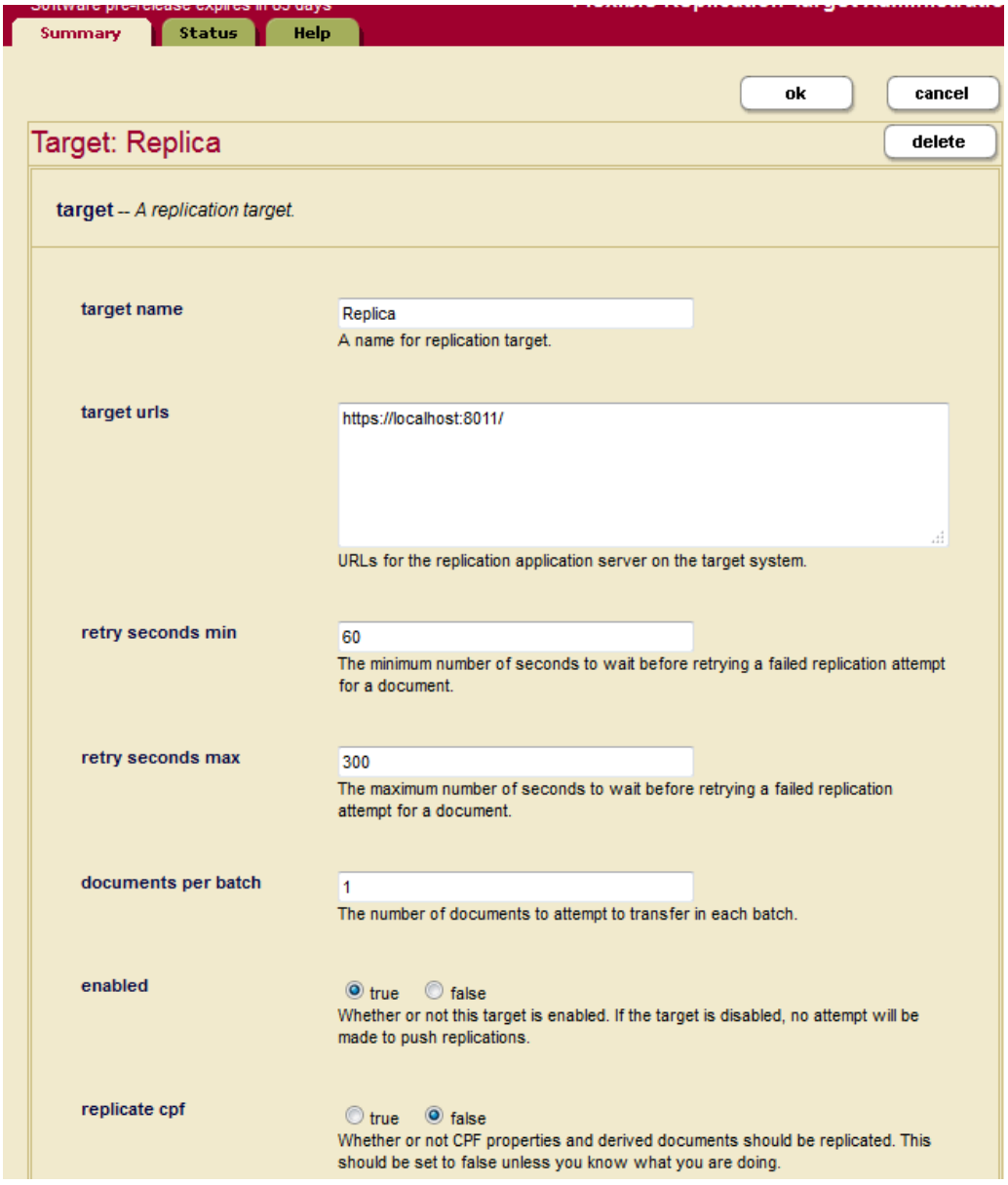


3. Click Create to configure a replication target for the Default Master domain:

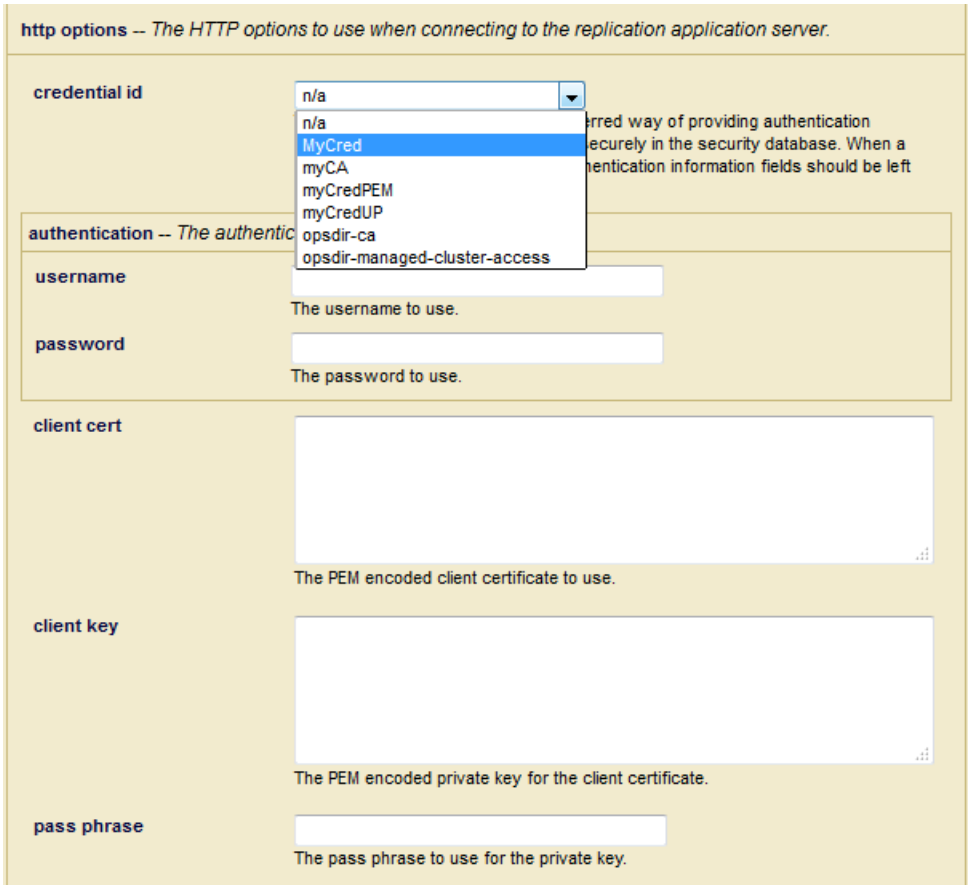


4. In the Database Replication Target page, specify a target name, the target URL that includes the hostname and port number you specified for the Replica's App Server in "Creating a Replication App Server" on page 15.

Note: In this example, you will be using a secure credential to access the Replica App Server, so the URL must start with `https`.



5. In order to communicate securely with the Replica, you should configure a secure credential, as described in Secure Credentials in the *Security Guide*. Assuming that you have configured a secure credential, select the name of the credential from the credential id menu.



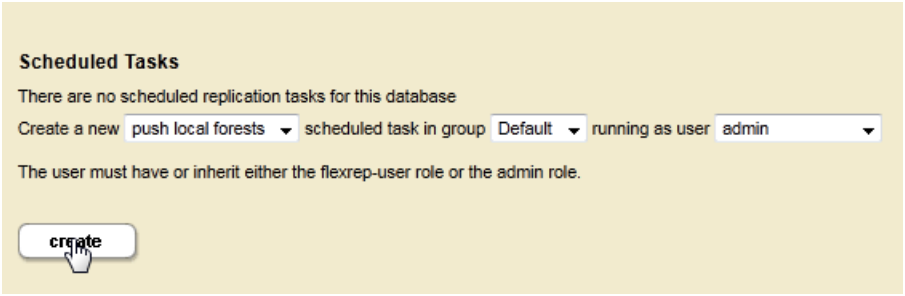
6. Click OK.

Note: There are additional security options you may want to enable for your configuration, as described in “Replication Security” on page 23.

2.5 Scheduling Replication Push Task

You must create a scheduled task to periodically push updated content on the Master to the Replicas. When a document is updated on the Master, it is immediately replicated to the Replica. A scheduled replication task provides a retry mechanism in the event the initial replication fails. A task is also required to replicate deletes and to replicate all of the documents that were on the master before replication was enabled. For more information on scheduled replication tasks, see “Creating a Scheduled Replication Task” on page 32.

1. For the Master database, navigate to Flexible Replication and click Create under Scheduled Tasks:



2. Create a Scheduled Task that defines the frequency in which documents are to be replicated from the Master to the Replica. You must also specify the task user. Do not specify a task host. Click OK.



2.6 Loading Documents into the Master Database and Checking Replication

At this point, any documents loaded into the Master database will be replicated to the Replica database.

> **Note:** Be sure to load your data using the same URI as specified in your replicated domain. For example, if you use the default URI (/), make sure all of your document names start with '/'.

Methods for loading content into a database include:

- Using the XQuery load document functions, as described in Loading Content Using XQuery in the *Loading Content Into MarkLogic Server Guide*.
- Setting up a WebDAV server and client, such as Windows Explorer, to load your documents. See the section Simple Drag-and-Drop Conversion in the *Content Processing Framework Guide* guide for information on how to configure a WebDAV server to work with Windows Explorer.
- Creating an XCC application, as described in Using the Sample Applications in the *XCC Developer's Guide*.

One way to confirm the content has been replicated to the replica is to use the explore feature in Query Console to view the contents of the Replica database. For details on how to use Query Console to explore the contents of a database, see Exploring a Database in the *Query Console User Guide*.

3.0 Configuring Replication

This chapter describes how to configure your MarkLogic Servers for replication. The topics in this chapter assume you are familiar with the replication principles described in “Understanding Flexible Replication” on page 6 and the basic configuration procedures described in the “Flexible Replication Quick Start” on page 11.

For details on how to write scripts to configure Flexible Replication, see Scripting Flexible Replication Configuration in the *Scripting Administrative Tasks Guide*.

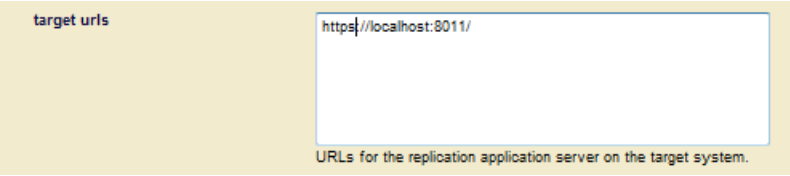
This chapter includes the following sections:

- Replication Security
- Defining Replicated Domains
- Configuring Replication App Servers
- Configuring Push Replication
- Creating a Scheduled Replication Task
- Configuring Pull Replication
- Configuring Alerting With Flexible Replication
- Backing Up, Restoring, and Clearing the Master and Replica Databases

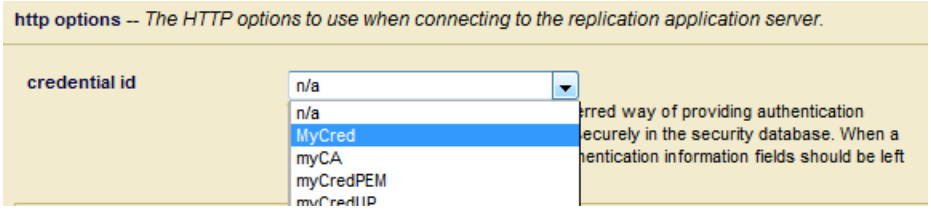
3.1 Replication Security

The `flexrep-admin` role is required to configure replication. The user who will access the Replica App Server when pushing, or access the Master App Server when pulling, requires the `flexrep-user` role. Though you will typically configure replication as the Admin user, you should create a unique replication user to be associated with the replication tasks. The replication user must be given the `flexrep-user` role and have the privileges necessary to update the domain content on both the Master and Replica App Servers.

If you configure your replication target to use any of the security schemes described in this section, the Target URL must start with `https`.



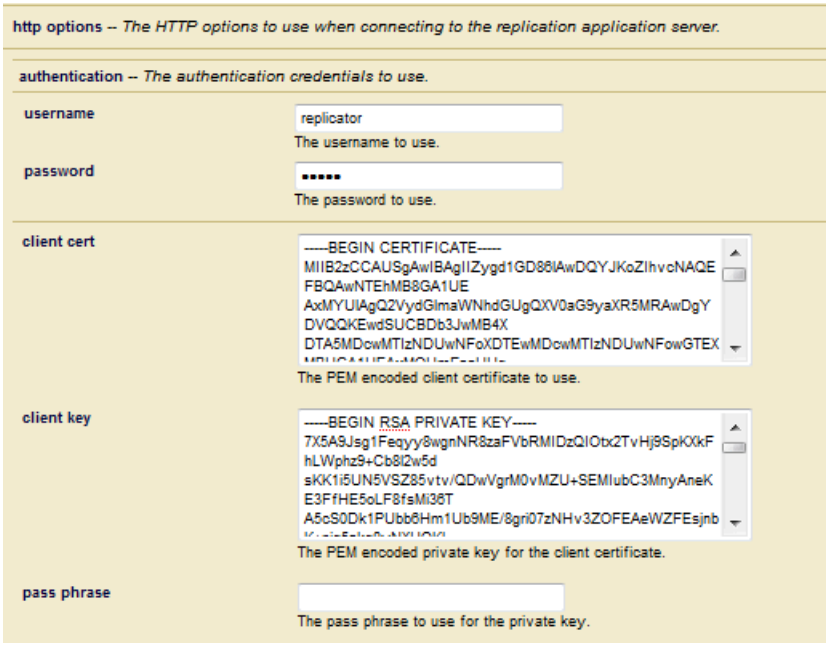
It is a best practice to create a secure credential, as described in Secure Credentials in the *Security Guide*, to communicate with the replication target. To do so, select the name of the secure credential from the credential id menu.



