

# *Global Format Digital Registry (GDFR)*

## **Node Interoperability Test Design**

*By: Andreas Stanescu*

Status: Draft

Date Last Modified: March 3, 2008

---

### **1. Introduction**

#### **1.1 Purpose**

This document describes the testing tasks to be performed on the GDFR nodes to determine the level of interoperability and the accuracy of data transmission from source nodes to mirror nodes.

The scope of testing is limited to interoperability of nodes: data transmission, accuracy, authentication, access control. This test plan will not add testing tasks for actions performed in a single node environment (add records, update records, search records, retrieve records), nor will demonstrate any performance benchmarks. However, where these actions are necessary for interoperability testing, they will be used to create the necessary pre-conditions for the testing tasks to complete. For example, to demonstrate accurate data transmission from node to another, records must first be added into the source node then searched and retrieved by a mirror node, so the testing task will require the operator to add (or update) a certain number of records.

#### **1.2 Definitions**

Source node – the central GDFR instance where records are authored (added and changed).

Mirror node – the satellite GDFR nodes which duplicate the records held in the source node for safe-keeping.

### **2. Test Design Strategy**

#### **2.1 General Approach**

The testing tasks listed in this document will be performed manually, by an individual trained to use the user interface of the GDFR software. No automated testing harness is provided.

Verification will be performed against input data – for example, when verifying records duplicated in a mirror node, all of the data entered in the version of record present in the source node must be identical to the one replicated in the mirror node. The replication augments the record with very specific information and so the replicated record isn't an exact duplicate, but one preserving all the original data, plus information documenting the export/import activity.

Test cases were generated from requirements (use cases) as well as a model of expected system behavior.

## **2.2 Test Preparation**

### **2.2.1 Hardware preparation**

Installation requirements are detailed in the release notes:

[https://collaborate.oclc.org/wiki/gdfr/index.php/Draft:2.0\\_Release#Before\\_You\\_Install\\_-\\_Installation\\_Requirements](https://collaborate.oclc.org/wiki/gdfr/index.php/Draft:2.0_Release#Before_You_Install_-_Installation_Requirements)

### **2.2.2 Additional preparation**

At least two nodes are necessary for this testing. One is the source node, the instance where GDFR users maintain registry records. The second is the mirror node, which simply replicates the collections and records held in the source node.

Most tests will be performed in the mirror node. However, some tests will be performed against both, in cases where verification with the originals is necessary.

### **2.2.3 Data preparation**

#### *1. Installation of GDFR collections*

The GDFR software contains 3 initialization scripts, which must all be executed in both source and mirror node. The scripts perform the following tasks:

- create and initialize the Collections collection, and all the system collections (Schemas, Templates, XHTML, XSL)
- create and initialize the access control collections (Users, Roles, Features)
- create and initialize the GDFR collections (all 13 of them)

See the installation and configuration directions for more information.  
[https://collaborate.oclc.org/wiki/gdfr/index.php/Draft:2.0\\_Release#Installation\\_Instructions](https://collaborate.oclc.org/wiki/gdfr/index.php/Draft:2.0_Release#Installation_Instructions) .

## 2. *Authorizations*

Three roles are automatically created in each instance.

- anonymous (for access to public resource)
- GDFR users (for access to GDFR collections and records)
- GDFR admin (for access to system collections)

Also, a default user is added to each instance. The installation instructions clearly point out that this user account should be used to create appropriate users, each granted separate roles, and then the default user account should be removed or disabled to ensure that each instance correctly maintains the approved list of users.

## 3. *Test data*

Test data will be provided in an installation package.

### **2.3 Location of Problem Reports (optional)**

All problem reports will be entered and tracked in the Bugzilla instance set up at Harvard University Libraries for this purpose. It can be accessed at the following location:

<http://bugz.hul.harvard.edu/bugzilla/>

### **2.4 Organization (optional)**

- *Define the order of tests or how functionality will be grouped into scripts*
- *Define new scripts and designate which scripts will require changes*
- *Define offline jobs and systems or subsystems involved in the test*

## 2.5 Features to be Tested

Feature	Test Case Identifier (optional)	Test Script (optional)	Responsibility (Role)
<i>Replicate Format collection.</i>	3.1.1		<i>GDFR-user</i>
<i>Search Formats collection for all records and verify list is complete.</i>	3.1.2		<i>GDFR-user</i>
<i>In the mirror node, find and attempt to edit a Format record.</i>	3.1.3		<i>GDFR-user</i>
<i>Verify updated record is correctly replicated.</i>	3.1.4		<i>GDFR-user</i>
<i>Repeat 3.1.1 to verify no-change</i>	3.1.5		<i>GDFR-user</i>
<i>Search mirrored Format collection by provenance information.</i>	3.1.6		<i>GDFR-user</i>
<i>Search mirrored Format collection by original creator.</i>	3.1.7		<i>GDFR-user</i>

Repeat this pattern for 4 additional collections:

- Agents – use the numbering pattern 3.2.x
- Hardware – use the numbering pattern 3.3.x
- Software – use the numbering pattern 3.4.x

Since the process and software is identical for all collections, it is unnecessary to continue for all collections. However, Files collection contains binary data (not XML records) so it would be valuable to perform further testing for the following collections:

- FileDescriptors – use the numbering pattern 3.5.x
- Files – use the numbering pattern 3.6.x

Feature	Test Case Identifier (optional)	Test Script (optional)	Responsibility (Role)
<i>Synchronize GDFR collections</i>	3.7.1		<i>GDFR-user</i>

### 2.5.1 Features to be Tested for access control restrictions

<b>Feature</b>	<b>Test Case Identifier (optional)</b>	<b>Test Script (optional)</b>	<b>Responsibility (Role)</b>
<i>Attempt to replicate Features collection to verify no access.</i>	3.10.1		<i>GDFR-admin</i>
<i>Attempt to replicate Roles collection to verify no access</i>	3.10.2		<i>GDFR-admin</i>
<i>Attempt to replicate UserAccounts collection to verify no access</i>	3.10.3		<i>GDFR-admin</i>

## 2.6 Test Results

The test results will be kept on the wiki (follow the link to the wiki from the project page: <http://formatregistry.org/> ). The template above will be used to iterate through the tests multiple times throughout the testing period. The errors will be logged in Bugzilla.

Each test case execution will contain the following data:

- description of test run and functionality tested
- software version tested
- date test was executed
- who executed the test
- status (pass/fail)
- the problem reports and identifier in Bugzilla.

### 3. Test Cases

#### Replicate Format collection.

<b>Test Case ID</b>	<b>3.1.1</b>
<b>Description</b>	Replicate all records from the Format collection hosted by the source node into a corresponding collection in the mirror node.
<b>Actors</b>	GDFR User
<b>Pre-conditions</b>	The user is authenticated.
	The source node contains at least one record. One can be created by using the online form pages, or by POST'ing a record to the Atom Publishing Protocol (atompub) service.
<b>Primary functional path</b>	Execute the search service on the source node and note the records found: <SOURCE_node_base_url>/webservices/registry/search/Formats To return all the existing records, enter "*" in the value field for dc.identifier index.
	Execute the importSRU service on the mirror node: <MIRROR_node_base_url>/webservices/registry/importSRU/Formats Append the service arguments: baseUrl=<SOURCE_node_base_url>/webservices/registry/importSRU/Formats since=1900
	Make a note of the list of the records returned by the importSRU service. It must match the list of records initially found during the search in step #1.
<b>Verification</b>	Search the mirror node, using exactly the same query as in primary functional path step #1 above and note the records found: <MIRROR_node_base_url>/webservices/registry/search/Formats To return all the existing records, enter "*" in the value field for dc.identifier index.
	The list of records found must match the list of records returned by the search in the source node AND it must also match the list of records returned by the importSRU service.
	Retrieve the XML content of the record in each instance, source and mirror, and verify that the entire data set has been replicated.
	Retrieve the XML adminData of the record in each instance, source and mirror, and verify that the resourceID, versionID, creator and all other fields are identical.
	Verify that only the mirror version contains a provenance section, showing where the record originated and that information correctly describes the source node.

### Search Formats collection for records.

<b>Test Case ID</b>	3.1.2
<b>Description</b>	Execute various searches against the Formats collection in source and mirror nodes and verify that the results are identical.
<b>Actors</b>	Public
<b>Pre-conditions</b>	The source node contains at least one record.
<b>Primary functional path</b>	Execute the search service on BOTH the source node and the mirror and note the records found: <SOURCE_node_base_url>/webservices/registry/search/Formats <MIRROR_node_base_url>/webservices/registry/search/Formats
	Enter "*" in the value field for dc.identifier index to return all the existing records. Match the resourceID of a record and use "exact" as the operand. Match the creator userid of a record and use "exact" as the operand.
	Make a note of the list of the records returned by each search.
<b>Verification</b>	For each search, verify that the records returned by the search against the source node are identical to the records returned by the search against the mirror node.