As an alternative to secure credentials and mostly to provide backward compatibility with previous versions of MarkLogic, you can configure SSL on your Master and Replica App Servers to encrypt the replicated data passed between them. For details on configuring SSL on App Server, see the Configuring SSL on App Servers in the *Security Guide*.

If SSL on the Replica App Server is configured to require a client certificate from the Master, paste the PEM-encoded client certificate and client key in the fields located at the bottom of the Database Replication Target Administration page. If the client key is encrypted, you must also specify a pass phase.

The same security configuration applies when configuring the Replica App Server for pull replication, if the Master App Server requires a client certificate from the Replica. For details on how to configure pull replication, see “Creating a Pull Replication Task” on page 38

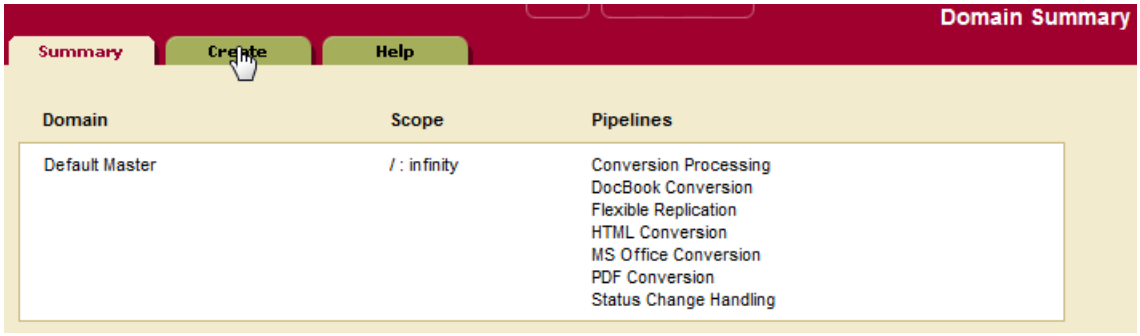


3.2 Defining Replicated Domains

Before you can define replicated domains, you must enable CPF as described in “Configuring a Flexible Replication Pipeline for the Master Database” on page 12.

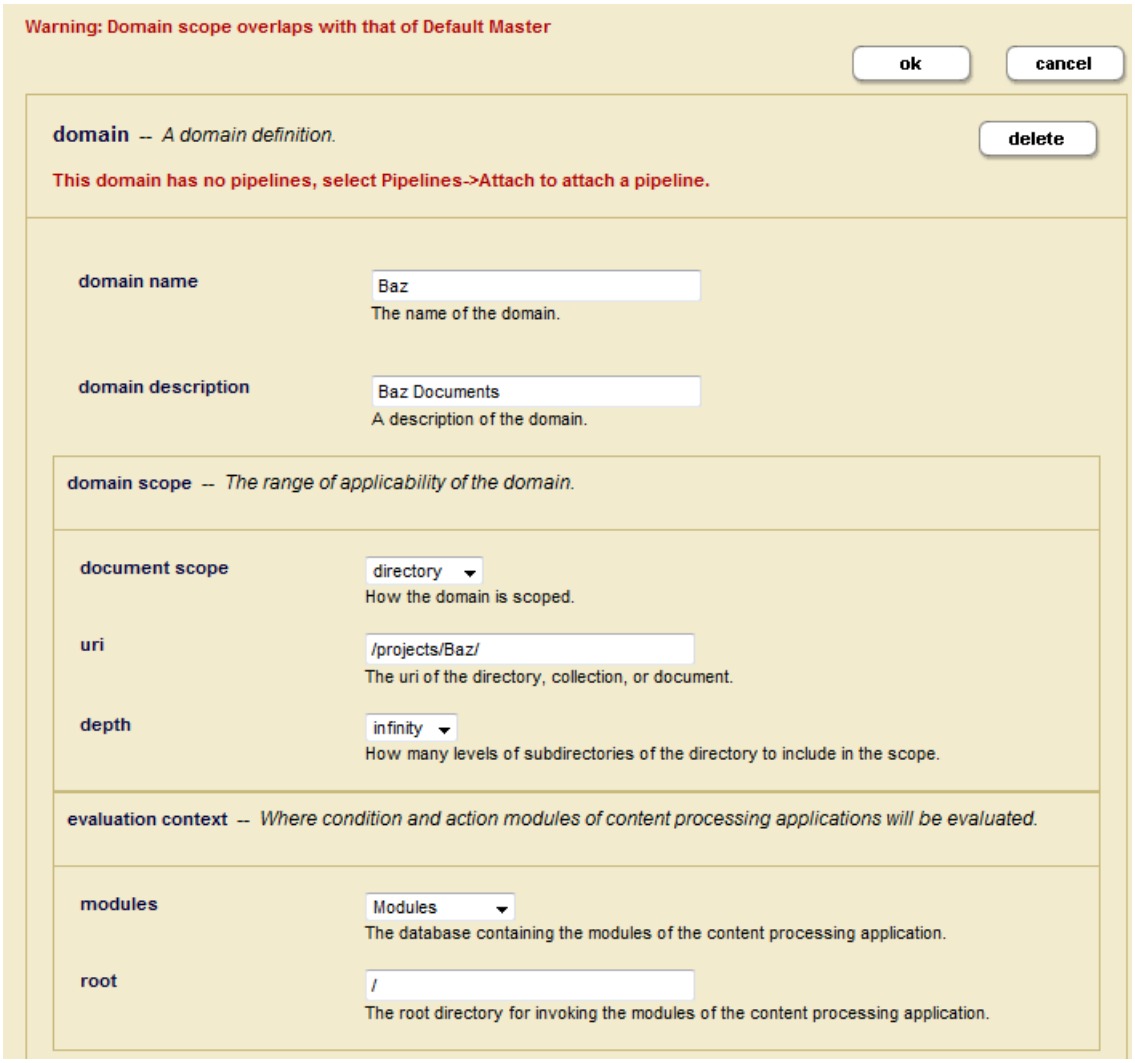
CPF domains are described in detail in the Understanding and Using Domains chapter in the *Content Processing Framework Guide* Guide. The purpose of this section is to describe how to create a domain that defines the scope of the documents in the Master database to be replicated to the Replica database.

Warning Each domain must contain a unique set of documents. No single document can be in more than one domain.



The following procedure describes how to create a new domain for defining the scope of the replicated documents. By default, CPF creates a Default Master domain to replicate from the "/" root directory of the Master database. The "/" root means that the document URIs must be preceded by a "/", such as `/foo.xml` or `/content/foo.xml`. You can also set the Document Scope of the domain to replicate a user collection or an individual document.

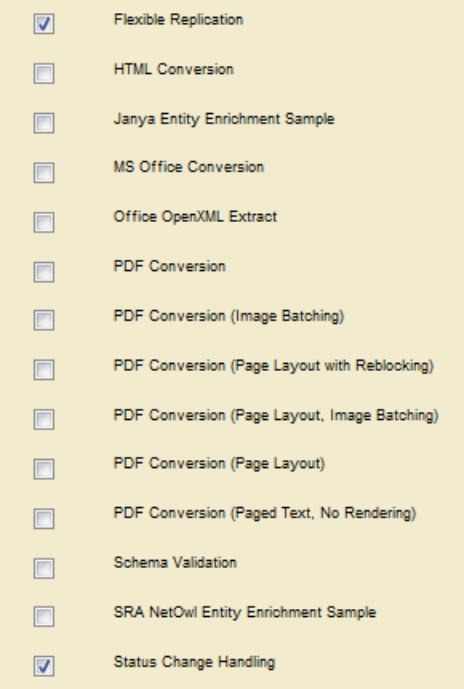
1. To replicate a portion of a database, you can create another domain. For example, you can create a domain that specifies the documents in the `/projects/Baz/` directory on the Master database, as shown below.



You may receive a warning that this domain overlaps with the Default Master domain. This means that the both the Baz domain and the Default Master domain are configured to replicate the documents in the `/projects/Baz/` directory. You cannot have any document that is in more than one domain, so you must either delete or modify the Default Master to resolve this conflict.

Note: If you want to create a domain to replicate the entire database, you can assign a default collection to all of your users and then set the Document Scope to Collections and the URI to the name of the default collection. For details on how to establish a default collection for a user, see Creating a User in the *Administrator's Guide*.

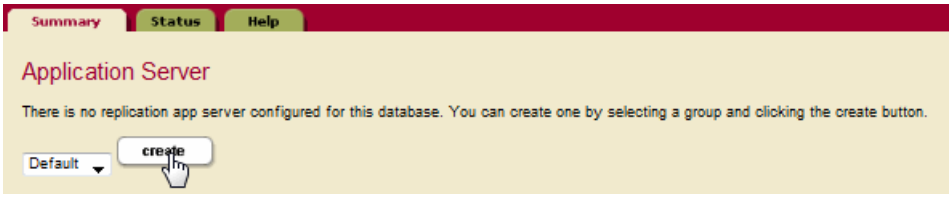
2. Select Pipelines in the navigation frame and select the Flexible Replication and Status Change Handling pipelines. These are the minimum pipelines required for replication. You can select other pipelines, as required for your configuration.



3.3 Configuring Replication App Servers

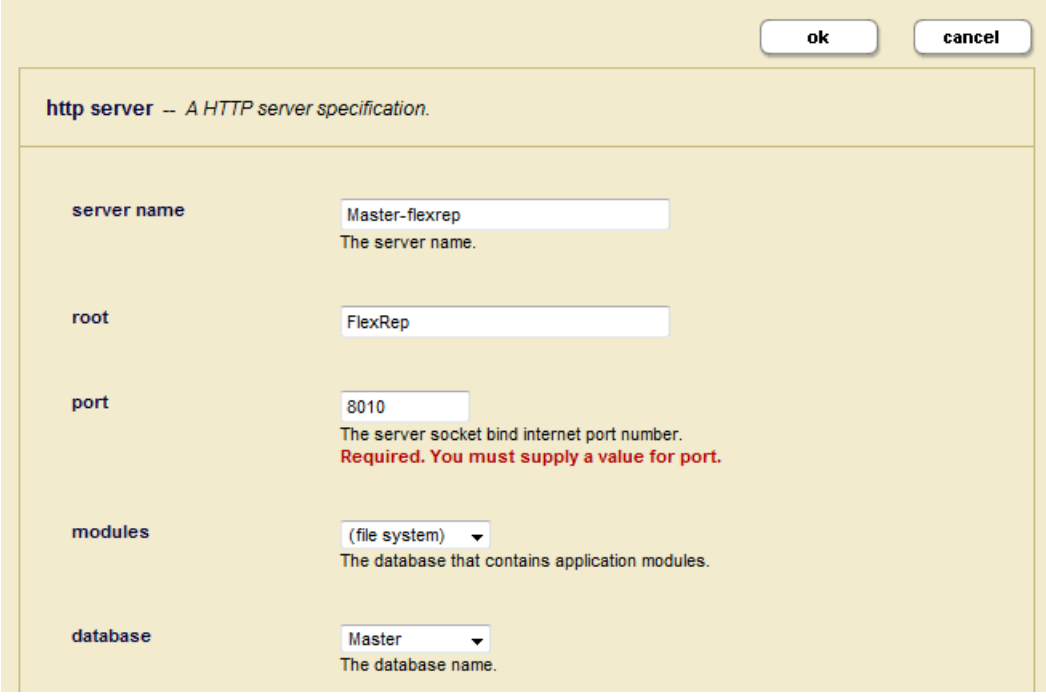
As described in “Creating a Replication App Server” on page 15, in a push replication configuration, the Replica database must be connected to a Replication App Server. In a pull replication configuration, the Master database must be connected to a Replication App Server. This section goes into more detail on creating and configuring Replication App Servers.

1. Create a Replication App Server by clicking Create in the Application Server section of the Flexible Replication Administration page:



Note: The Server Name, Root, and Database fields are preconfigured and should not be changed.

2. Enter a unique port number for the App Server:



The purpose of the other fields in the HTTP Server Configuration page are described in Creating a New HTTP Server in the *Administrator’s Guide*. If you are going to configure SSL on the App Server to require a client certificate, you must specify the certificate in the

Database Replication Target Administration configuration page, as described in “Replication Security” on page 23.

For example, if you are configuring the Master database to push updates to a Replica App Server that requires a client certificate, you must include a client certificate in the Database Replication Target Administration configuration page for the Master database. If you are configuring a Replica App Server to pull updates from the Master database that requires a client certificate, you must you must include a client certificate in the Replication Pull configuration page for the Replica database. For details on providing a client certificate for either a push or pull replication configuration, see “Replication Security” on page 23.

3.4 Configuring Push Replication

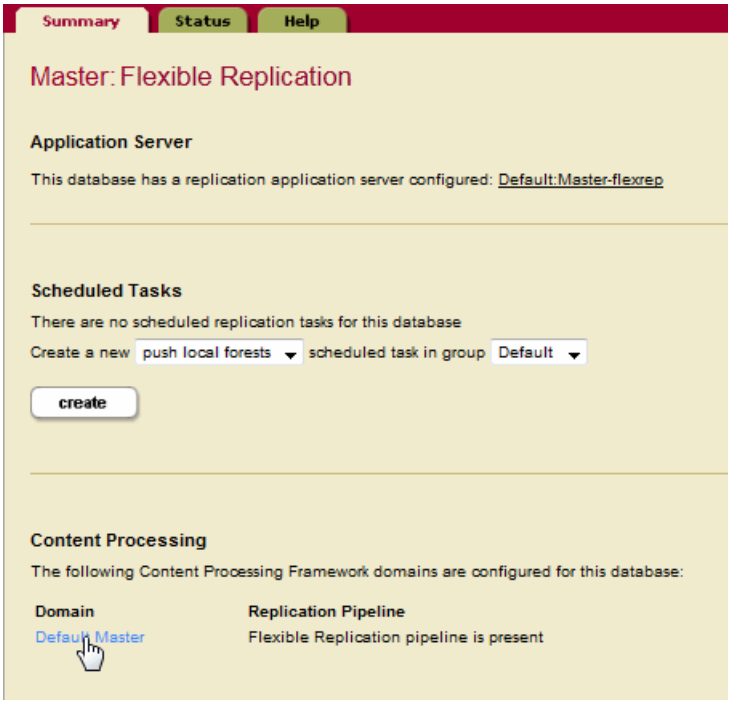
This section describes how to configure your Master database for Push Replication.

Before you can configure a replication target, you must have content processing enabled for the Master database. After configuring push replication, you must create a scheduled task, as described in “Creating a Push-Local-Forests Replication Task” on page 36.

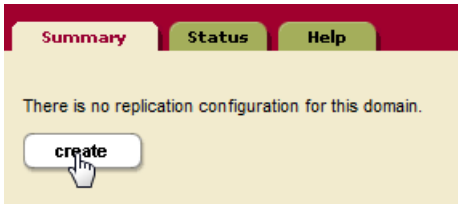
1. Under Databases, click the Master database.

2. Click Flexible Replication under the Master database.

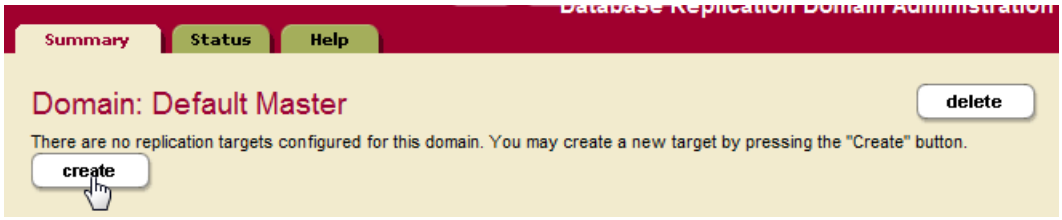
3. Locate the Content Processing section on the Flexible Replication Administration page and click on the domain name.



4. Click Create to configure replication for the domain:



5. Click Create to configure a replication target for the domain:

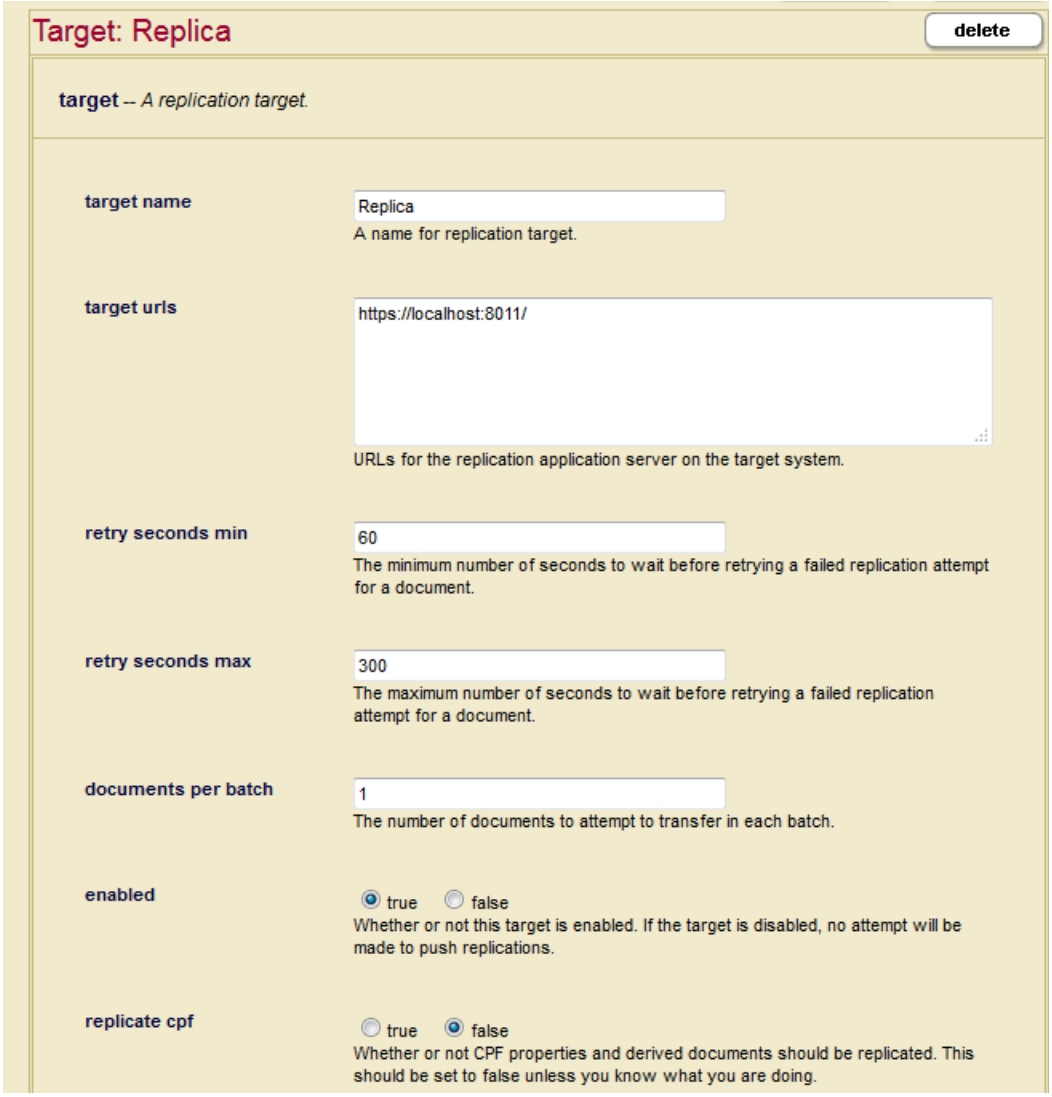


6. In the Database Replication Target Administration page, specify a target name and one or more target URLs that include the hostname and port number you specified for your Replica App Servers in “Creating a Replication App Server” on page 15. The Retry Seconds Min/Max and Documents Per Batch setting are for scheduled tasks, as described in “Creating a Scheduled Replication Task” on page 32.

   Under Authentication, specify the username and password for a user assigned the `flexrep-user` role. This user must have the same username/password as the user on the Master App Server who creates and updates the documents. This user needs to have sufficient permissions on the target system to insert and update the replicated documents.

   Because Flexible Replication passes through the permissions set on the Master App Server when replicating, the user on the target must have permissions to update the document later on. Permissions can either be configured by the administrator on both the Master and Replica App Server, or you can use a filter to adjust the document permissions so the target user can later update the document, as described in “Adding Document Permissions” on page 58.

You can optionally disable pushing replication to the target by setting `Enabled` to `false`. Do not set the `Replicate CPF` option to `true` without the direction from MarkLogic Customer Support.



7. If the Replica App Server requires a client certificate, the Target URL must start with HTTPS. Paste the client certificate, client key and pass phrase (if the client key is encrypted) in the fields located at the bottom of the Database Replication Target Administration page. For details on how to configure the Master to provide a client certificate, see “Replication Security” on page 23.

3.5 Creating a Scheduled Replication Task

Regardless of whether you configure replication as push or pull, you must create a scheduled task to periodically replicate updated content on the Master to the Replicas. A scheduled replication task does the following:

- Moves existing content that was on the Master before replication was configured (zero-day replication).
- Provides a retry mechanism in the event the initial replication fails.
- Replicates deletes on the Master to the Replica.
- Provides the ability to pull data from a Master server located outside a firewall.

There are three types of scheduled tasks, as described in the table below.

Task	Description
push-local-forests	The push-local-forests task setting is the preferred method for retry and zero-day replication. Once the initial scheduled interval is reached, this task pushes each batch of documents, immediately following the previous batch, regardless of the scheduled task frequency. If you had previously used the push task, switching over to the push-local-forests setting will provide better performance. **Note:** The push-local-forests task for each forest can only be respawned a maximum of 500 times per replication period. in order to avoid overflowing the task limit. After 500 respawns, the task will wait for the next replication period to start replicating again. For each respawn, the task will replicate the batch size to the replica. Do not specify a task host for the push-local-forests task. This allows the task to run on all hosts in the cluster.
push	The push task pushes a batch of documents from all of the forests in the database. This task pushes one batch from the entire database per scheduled interval. This form of push replication is available for users who configured replication on earlier versions of MarkLogic Server. If you are currently configured for this form of push replication, you will obtain better performance if you switch to push-local-forests replication. The push task runs on a single host, which you must specify as the task host.

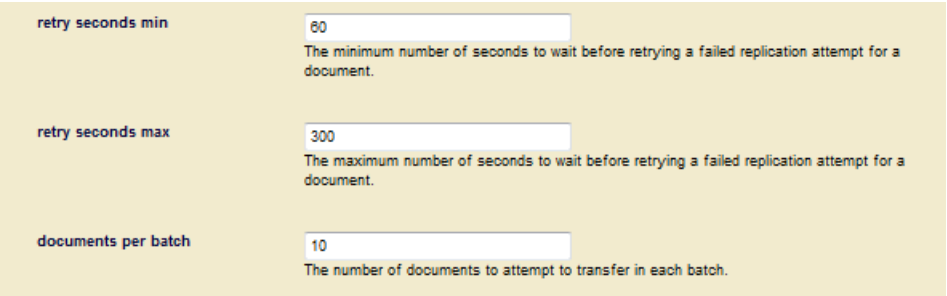
Task	Description
pull	The pull task is configured on a Replica database to pull data from the Master database. This is the only replication option available to a Replica, so the pull scheduled task must serve as the initial replication mechanism in addition to retry and zero-day replication. The pull task runs on a single host, which you must specify as the task host.

For all scheduled replication tasks, the replication retry configuration is a combination of the scheduled task frequency settings and the target settings, Retry Seconds Min, Retry Seconds Max, and Document Per Batch.

The task type in the Scheduled Task Configuration page indicates the interval at which each scheduled replication task is to run. In most configurations, the task type will be minutely. For details on configuring scheduled tasks, see Scheduling Tasks in the *Administrator's Guide*.



The Retry Seconds Min/Max settings in the Database Replication Target Administration page indicate the minimum and maximum number of seconds any documents that failed to replicate are eligible for replication retries. In a push configuration, the Documents Per Batch setting specifies how many documents that have failed to replicate are to be retried during each scheduled task interval. In a pull configuration, the Documents Per Batch setting specifies the total number of replicated documents (retries of failed documents and newly added or updated documents) that are to be pulled from the Master during each scheduled task interval.



For example, the retry minimum is 30 seconds, the retry maximum is 300 seconds (five minutes) and the scheduled replication task period is every one minute. If a document fails replication, it is eligible to be retried in 30 seconds. This means that MarkLogic Server will attempt to replicate the document at the next minute interval. The retry interval is doubled each time the document fails to replicate until the interval reaches the maximum retry setting, at which time the retry interval remains at the maximum. So, if the document in our example fails a second time, it will be eligible for retry in one minute. Should the document fail to replicate the second time, the retry interval is set to two minutes, and so on until the interval reaches the five-minute retry maximum setting, after which MarkLogic Server tries to replicate the document every five minutes.

The Documents Per Batch setting also plays a role with replication retries. For example, if the batch value is one and five documents have failed to replicate, then MarkLogic Server will only attempt to retry one failed document that is eligible for replication retries during each scheduled task interval. The document selected for replication retry is the earliest eligible document.

Though the Documents Per Batch setting is 1 by default, a more typical value is in the range of 10 - 100. If there is a large number of documents to be replicated in a zero-day fashion, you can maximize the load speed by closely matching the scheduled interval with the Documents Per Batch setting, so that the maximum number of documents are loaded within the time interval. For example, if your scheduled interval is one minute and you have determined that you can replicate a maximum of 250 documents per minute, then the optimum Documents Per Batch setting would be 250.

When a flexible replication domain includes multiple targets, the scheduled task will use the smallest of the configured Documents per Batch settings. So if one target has Documents per Batch set to 1, and another has Documents per Batch set to 100, the scheduled task will use a batch size of 1.