### Attempt to edit a Format record in the mirror node.

<b>Test Case ID</b>	3.1.3
<b>Description</b>	Find and attempt to update a record in the mirror node.
<b>Actors</b>	GDFR User
<b>Pre-conditions</b>	The user is authenticated.
	The source node contains at least one record.
<b>Primary functional path</b>	Execute the search service on the mirror node: <MIRROR_node_base_url>/webservices/registry/search/Formats
	Match the resourceID of a record and use "exact" as the operand.
	Select the returned record and follow "Edit this record" link.
	The URL <b>*must*</b> link to formatregistry.org and not the mirror node.
	Manually change the URL in the browser and replace the base URL of the source (formatregistry.org) with the mirror.
	The edit page <b>*must not*</b> be displayed and an editing error should be shown.
	Using CURL or another testing tool, attempt to post an update to the "formatProcessor" service.
	The record <b>*must not*</b> be updated and an editing error should be shown.
	Using CURL or another testing tool, attempt to PUT an update to the "atompub" service.
	The record <b>*must not*</b> be updated and an HTTP error should be returned.
<b>Verification</b>	n/a

## Verify updated source record is correctly replicated.

<b>Test Case ID</b>	<b>3.1.4</b>
<b>Description</b>	Edit a Format record in the source node and replicate the Formats collection in the mirror node.
<b>Actors</b>	GDFR User
<b>Pre-conditions</b>	The user is authenticated.
	The source node contains at least one record.
<b>Primary functional path</b>	Execute the search service on the source node: <SOURCE_node_base_url>/webservices/registry/search/Formats
	Match the resourceID of a record and use "exact" as the operand.
	Select the returned record and follow "Edit this record" link.
	Change some data in the record and submit the change. The system must successfully update the record and inform the user as such.
	Invoke the importSRU service on the mirror node: <MIRROR_node_base_url>/webservices/registry/importSRU/Formats  Append the service arguments: baseURL=<SOURCE_node_base_url>/webservices/registry/importSRU/Formats since=1900
	Note in the response the identifier of the updated record.
<b>Verification</b>	Execute the search service on the source node: <SOURCE_node_base_url>/webservices/registry/search/Formats
	Match the resourceID of the updated record and use "exact" as the operand.
	Retrieve the XML content of the record in each instance, source and mirror, and verify that the entire data set has been replicated.
	Retrieve the XML adminData of the record in each instance, source and mirror, and verify that the resourceID, versionID, creator and all other fields are identical.
	Verify that only the mirror version contains a provenance section, showing where the record originated and that information correctly describes the source node.
	Execute the history service on this record on source and mirror: <SOURCE_node_base_url>/webservices/registry/history/Formats/<resourceID> <MIRROR_node_base_url>/webservices/registry/history/Formats/<resourceID>  Verify that in both source and mirror that the record has a current (active) version and an inactive version. Verify that the versionID of the inactive versions are identical.

**Repeat collection replication (3.1.1) to verify no-change.**

<b>Test Case ID</b>	<b>3.1.5</b>
<b>Description</b>	Repeated replication attempts must not add or change any records in the mirror node.
<b>Actors</b>	GDFR User
<b>Pre-conditions</b>	The user is authenticated.
	The source node contains at least one record.
	The collection has already been replicated in the mirror. To do that: Execute the search service on the source and the mirror nodes: <SOURCE_node_base_url>/webservices/registry/search/Formats <MIRROR_node_base_url>/webservices/registry/search/Formats Enter "*" in the value field for dc.identifier index to return all the existing records. Verify that the list of records is identical in both nodes.
<b>Primary functional path</b>	Execute the importSRU service on the mirror node: <MIRROR_node_base_url>/webservices/registry/importSRU/Formats Append the service arguments: baseUrl=<SOURCE_node_base_url>/webservices/registry/importSRU/Formats since=1900
	Make a note of the list of the records returned by the importSRU service – the list must contain only exceptions, duplicate records found or records too old (if histories exist), and none should have been imported.
<b>Verification</b>	Execute the searches described in pre-conditions above.
	Verify that the list of records returned by the mirror matches exactly the one returned earlier and that it matches exactly the list of records returned by the source.

### Search mirrored Format collection by provenance information.

<b>Test Case ID</b>	<b>3.1.6</b>
<b>Description</b>	Search the mirror collection using provenance information to verify that the indexing is correct and that all the mirrored records contain provenance data describing the source node.
<b>Actors</b>	Public
<b>Pre-conditions</b>	The mirror node contains records replicated from the source node.
<b>Primary functional path</b>	Execute the search service on the mirror and note the records found: <code>&lt;MIRROR_node_base_url&gt;/webservices/registry/search/Formats</code>
	Enter the URL of the source node in the value field for <code>local.provenance</code> index to return all the existing records and use "=" as operator.
	Make a note of the list of the records returned by each search.
<b>Verification</b>	All the records in the collection must have been returned.
	Retrieve the XML <code>adminData</code> of the each of the returned records and verify that the provenance data correctly describes where the record originated and that information correctly describes the source node.