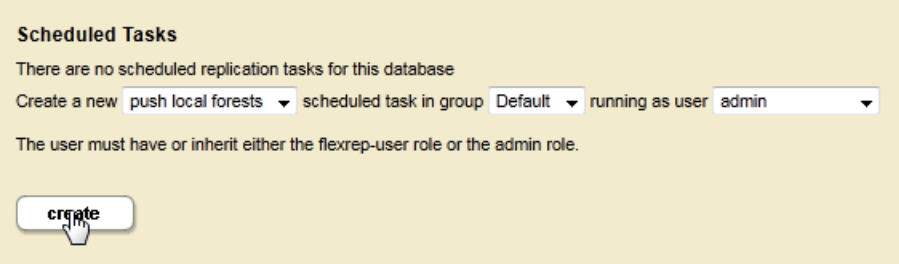
3.5.1 Creating a Push-Local-Forests Replication Task

This section describes how to configure a push-local-forests replication task to provide the Master with the means to replicate deletes to the Replica, retry replication on documents that have failed the initial attempt to replicate, and to replicate documents that were in the replicated domain before replication was configured.

> **Note:** A scheduled push-local-forests task should be run by a user assigned the `flexrep-admin` role or the `admin` role.

Do the following to create a push-local-forests scheduled replication task:

1. For the Master database, navigate to Flexible Replication, select push local forests in the drop-down menu, select the group in which you want to schedule the task, and click Create under Scheduled Tasks:



2. Create a Scheduled Task that defines the frequency in which documents are to be replicated from the Master to the Replica (when using push-local-forests, this interval only defines the timeframe to wait before pushing the initial batch). You must also specify the task user.

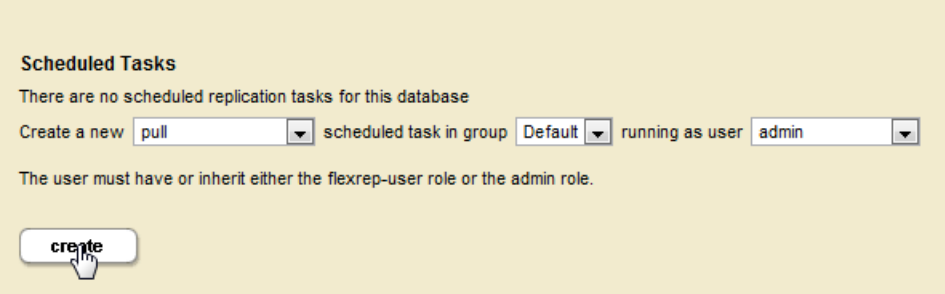
   Note: Do not specify a host if you are creating a push-local-forests scheduled task.



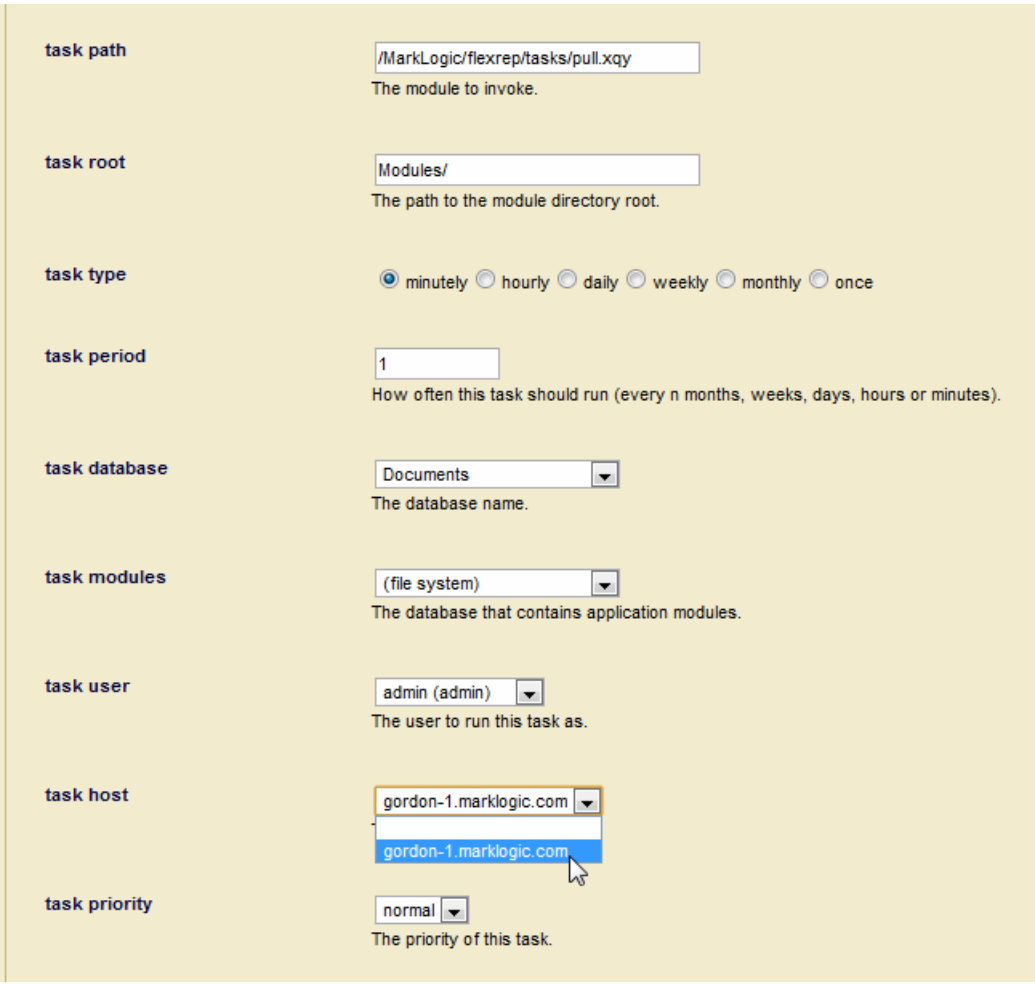
3.5.2 Creating a Pull Replication Task

In a pull replication configuration, the Replicas pull updates from the Master. Unlike push replication, which replicates as soon as the documents on the Master are updated, the only way to configure pull replication is by means of a scheduled task.

1. On the Replica, create a scheduled task to pull the content from the Master:



2. Specify the pull frequency. In the task user pull-down menu, select the user with the `flexrep-user` role on a Replica host. Under task host, select a Replica host or leave the field empty so the task can run on any host in the cluster. Click OK.



3.6 Configuring Pull Replication

Pull Replication is useful when a Replica is behind a firewall that only allows its internal servers to pull from a Master server outside the firewall. This section describes how to configure your Replica database for Pull Replication.

Note: If the Replica is to act as a Master for another Replica, then you must have CPF enabled on the Replica, as described in "Configuring a Flexible Replication Pipeline for the Master Database" on page 12. However, if the Replica has CPF installed and it is not to act as a Master, then you must disable the Flexible Replication pipeline on the Replica.

The procedure for configuring Pull Replication on the Master and Replica databases is as follows:

1. On a Master host, define a replication target for the Master Content Processing Domain with push replication disabled. You can do this by either not defining a target URL in the target configuration page and enabling the replication target.

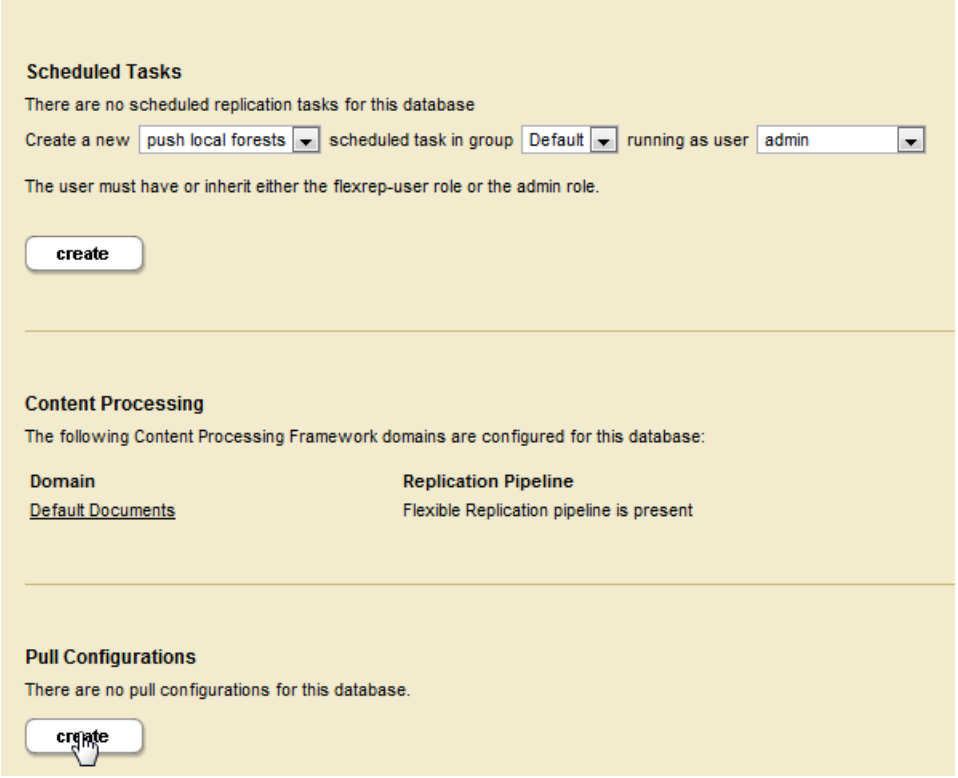


2. Under Databases, click the Replica database.

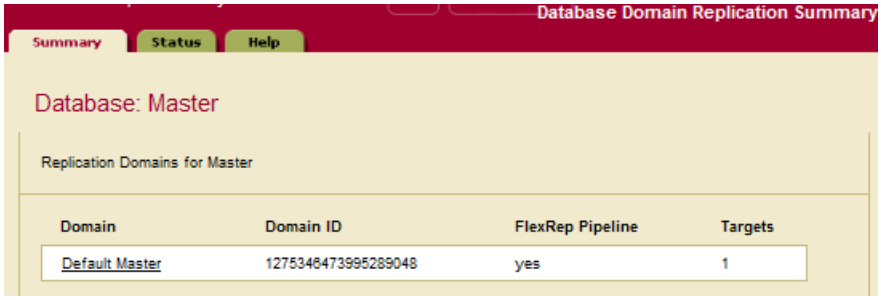
3. Click Flexible Replication under the Replica database.

4. Create a Pull Replication Task, as described in “Creating a Pull Replication Task” on page 38.

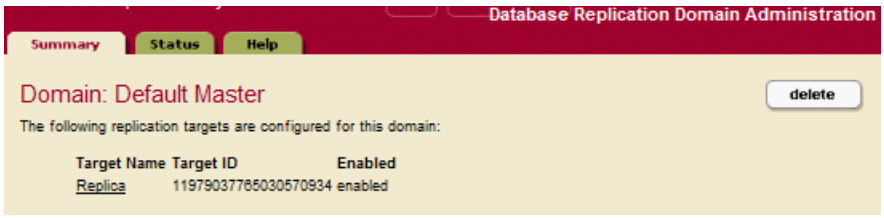
5. Under Pull Configuration, click Create:



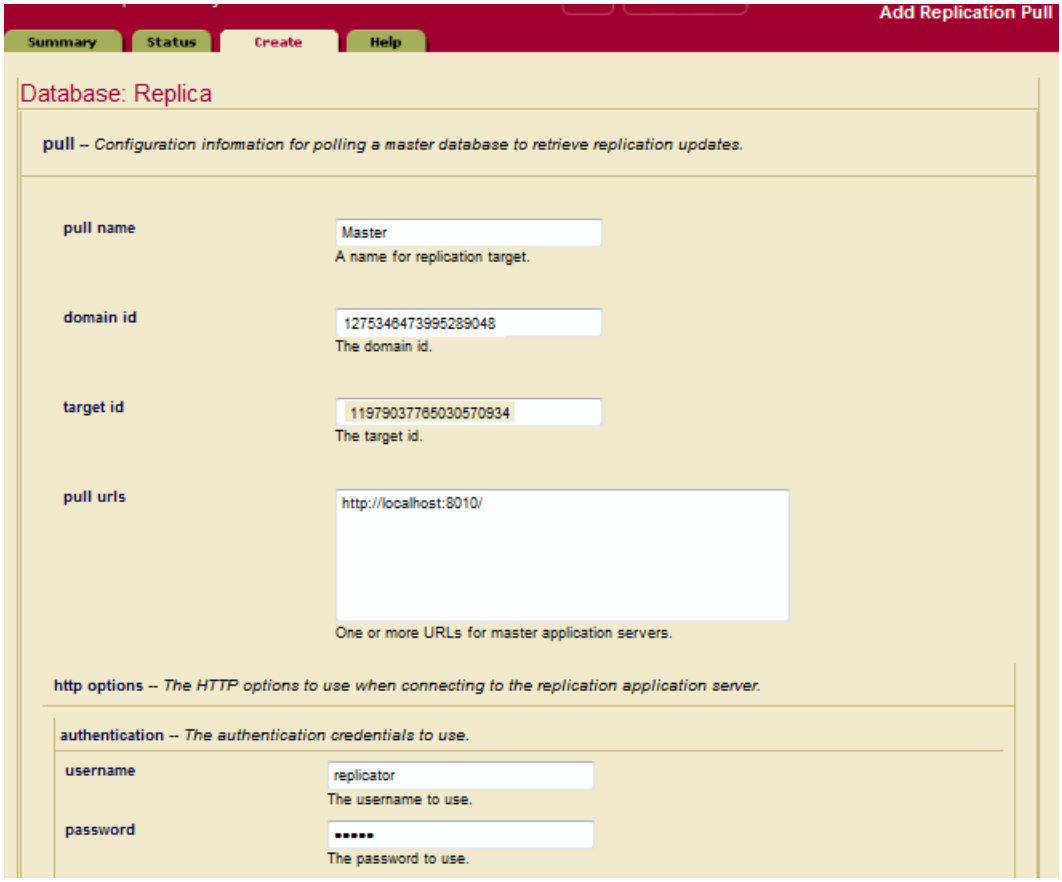
6. On a Master host, obtain the Master domain id from the Database Domain Replication Summary page:



7. On a Master host, obtain the Replica target id from the Database Domain Replication Administration page. If a Replica target does not exist, create one, as described in “Configuring Push Replication” on page 29, Step 5.



8. In the Replication Pull Administration page on the Replica host, enter a pull name, the domain id of the Master, the target id of the Replica, and one or more pull URLs for the Master App Servers. Fill in the username and password fields required to access the Master App Servers.



9. You can optionally specify a client certificate, if one is required by the Master App Server. For details on how to configure the Replica to provide a client certificate, see “Replication Security” on page 23.

3.7 Configuring Alerting With Flexible Replication

Combining alerting with flexible replication is often referred to as QBFR (Query-Based Flexible Replication). Query-based flexible replication enables customizable information sharing (using filters) between systems, allowing for the easy and secure distribution of portions of data even across disconnected, intermittent, and latent networks.

This type of replication is based on a query (an alert) that triggers because of an event that matches that query. A user can have more than one alert, in which case they would receive documents that match any of their alerts. In addition to queries, the permissions for a user are taken into account. The user will only receive replicated content that they have permission to view in the database. If the permissions change, the replica will be updated accordingly. Most often query-based flexible replication is a pull configuration, but it can also be set up as a push configuration.

By setting up alerts, replication takes place any time content in the database matches that query. Any new or updated content within the domain scope will cause all matching rules or alerts to perform their corresponding action. Query-based flexible replication can be used with filters to share specific documents or parts of documents. Filters can be set up in either a push or pull configuration.

To set up query-based flexible replication, you need to have the environment for flexible replication already configured. See “Flexible Replication Quick Start” on page 11 for information on setting up flexible replication using the Admin UI. Configuring Query-based Replication using the REST API in the *Scripting Administrative Tasks Guide* explains how to set up query-based flexible replication including alerts using scripting and REST.

QBFR users should have the `flexrep-user` role. QBFR targets will only replicate documents for which the associated user has read permission, as in normal MarkLogic security. Your configuration will most likely have additional or more complex permissions on documents.

This section contains these topics:

- Configuring Alerts
- Using QBFR

3.7.1 Configuring Alerts

Query-based flexible replication requires an alert to trigger the replication. These steps use XQuery to access MarkLogic built-in functions to create and configure alerts. To see the complete scripted version of this process, see Configuring Query-based Replication using the REST API in the *Scripting Administrative Tasks Guide*.

To configure an alert for query-based flexible replication you need to:

- Create an alerting domain using `alert:make-config`

- Create an alerting action using `alert:make-action`
- Associate flexible replication with the alerting domain using `flexrep:configuration-set-alerting-uri`

After the flexible replication, alerting domain, alerting action, and alert(s) have been set up, you associate the target of the alert with a user and create an alerting rule.

To do this:

- Associate the target of the alert with a user using `flexrep:configuration-target-set-user-id`
- Create an alerting rule with `alert:make-rule`

This section includes examples of these tasks using XQuery. See Configuring Query-based Replication using the REST API in the *Scripting Administrative Tasks Guide* for a complete scripting example using REST.

3.7.1.1 Create an Alerting Domain

Once you have your flexible replication environment configured, to set up query-based flexible replication you need to create an alerting domain. This example uses `alert:make-config` to create an alerting domain named “http://acme.com/alerting” containing “alerting rules for query-based flexrep” and then `alert:config-insert` is used to add it to the acme.com alert configuration:

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

alert:config-insert(
  alert:make-config(
    "http://acme.com/alerting",
    "qbfr", "alerting rules for query-based flexrep",
<alert:options/>))
```

3.7.1.2 Create an Alerting Action

Next you would create an alerting action using `alert:make-action`. This example creates an alerting action named “log”:

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

alert:action-insert("http://acme.com/alerting",
  alert:make-action(
    "log", "QBFR log action",
    xdmp:database("master-modules"), "/", "/log.xqy",
<alert:options/>))'
```

Note: Traditional alerting in MarkLogic requires that `log.xqy` exist in the modules database so it can be called when the alert triggers. For QBFR `log.xqy` will not be called and therefore does not actually need to exist.

3.7.1.3 Associate Flexible Replication with the Alerting Configuration

Next you need to associated the alerting configuration with a CPF domain. Use `flexrep:configuration-set-alerting-uri` to associate the configure domain (`my-cpf-domain` in this example) to be used for flexible replication.

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

let $domain := "my-cpf-domain"
let $cfg := flexrep:configuration-get($domain, fn:true())
let $domain-id :=

flexrep:configuration-insert(
  flexrep:configuration-set-alerting-uri($cfg,
    flexrep:domain-alerting-uri($domain-id)))
```

3.7.1.4 Associate the Target With a User

Once alerting has been set up, the next step is to associate the QBFR target with a user using `flexrep:configuration-target-set-user-id`.

```
xquery version "1.0-ml";
import module namespace flexrep =
  "http://marklogic.com/xdmp/flexible-replication"
  at "/MarkLogic/flexrep.xqy";

let $domain := "my-cpf-domain"
let $cfg := flexrep:configuration-get($domain, fn:true())
let $target-id := flexrep:configuration-target-get-id($cfg, "QBFR target")
let $user-id := xdmp:user("User1")

flexrep:configuration-insert(
  flexrep:configuration-target-set-user-id(
$cfg, $target-id, $user-id))
```

The user can also be configured with `flexrep:target-create` when creating the target.

3.7.1.5 Create an Alerting Rule

The next step is to create an alerting rule for the replication using `alert:make-rule`. This example uses `alert:make-rule` to create an alert that says if any new content contains the words “dna” or “rna” send an email alert to `me@somedomain.com`.

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

alert:make-rule(
  "nucleic acids email",
  "Alert me to anything concerning nucleic acids",
  0,
  cts:or-query((
    cts:word-query("dna"),
    cts:word-query("rna")
   )),
  "email",
  <alert:options>
    <alert:email-address>me@somedomain.com</alert:email-address>
  </alert:options>
)
```

See Configuring Query-based Replication using the REST API in the *Scripting Administrative Tasks Guide* for a complete scripting example, including how to create an alerting rule. For more details about creating an alert, see Creating Alerting Applications in the *Search Developer’s Guide*.

3.7.2 Using QBFR

Query-based flexible replication is useful in circumstances where you need to share specific information (documentation, photographs, and so on) with many others in different locations. to improve security or reliability, flexible replication allows data to be transformed and filtered before replication. This enables control of what documents or parts of documents are shared, or how data should be presented. In remote locations you might have disconnected devices or intermittent connectivity, so a pull setup will be more effective.

Query-based flexible replication enables people in the field to have access to specific, targeted information in a timely manner, be able to update or add to that information, and then replicate the results back to headquarters, with all the security and reliablility provided by MarkLogic. For example, geologists for oil and gas companies can take information that they need with them to remote locations, perform field analysis, take pictures, write up reports, and share that information when they reconnect to the network.

Safety inspectors could perform a quick search online to acquire the exact information they need to inspect a certain location and replicate the data to a laptop. When the on-site inspection has been performed, analysis and information can be updated locally. Once connection to the network has been established, the results are replicated and shared back the main office.

Researchers working in different company locations can share large data sets of information, with fast local access and the ability to independently update the data. The information can then be aggregated and shared, as appropriate, with others throughout the company.

QBFR does not guarrantee the order in which documents will arrive, it only guarrantees that the final version will arrive.

With a push configuration, if the devices are disconnected, the server will continue to retry the operation to send the information. You could end up with a document error condition if you have too many retries in a row. The `flexrep:document-reset` function can be used to clear the error condition and schedule replication of the document. You can use `flexrep:domain-target-status` to query for documents that have had an error in replication.

3.8 Backing Up, Restoring, and Clearing the Master and Replica Databases

The configuration for flexible replication is stored in the Master and Replica databases. Clearing a Master database has the effect of deleting the target configurations, as well as all document-level information about target replication status. When you backup a Master database, you are also saving the replication configuration information. Should you restore that database to another cluster, the flexible replication configuration will also be present for the database in that cluster. Should you restore a database that is not configured for replication to one that was previously configured for replication, the replication configuration will be lost.

Before clearing or restoring a Master database, you can save the replication configuration by calling the `flexrep:configuration-get` function. Once the Master is cleared or restored, the replication configuration can be restored using the `flexrep:configuration-insert` function. Alternatively you can recreate the replication configuration programmatically, as described in Scripting Flexible Replication Configuration in the *Scripting Administrative Tasks Guide*.

Clearing a Replica database impacts the Master in that it no longer understands the status of the target. The same issues related to backing up and restoring the Master database described above also apply to backing up and restoring the Replica database. If the target status is lost on the Replica, new updates will replicate to the target, but the Master will not know that the older, unchanged documents are missing. Before clearing or restoring a Replica database, disable replication. After clearing or restoring the Replica database, delete the existing target on the Master, create a new target with the same configuration as the old one, and re-enable replication. The scheduled task will gradually populate the new target's database with all the documents in the domain.

If you have pull replication configured on the Replica, then you can save the configuration by calling the `flexrep:pull-get` function before clearing the database. After clearing or restoring the Replica database, you can restore the pull replication configuration by calling the `flexrep:pull-insert` function.

3.8.1 Interrupted replication

In flexible replication, a large binary is replicated in chunks to the replica. In normal circumstances, when all chunks are replicated, the replica will reassemble these chunks back into one large binary. If the master dies while replicating some of the chunks, the chunks that were already replicated will be left on the replica. When the master comes back online again, the process will resume.

In some cases the master may be permanently removed while the chunks are being replicated. To reclaim disk space, you can use the timestamp of the binary chunks to find and remove them. The `flexrep:binary-chunk-uris(ts as xs:dateTime)` function returns the URIs of all binary chunks that are older than the given wall clock time. This will list all of the binary chunks that are older than the time specified by `ts`.

For example:

```
xquery version "1.0-ml";
import module namespace flexrep =
"http://marklogic.com/xdmp/flexible-replication"
  at "/MarkLogic/flexrep.xqy";

  flexrep:binary-chunk-uris(xs:dateTime("2014-10-01T08:00:00"))

  (: Returns the URIs of binary chunks that were created before
2014-10-01T08:00:00. :)
```

Note: The `flexrep:binary-chunk-uris` API requires the `flexrep-admin` privilege and the URI lexicon must be enabled (by default it is enabled).

Once you have the list of URIs, you can use `flexrep-delete` to remove these binary chunks.

```
xquery version "1.0-ml";
import module namespace flexrep =
"http://marklogic.com/xdmp/flexible-replication"
  at "/MarkLogic/flexrep.xqy";

let $delete :=
<flexrep:delete
xmlns:flexrep="http://marklogic.com/xdmp/flexible-replication">
<doc:uri xmlns:doc="xdmp:document-load">/content/foo.xml</doc:uri>
<flexrep:last-updated>2010-09-28T14:35:12.714-08:00</flexrep:last-upda
ted>
</flexrep:delete>
return flexrep:delete($delete)

(: Applies the specified delete element to /content/foo.xml.  This
effectively deletes the document from both the Master and Replica
databases. :)
```

Note: To use the`flexrep-delete` function you need the `flexrep-user` privilege.

4.0 Filtering Replicated Documents

Documents may be optionally filtered as part of the replication process by configuring a filter module for either outbound documents from the Master database or inbound documents to the Replica database. Filter modules are placed in the modules database for the replicated domain, and configured for either the Master or Replica database.

When a document is replicated, the document node, along with either an update or delete node, are sent by the Master to the Replicas. You can create filters to modify the contents of the document, update or delete node before it is sent to the Replicas (outbound filter) or when it is received by a Replica (inbound filter).

This chapter includes the following sections:

- Creating a Filter Module
- Configuring MarkLogic Server to use a Replication Filter Module
- Example Outbound Filter Modules
- Example Inbound Filter Modules
- Setting Outbound Filter Options

4.1 Creating a Filter Module

You can create replication filters to modify the document node, update node, or delete node before it is replicated to the target or after it is received by the target. If no document node follows the update node in the sequence, the document's root node will be removed on the Replica. If the filter returns an empty sequence, the framework will not replicate the document to the target.

A filter returns both an update and a document node in the case of a document update, or an update node only, in the case of a document delete. If a filter returns multiple update nodes, they will all be applied to the target. This could be used to break a replicated document apart into multiple documents on the target.

4.1.1 Outbound Filters

An outbound replication filter receives the following external variables as parameters:

```
declare variable $flexrep:uri as xs:string external;
declare variable $flexrep:target as element(flexrep:target) external;
declare variable $flexrep:update as element() external;
declare variable $flexrep:doc as document-node()? external;
```

The `$flexrep:target` variable contains the replication target configuration and the `$flexrep:doc` variable identifies the replicated document node. The `$flexrep:update` variable will be either a `flexrep:update` or a `flexrep:delete` node. Following is an example of each to illustrate the contained information (in the flexrep namespace).

A `flexrep:update` node looks like:

```
<flexrep:update
  xmlns:flexrep="http://marklogic.com/xdmp/flexible-replication">
    <doc:uri xmlns:doc="xdmp:document-load">
       /content/myDoc.xml
    </doc:uri>
    <flexrep:last-updated>
        2010-09-29T14:08:28.391-07:00
    </flexrep:last-updated>
    <doc:format xmlns:doc="xdmp:document-load">xml</doc:format>
    <flexrep:permissions>
      <flexrep:permission>
        <sec:role-name xmlns:sec="http://marklogic.com/xdmp/security">
            admin
        </sec:role-name>
        <sec:capability
          xmlns:sec="http://marklogic.com/xdmp/security">
            read
        </sec:capability>
      </flexrep:permission>
      <flexrep:permission>
        <sec:role-name xmlns:sec="http://marklogic.com/xdmp/security">
            admin
        </sec:role-name>
        <sec:capability
          xmlns:sec="http://marklogic.com/xdmp/security">
            update
        </sec:capability>
      </flexrep:permission>
    </flexrep:permissions>
    <doc:collections xmlns:doc="xdmp:document-load"/>
    <doc:quality xmlns:doc="xdmp:document-load">0</doc:quality>
    <flexrep:forests/>
    <prop:properties xmlns:prop="http://marklogic.com/xdmp/property"/>
</flexrep:update>
```

A `flexrep:delete` node looks like:

```
<flexrep:delete xmlns:flexrep=
  "http://marklogic.com/xdmp/flexible-replication">
    <doc:uri xmlns:doc="xdmp:document-load">
       /content/myDoc.xml
    </doc:uri>
    <flexrep:last-updated>
       2010-03-04T14:35:12.714-08:00
    </flexrep:last-updated>
</flexrep:delete>
```

4.1.2 Inbound Filters

An inbound replication filter receives the following external variables as parameters:

```
declare variable $dts as element(flexrep:domain-target-status) external;
declare variable $update as node()* external;
```

The `$update` sequence is a sequence of elements describing the replicated document. It could be a `flexrep:update` element followed by a document, a `flexrep:delete` element, or a series of these. The sequence may contain a mix of updates and deletes (the result of having passed through a target filter on the master that returned something more complicated than the original document).

The filter module should return a sequence that is derived from `$update`. For example, the same sequence but mapping the document URI in any `update` or `delete` element to a different directory. Another example might be to add a collection or document properties that track where the document was received from.

If an empty sequence is returned, then the replication is dropped and treated as successful.

If the replication was the result of a push, the module will run as the user that was used to log in to the flexible replication application server, but with the `flexrep:internal` role added. If the replication was the result of a pull, the module will run as whatever user did the pull (e.g. as configured on a scheduled task), and also with the `flexrep:internal` role.

If an error is thrown, the replication attempt will fail. If an empty sequence is returned, the replication attempt will become a successful nop.

4.2 Configuring MarkLogic Server to use a Replication Filter Module

This section describes:

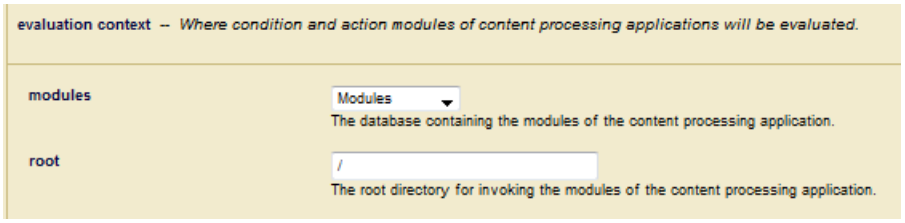
- Configuring a Master Database to Use an Outbound Filter
- Configuring a Replica Database to Use an Inbound Filter

4.2.1 Configuring a Master Database to Use an Outbound Filter

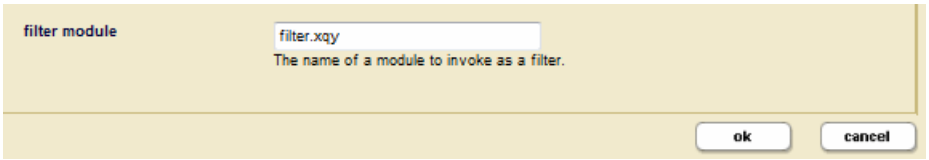
Once you have written an outbound filter module, such as those shown in "Example Outbound Filter Modules" on page 53, you can configure replication on the Master database to use the outbound replication filter module.

To configure MarkLogic Server to use a outbound replication filter module, you can call the `flexrep:configuration-target-set-filter-module` function or do the following in the Admin Interface:

1. Navigate to the Domain Definition page for the replicated domain, as described in "Defining Replicated Domains" on page 25. At the bottom of the page, specify the location of your replication filters. The default location for filters is the root directory in the modules database for the replicated domain:



2. Navigate to the Replication Target page, as described in "Configuring Push Replication" on page 29. At the bottom of the page, specify the name of the replication filter module to be used:



4.2.2 Configuring a Replica Database to Use an Inbound Filter

A target system can run a filter on any inbound replication operations (regardless of whether it's push or pull). Once you have written an inbound filter module, such as those shown in "Example Inbound Filter Modules" on page 64, you can create an inbound filter using `flexrep:inbound-filter-create` and load the filter into the database using `flexrep:inbound-filter-insert`. You will need to create a script with XQuery to access these built-in functions.

```
xquery version "1.0-ml";
import module namespace flexrep =
  "http://marklogic.com/xdmp/flexible-replication"
  at "/MarkLogic/flexrep.xqy";

flexrep:inbound-filter-insert(
  flexrep:inbound-filter-create(
    "/inbound-filter.xqy",
    <flexrep:filter-options xmlns="xdmp:eval">
      <modules>{xdmp:database("Modules")}</modules>
      <root>/</root>
    </flexrep:filter-options>))
```

4.3 Example Outbound Filter Modules

This section shows the following outbound filter examples:

- Adding a Collection
- Changing the Document Quality
- Adding Document Permissions
- Adding a Forest Name
- Changing the Document URI
- Changing a Document Element
- Prohibiting Replication on Select Documents

The first example, Adding a Collection, demonstrates how to use either an XQuery function or an XSL stylesheet to produce the same results. The remaining examples make use of XQuery functions, only.

> **Note:** Filter modules must be in the modules database for the replicated domain. You can either use `xdmp:document-insert` to insert the module into the modules database or specify '(file system)' for the modules database and place the module in the `/MarkLogic/Modules` directory.

4.3.1 Adding a Collection

This section shows two filters that add a collection to replicated documents. The first example is a filter that makes use of an XQuery function. The second is a filter that makes use of an XSL stylesheet. Both filters produce the same results.

The following filter defines an XQuery function that iterates through the elements of the update node, locates the `doc:collections` element, and inserts a `sec:uri` element with the value of `http://marklogic.com/flexrep/collection-two`:

```
xquery version "1.0-ml";

declare namespace flexrep =
  "http://marklogic.com/xdmp/flexible-replication";

declare namespace doc = "xdmp:document-load";

declare variable $flexrep:uri as xs:string external;
declare variable $flexrep:target as element(flexrep:target) external;
declare variable $flexrep:update as element() external;
declare variable $flexrep:doc as document-node()? external;

declare function local:add-my-collection(
  $update as element(flexrep:update))
  {
    element flexrep:update {
      $update/@*,
      for $n in $update/node()
      return
        typeswitch($n)
        case element(doc:collections)
          return element doc:collections {
            $n//sec:uri,
            element sec:uri
               { "http://marklogic.com/flexrep/collection-two" }
          }
        default return $n
    }
  };

(
  xdmp:log(fn:concat("Filtering ", $flexrep:uri)),
  typeswitch($flexrep:update)
  case element(flexrep:update)
    return (local:add-my-collection($flexrep:update), $flexrep:doc)
  case element(flexrep:delete)
    return $flexrep:update
  default
    return fn:error((), "FILTER-UNEXPECTED", ())
)
```

The following filter defines an XSLT stylesheet that creates a copy of the update node, locates the `doc:collections` element, and inserts a `sec:uri` element with the value of `http://marklogic.com/flexrep/collection-two`:

```
xquery version "1.0-ml";

declare namespace flexrep =
"http://marklogic.com/xdmp/flexible-replication";
declare namespace doc = "xdmp:document-load";
declare variable $flexrep:uri as xs:string external;
declare variable $flexrep:target as element(flexrep:target) external;
declare variable $flexrep:update as element() external;
declare variable $flexrep:doc as document-node()? external;

let $stylesheet :=
 <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:doc = "xdmp:document-load"
       xmlns:sec="http://marklogic.com/xdmp/security"
       version="2.0">
   <!-- Default recursive copy transform -->
   <xsl:template match="@*|node()">
     <xsl:copy>
       <xsl:apply-templates select="@*|node()"/>
     </xsl:copy>
   </xsl:template>

   <!-- Add my collection to the existing collections  -->
   <xsl:template match="doc:collections">
     <xsl:copy>
       <xsl:apply-templates select="node()"/>
       <sec:uri>http://marklogic.com/flexrep/collection-two</sec:uri>
     </xsl:copy>
   </xsl:template>
 </xsl:stylesheet>

return (
  xdmp:log(fn:concat("Filtering ", $flexrep:uri)),
  typeswitch($flexrep:update)
  case element(flexrep:update)
    return (
       xdmp:xslt-eval($stylesheet, $flexrep:update)/flexrep:update,
       $flexrep:doc)
  case element(flexrep:delete)
    return $flexrep:update
  default
    return fn:error((), "FILTER-UNEXPECTED", ())
)
```

Either of the above filters will convert the update node to:

```
<flexrep:update xmlns:flexrep=
  "http://marklogic.com/xdmp/flexible-replication">
    ........
    <doc:collections xmlns:doc="xdmp:document-load">
       <sec:uri xmlns:sec="http://marklogic.com/xdmp/security">
          http://marklogic.com/flexrep/collection-two
       </sec:uri>
    </doc:collections>
    ........
    <prop:properties xmlns:prop="http://marklogic.com/xdmp/property"/>
</flexrep:update>
```

4.3.2 Changing the Document Quality

The following filter changes the quality of documents to 3. This is done by iterating through the elements of the update node, locating the `doc:document-quality` element, and resetting its value to 3.

```
xquery version "1.0-ml";

declare namespace flexrep =
  "http://marklogic.com/xdmp/flexible-replication";

declare namespace doc = "xdmp:document-load";

declare variable $flexrep:uri as xs:string external;
declare variable $flexrep:target as element(flexrep:target) external;
declare variable $flexrep:update as element() external;
declare variable $flexrep:doc as document-node()? external;

declare function local:change-quality($update as
element(flexrep:update))
{
  element flexrep:update {
    $update/@*,
    for $n in $update/node()
    return
      typeswitch($n)
      case element(doc:quality)
        return element doc:quality   { 3 }
      default return $n
  }
};
```

```
(
  xdmp:log(fn:concat("Filtering ", $flexrep:uri)),
  typeswitch($flexrep:update)
  case element(flexrep:update)
    return (local:change-quality($flexrep:update), $flexrep:doc)
  case element(flexrep:delete)
    return $flexrep:update
  default
    return fn:error((), "FILTER-UNEXPECTED", ())
)
```

This will convert the update node to:

```
<flexrep:update xmlns:flexrep=
  "http://marklogic.com/xdmp/flexible-replication">
    ........
    <doc:quality xmlns:doc="xdmp:document-load">3</doc:quality>
    ........
</flexrep:update>
```

4.3.3 Adding Document Permissions

The following filter adds read and update permission for users with the "developer" role to documents. This is done by iterating through the elements of the update node, locating the `doc:permissions` element, and inserting `sec:permission` elements containing `sec:capability` and `sec:role-id` elements that establish read and update permissions for "developer" users.

```
xquery version "1.0-ml";

declare namespace flexrep =
  "http://marklogic.com/xdmp/flexible-replication";

declare namespace doc = "xdmp:document-load";

declare variable $flexrep:uri as xs:string external;
declare variable $flexrep:target as element(flexrep:target) external;
declare variable $flexrep:update as element() external;
declare variable $flexrep:doc as document-node()? external;

declare function local:change-permission(
  $update as element(flexrep:update))
{
  element flexrep:update {
    $update/@*,
    for $n in $update/node()
    return
      typeswitch($n)
      case element(flexrep:permissions)
        return element flexrep:permissions {
          $n/flexrep:permission ,
          element flexrep:permission  {
            element sec:role-name { "developer" },
            element sec:capability  { "read" }
          },
          element flexrep:permission  {
            element sec:role-name { "developer" },
            element sec:capability  { "update" }
          }
        }
      default return $n
  }
};

(
  xdmp:log(fn:concat("Filtering ", $flexrep:uri)),
  typeswitch($flexrep:update)
  case element(flexrep:update)
    return (local:change-permission($flexrep:update), $flexrep:doc)
  case element(flexrep:delete)
    return $flexrep:update
  default
    return fn:error((), "FILTER-UNEXPECTED", ())
)
```

This will convert the update node to:

```
<flexrep:update xmlns:flexrep=
  "http://marklogic.com/xdmp/flexible-replication">
    ........
    <flexrep:permissions>
      <flexrep:permission>
        <sec:role-name xmlns:sec="http://marklogic.com/xdmp/security">
            admin
        </sec:role-name>
        <sec:capability
          xmlns:sec="http://marklogic.com/xdmp/security">
            read
        </sec:capability>
      </flexrep:permission>
      <flexrep:permission>
        <sec:role-name xmlns:sec="http://marklogic.com/xdmp/security">
            admin
        </sec:role-name>
        <sec:capability
          xmlns:sec="http://marklogic.com/xdmp/security">
            update
        </sec:capability>
      </flexrep:permission>
    </flexrep:permissions>
    <flexrep:permissions>
      <flexrep:permission>
        <sec:role-name xmlns:sec="http://marklogic.com/xdmp/security">
            developer
        </sec:role-name>
        <sec:capability
          xmlns:sec="http://marklogic.com/xdmp/security">
            read
        </sec:capability>
      </flexrep:permission>
      <flexrep:permission>
        <sec:role-name xmlns:sec="http://marklogic.com/xdmp/security">
            developer
        </sec:role-name>
        <sec:capability
          xmlns:sec="http://marklogic.com/xdmp/security">
            update
        </sec:capability>
      </flexrep:permission>
    </flexrep:permissions>
    ........
</flexrep:update>
```

4.3.4 Adding a Forest Name

The following filter adds the forest name, `myFavoriteForest`, to the update node. MarkLogic Server maps the forest name to its ID and passes it to the `xdmp:document-insert` function to insert the document into the named forest. If you specify multiple forests, MarkLogic Server will insert the document into one of them. See the documentation for the `xdmp:document-insert` function for more information.

```
xquery version "1.0-ml";

declare namespace flexrep =
  "http://marklogic.com/xdmp/flexible-replication";

declare namespace doc = "xdmp:document-load";

declare variable $flexrep:uri as xs:string external;
declare variable $flexrep:target as element(flexrep:target) external;
declare variable $flexrep:update as element() external;
declare variable $flexrep:doc as document-node()? external;

declare function local:add-forest(
  $update as element(flexrep:update))
  {
    element flexrep:update {
      $update/@*,
      for $n in $update/node()
      return
        typeswitch($n)
        case element(flexrep:forests)
          return element flexrep:forests {
            element flexrep:forest { "myFavoriteForest" }
          }
        default return $n
    }
  };

(
  xdmp:log(fn:concat("Filtering ", $flexrep:uri)),
  typeswitch($flexrep:update)
  case element(flexrep:update)
    return (local:add-forest($flexrep:update), $flexrep:doc)
  case element(flexrep:delete)
    return $flexrep:update
  default
    return fn:error((), "FILTER-UNEXPECTED", ())
)
```

This will convert the update node to:

```
<flexrep:update xmlns:flexrep=
  "http://marklogic.com/xdmp/flexible-replication">
    ........
    <flexrep:forests>
        <flexrep:forest>myFavoriteForest</flexrep:forest>
    </flexrep:forests>
    ........
</flexrep:update>
```

4.3.5 Changing the Document URI

The following filter adds `/replicated/` to the front of each document URI. This is done by iterating through the elements of the update node, locating the `doc:uri` element and adding `/replicated/` to its value.

```
xquery version "1.0-ml";

declare namespace flexrep =
  "http://marklogic.com/xdmp/flexible-replication";

declare namespace doc = "xdmp:document-load";

declare variable $flexrep:uri as xs:string external;
declare variable $flexrep:target as element(flexrep:target) external;
declare variable $flexrep:update as element() external;
declare variable $flexrep:doc as document-node()? external;

declare function local:change-uri($update as element(flexrep:update))
{
  element flexrep:update {
    $update/@*,
    for $n in $update/node()
    return
      typeswitch($n)
      case element(doc:uri)
        return element doc:uri {
          fn:concat("/replicated", $flexrep:uri)
        }
      default return $n
  }
};

(
  xdmp:log(fn:concat("Filtering ", $flexrep:uri)),
  typeswitch($flexrep:update)
  case element(flexrep:update)
     return (local:change-uri($flexrep:update), $flexrep:doc)
  case element(flexrep:delete)
    return $flexrep:update
  default
```

```
    return fn:error((), "FILTER-UNEXPECTED", ())
)
```

This will convert the update node to:

```
<flexrep:update xmlns:flexrep=
  "http://marklogic.com/xdmp/flexible-replication">
    <doc:uri xmlns:doc="xdmp:document-load">
       <doc:uri>/replicated//content/foo.xml</doc:uri>
    </doc:uri>
    ........
</flexrep:update>
```

4.3.6 Changing a Document Element

The following filter changes all <PARA> elements in replicated documents to <PARAGRAPH> and leaves all of the other elements in the documents unchanged. This is done by iterating through the elements of the document node, locating each PARA element and converting its value to PARAGRAPH.

```
xquery version "1.0-ml";

declare namespace flexrep =
   "http://marklogic.com/xdmp/flexible-replication";
declare namespace doc = "xdmp:document-load";

declare variable $flexrep:uri as xs:string external;
declare variable $flexrep:target as element(flexrep:target) external;
declare variable $flexrep:update as element() external;
declare variable $flexrep:doc as document-node()* external;

(: recursive typeswitch function to transform the element :)
declare function local:change-element($x as node()*) as node()*
{
for $n in $x return
  typeswitch ($n)
    case document-node() return
       document {local:change-element($n/node())}
    case text() return $n
    case element (PARA)
       return <PARAGRAPH>{local:change-element($n/node())}</PARAGRAPH>
    default return element {
       fn:node-name($n)} {$n/@*, local:change-element($n/node())}
};

  xdmp:log(fn:concat("Filtering ", $flexrep:uri)),
  typeswitch($flexrep:update)
  case element(flexrep:update)
    return ($flexrep:update, local:change-element($flexrep:doc) )
  case element(flexrep:delete)
    return $flexrep:update
  default return fn:error((), "FILTER-UNEXPECTED", ())
```

4.3.7 Prohibiting Replication on Select Documents

Should you want to prohibit replication on certain documents in a replicated domain, you can add a property to the document that flags it as a no-replicate document. You can then write a filter that checks for the property and determines whether or not to replicate the document, depending on the presence or value of the property.

For example, if the `/content` directory is in a replicated domain, but you don't want to replicate the document, `/content/foo.xml`, you can assign the document a `replicate` property with a value of `no`.

```
xquery version "1.0-ml";

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

xdmp:document-add-properties(
    "/content/foo.xml",
    (<prop:replicate>no</prop:replicate>) )
```

You can write a filter that looks for the `replicate` property on each document. If the property is missing or it is some value other than `no`, then the document is replicated. If the property is set on the document and its value is `no`, then the document will not be replicated.

```
xquery version "1.0-ml";

declare namespace flexrep =
  "http://marklogic.com/xdmp/flexible-replication";

declare namespace doc = "xdmp:document-load";

declare variable $flexrep:uri as xs:string external;
declare variable $flexrep:target as element(flexrep:target) external;
declare variable $flexrep:update as element() external;
declare variable $flexrep:doc as document-node()? external;

(
  xdmp:log(fn:concat("Filtering ", $flexrep:uri)),
  typeswitch($flexrep:update)
  case element(flexrep:update)
    return
    if (xdmp:document-properties($flexrep:uri)//prop:replicate = "no")
    then ()
    else ($flexrep:update, $flexrep:doc)
  case element(flexrep:delete)
    return $flexrep:update
  default
    return fn:error((), "FILTER-UNEXPECTED", ())
)
```

4.4 Example Inbound Filter Modules

This section shows a few examples of inbound filters that do the same operations as two of the outbound filters shown above for the Master database, but these example filters are inbound filters on the Replica database.

This section shows the following inbound filter examples:

- Adding a Collection
- Changing the Document URI

4.4.1 Adding a Collection

This section an inbound filter that add a collection to replicated documents in the same manner as the outbound filter described in “Adding a Collection” on page 54.

The following filter defines an XQuery function that iterates through the elements of the update node, locates the `doc:collections` element, and inserts a `sec:uri` element with the value of `http://marklogic.com/flexrep/collection-A`:

```
xquery version "1.0-ml";

import module
  namespace flexrep = "http://marklogic.com/xdmp/flexible-replication"
     at "/MarkLogic/flexrep.xqy";

declare namespace doc = "xdmp:document-load";

declare variable $dts as element(flexrep:domain-target-status)
external;
declare variable $update as node()* external;

declare function local:add-my-collection(
  $update as element(flexrep:update))
  {
    element flexrep:update {
      $update/@*,
      for $n in $update/node()
      return
        typeswitch($n)
        case element(doc:collections)
          return element doc:collections {
            $n//sec:uri,
            element sec:uri
               { "http://marklogic.com/collection-A" }
          }
      default return $n
    }
  };
```

```
for $u in $update
return
  typeswitch ($u)
     case element(flexrep:update)  return
        local:add-my-collection($u)
     default
      return $u
```

4.4.2 Changing the Document URI

This section an inbound filter that changes the URI of the replicated documents in the same manner as the outbound filter described in “Changing the Document URI” on page 61.

The following filter adds `/replicated/` to the front of each document URI. This is done by iterating through the elements of the update node, locating the `doc:uri` element and adding `/replicated/` to its value.

```
xquery version "1.0-ml";

import module
  namespace flexrep = "http://marklogic.com/xdmp/flexible-replication"
     at "/MarkLogic/flexrep.xqy";

declare namespace doc = "xdmp:document-load";

declare variable $dts as element(flexrep:domain-target-status)
external;
declare variable $update as node()* external;

declare function local:change-uri(
  $update as element(flexrep:update))
  {
    element flexrep:update {
      $update/@*,
      for $n in $update/node()
      return
       typeswitch($n)
       case element(doc:uri)
         return element doc:uri {
             fn:concat("/replicated", $n)
         }
         default return $n
    }
   };

for $u in $update
return
   typeswitch ($u)
    case element(flexrep:update)  return
      local:change-uri($u)
    default
      return $u
```

4.5 Setting Outbound Filter Options

You can use the `flexrep:configuration-target-set-filter-options` function to change the evaluation parameters used to invoke an outbound filter. For example, you can specify filter options that determine which user can invoke the outbound filter or on what database the filter is to be invoked. The options specified by the `flexrep:configuration-target-set-filter-options` function are passed to the `xdmp:invoke` function of the filter module, so any of the options you would specify in the `xdmp:eval` function are recognized.

Outbound filter options cannot be set in by the Admin Interface. You must set outbound filter options programmatically using the flexrep API. The flexrep API is described in the Scripting Flexible Replication Configuration chapter in the *Scripting Administrative Tasks Guide* and the reference documentation for each function is in the *MarkLogic XQuery and XSLT Function Reference*.

For example, you can write a module that specifies that the outbound filter can only be invoked by the user John:

```
xquery version "1.0-ml";

import module namespace flexrep =
   "http://marklogic.com/xdmp/flexible-replication"
      at "/MarkLogic/flexrep.xqy";

import module namespace trgr="http://marklogic.com/xdmp/triggers"
   at "/MarkLogic/triggers.xqy";

let $trigger := trgr:get-trigger("cpf:update Replicated Content")

(: Obtain the id of the replicated CPF domain from the
   Triggers database. :)

let $domain := xdmp:eval(
    'xquery version "1.0-ml";
    import module namespace dom = "http://marklogic.com/cpf/domains"
      at "/MarkLogic/cpf/domains.xqy";
    fn:data(dom:get( "Replicated Content" )//dom:domain-id)',
    (),
    <options xmlns="xdmp:eval">
      <database>{xdmp:database("MyTriggers")}</database>
    </options>)

(: Obtain the replication configuration. :)
let $cfg := flexrep:configuration-get($domain, fn:true())

(: Obtain the ID of the replication target. :)
let $target-id := flexrep:configuration-target-get-id($cfg, "Replica")
```

```
(: Define a flexrep:filter-options element. :)
let $filter-opts :=
  <flexrep:filter-options>
    <user-id xmlns="xdmp:eval">{xdmp:user("John")}</user-id>
  </flexrep:filter-options>

(: Set the flexrep:filter-options element. :)
let $cfg :=
  flexrep:configuration-target-set-filter-options(
    $cfg,
    $target-id,
    $filter-opts)

(: Save the new replication configuration. :)
return flexrep:configuration-insert($cfg)
```

5.0 Checking Replication Status

This chapter describes how to check replication status for a Master database, a domain, and a target. This chapter includes the following sections:

- Special Circumstances that Impact Replication
- Checking the Replication Status of a Master Database
- Checking the Replication Status of a Domain
- Checking the Replication Status of a Target

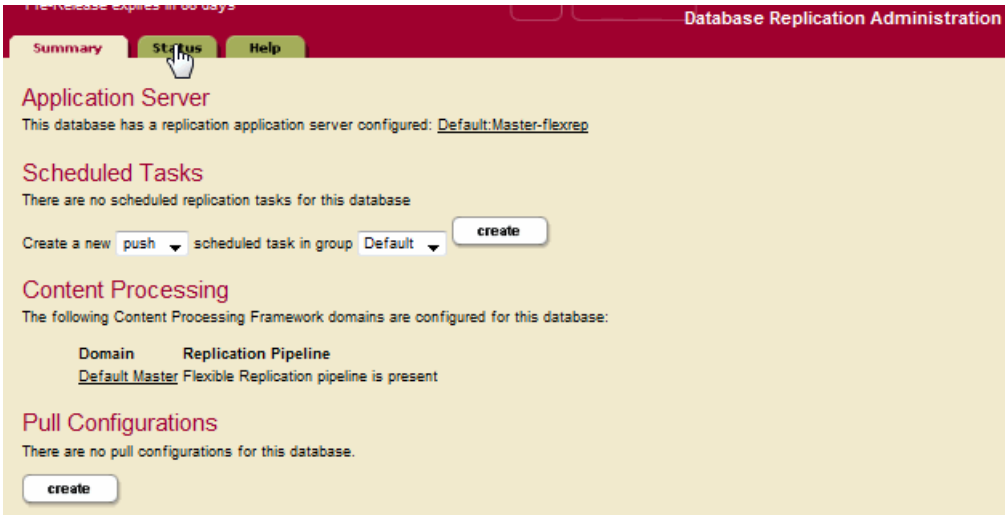
5.1 Special Circumstances that Impact Replication

If the task server queue is more than half full, the Master Server will not push documents to the Replica and will instead leave it for the scheduled push task. This is to help avoid overflowing the task queue when ingesting at a high rate.

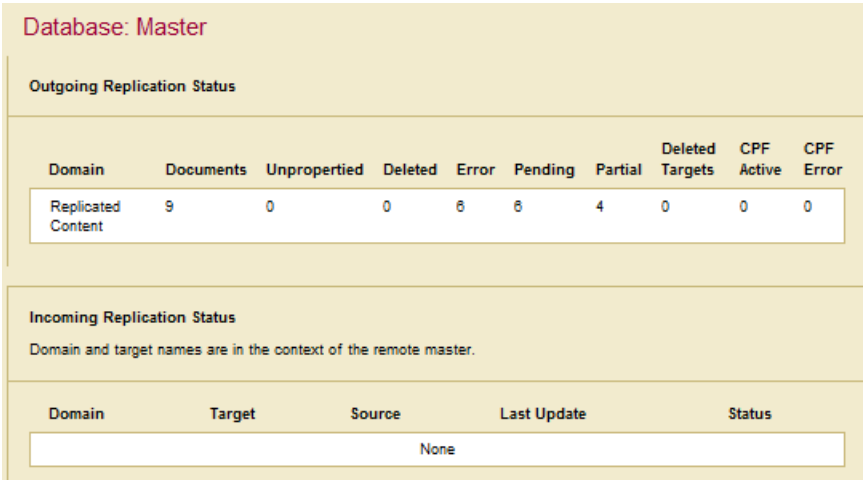
See Examining the Host and Task Server Status Pages For Tasks in the Queue in the *Content Processing Framework Guide* for information on checking the status of the task server.

5.2 Checking the Replication Status of a Master Database

You can check the replication status of a Master database by selecting the Flexible Replication icon under the Master database in the left tree menu and clicking on the Status tab on the Flexible Replication Administration page.



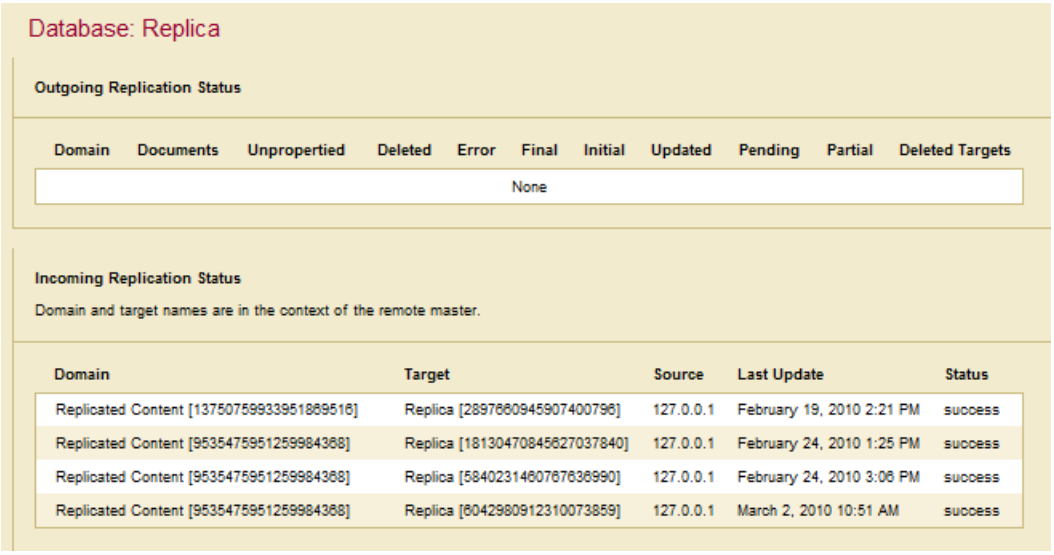
The Outgoing Replication Status indicates the replication status for each replicated domain from the perspective of the Master database.



Field	Description
Domain	The name of the replicated domain.
Documents	The number of documents in the domain on the Master.
Unpropertied	The number of documents in the replicated domain without a replication status. This generally means that these documents existed before the replication domain was configured and have not yet been replicated.
Deleted	The number of replicated documents that were deleted, but the delete has not yet been replicated.
Error	The number of replication errors.
Pending	The number of documents in the replicated domain that have not yet been replicated to the Replicas.
Partial	The number of documents in the replicated domain that have been sent to one or more Replicas, but are pending or have errors for at least one Replica.
Deleted Targets	The number of targets that were deleted from the replication configuration for this domain.

Field	Description
CPF Active	The number of documents that are currently in the CPF pipeline.
CPF Error	The number of documents that encountered an error while in the CPF pipeline.

The Incoming Replication Status indicates the replication status for each replicated domain from the perspective of the Replica database.

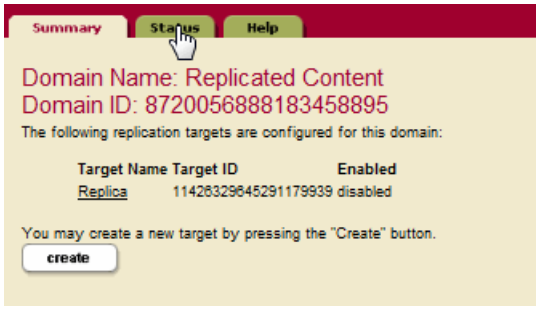


Field	Description
Domain	The name of the replicated domain.
Target	The name and id of the replication target.
Source	The IP address of the Master host server.
Last Update	The date and time of the last replicated update.
Status	The status of the last replicated update

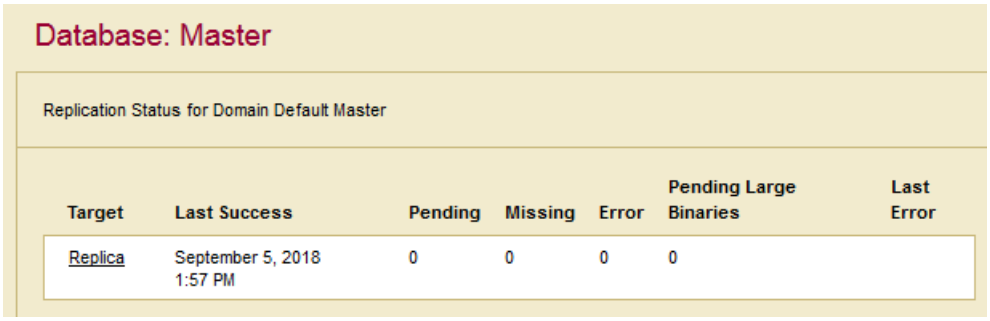
5.3 Checking the Replication Status of a Domain

You can check the replication status of a replicated domain by the following procedure:

1. Select the domain icon under Flexible Replication in the left tree menu and click on the Status tab on the Flexible Replication Domain Administration page:



2. The page containing the replication status of the domain appears:



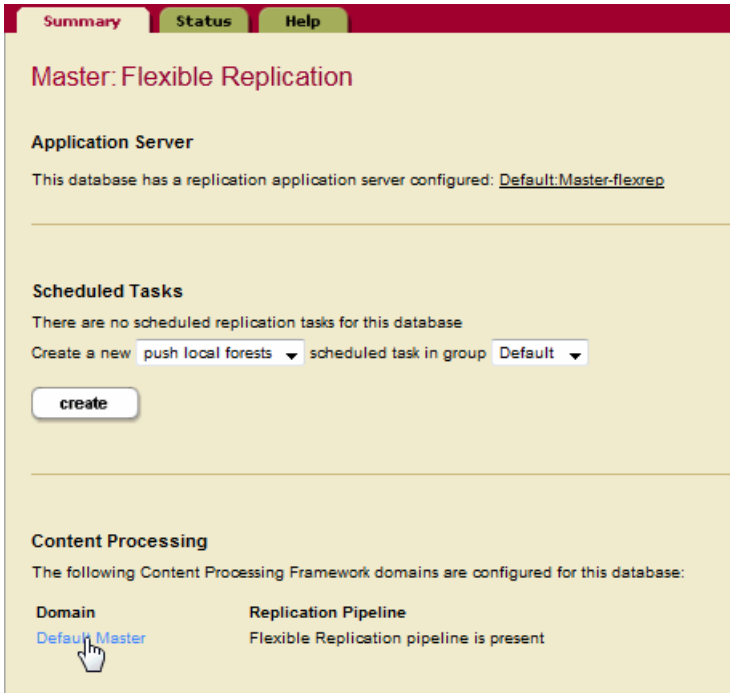
Field	Description
Target	The name of the replication target.
Last Success	The date and time of the last successful update to the target.
Pending	The number of documents in the replicated domain that have not yet been replicated to the target.
Missing	The number of documents missing replication properties. For example, documents that were already in the replicated domain before replication was configured.
Error	The number of replication errors.
Pending Large Binaries	The number of large binaries in the replicated domain that have not yet been replicated to the target. See “Interrupted replication” on page 48 for details.

Field	Description
Last Error	The last replication error.

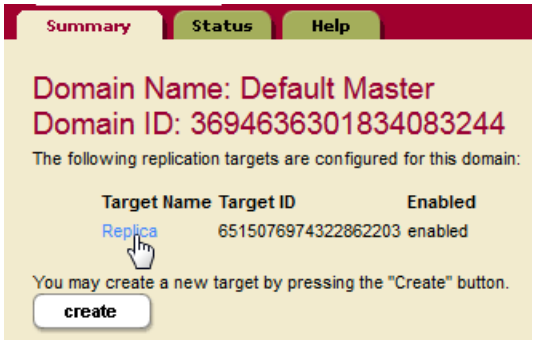
5.4 Checking the Replication Status of a Target

You can check the replication status of a replication target by the following procedure:

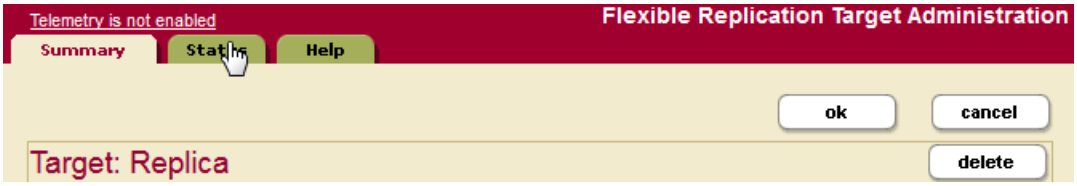
1. Select the domain icon under Flexible Replication in the left tree menu, selecting the Target Name from the Domain Administration page, and clicking on the Status tab on the Flexible Replication Target Administration page.



2. Click the Target Name on the Flexible Replication Domain Administration page:



3. Click the Status tab on the Flexible Replication Target Administration page:



4. The page containing the replication status of the target appears. The overall status of replicated documents to the target appears in the top field. The documents that failed to replicate are listed in the second field, along with last and next replication retry times. You can click Retry to retry replication on selected documents or Retry All Documents to retry replication on all of the documents in the list.



Field	Description
Target	The name of the replication target.
Last Success	The date and time of the last successful update to the target.
Pending	The number of documents in the replicated domain that have not yet been replicated to the target.
Missing	The number of documents missing replication properties. For example, documents that were already in the replicated domain before replication was configured.
Error	The number of replication errors.

Field	Description
Pending Large Binaries	The number of large binaries in the replicated domain that have not yet been replicated to the target. See “Interrupted replication” on page 48 for details.
Document	The document that failed to replicate to the Replica.
Last Try	The last attempt to replicate the failed document.
Next Try	The next attempt to replicate the failed document.
Tries	The number of times MarkLogic Server attempted to replicate the failed document.
Status	The error generated on the last attempt to replicate the failed document.
Retry	Check box to retry replicating the failed document.

Check the `Retry all # error documents` box to retry replicating all documents that were not replicated due to errors, or `Retry all documents in the domain` to retry replicating all documents in the domain.

You can query the status of a specific document by entering its URI in the Check URI field and clicking Query. The status of the document, whether it replicated successfully or returned an error is displayed and highlighted in the above field.



6.0 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 technical questions, we encourage you to ask your question on Stack Overflow.

7.0 Copyright

MarkLogic Server 9.0 and supporting products.
Last updated: November 18, 2019

COPYRIGHT

© 2019 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 for your version of MarkLogic.