## Search mirrored Format collection by original creator.

<b>Test Case ID</b>	<b>3.1.6</b>
<b>Description</b>	Search the mirror collection using creator information to verify that the indexing is correct and that all the mirrored records contain the same creator as the source records.
<b>Actors</b>	Public
<b>Pre-conditions</b>	The mirror node contains records replicated from the source node. Users were authenticated as the build-in user account, <code>gdfrUser</code> , when the records were created.
<b>Primary functional path</b>	Execute the search service on BOTH the source and mirror nodes and note the records found: <code>&lt;SOURCE_node_base_url&gt;/webservices/registry/search/Formats</code> <code>&lt;MIRROR_node_base_url&gt;/webservices/registry/search/Formats</code> Enter the userid "gdfrUser" in the value field for <code>local.creator</code> index to return all the existing records and use "=" as operator. Make a note of the list of the records returned by each search.
<b>Verification</b>	All the records in the collection must have been returned. Retrieve the XML <code>adminData</code> of the each of the returned records and verify that the creator information correctly identifies the userid in the source node.

Repeat this pattern for 4 additional collections:

Agents – use the numbering pattern 3.2.x

Hardware – use the numbering pattern 3.3.x

Software – use the numbering pattern 3.4.x

Since the process and software is identical for all collections, it is unnecessary to continue for all collections. However, Files collection contains binary data (not XML records) so it would be valuable to perform further testing for the following collections:

FileDescriptors – use the numbering pattern 3.5.x

Files – use the numbering pattern 3.6.x

## Synchronize GDFR collections.

<b>Test Case ID</b>	<b>3.7.1</b>
<b>Description</b>	Replicate all the GDFR collections.
<b>Actors</b>	GDFR User
<b>Pre-conditions</b>	The user is authenticated.
	The mirror node was initialized correctly, and replication configuration for all GDFR collections has been created.
	GDFR collections contain some records – for example, make sure Agents, Files, FileDescriptors, Software and Hardware each contain one or two records.
<b>Primary functional path</b>	Execute the search service on the source node on each collection and note the records found: <code>&lt;SOURCE_node_base_url&gt;/webservices/registry/search/Formats</code> <code>&lt;SOURCE_node_base_url&gt;/webservices/registry/search/Agents</code> <code>&lt;SOURCE_node_base_url&gt;/webservices/registry/search/Files</code> [... etc ... for all GDFR collections ] Enter "*" in the value field for <code>dc.identifier</code> index to return all the existing records.
	Execute the synchronize service on the mirror node and note the records found: <code>&lt;MIRROR_node_base_url&gt;/webservices/registry/synchrhonorize/Collections</code>
<b>Verification</b>	Retrieve all the records in the ReplicationConfig collection. <code>&lt;MIRROR_node_base_url&gt;/webservices/registry/search/ReplicationConfig</code>
	Enter "*" in the value field for <code>dc.identifier</code> index to return all the existing records.
	Verify that one record exists for each of the GDFR collections.
	Verify that the results in each of those records match the results of the searches executed in step #1 above.
	Verify that the system collections (Users, Roles, Features) have NOT been replicated.

**Attempt to replicate Features collection to verify no access.**

<b>Test Case ID</b>	<b>3.10.1</b>
<b>Description</b>	Some system collections, such as Features, are not available for replication and synchronization. The correct result of this test case is an authorization error.
<b>Actors</b>	Any: Public, GDFR User, GDFR Admin.
<b>Pre-conditions</b>	The user is not authenticated.
<b>Primary functional path</b>	Execute the importSRU service on the mirror node: <MIRROR_node_base_url>/webservices/registry/importSRU/Features Append the service arguments: baseUrl=<SOURCE_node_base_url>/webservices/registry/importSRU/Features since=1900
	Authenticate as GDFR User.
	Execute the importSRU service on the mirror node: <MIRROR_node_base_url>/webservices/registry/importSRU/Features Append the service arguments: baseUrl=<SOURCE_node_base_url>/webservices/registry/importSRU/Features since=1900
	Authenticate as GDFR Admin.
<b>Verification</b>	Execute the importSRU service on the mirror node: <MIRROR_node_base_url>/webservices/registry/importSRU/Features Append the service arguments: baseUrl=<SOURCE_node_base_url>/webservices/registry/importSRU/Features since=1900
	All three attempts must fail with HTTP 403 Unauthorized condition.  No records were retrieved from the source node.

**Attempt to replicate Roles collection to verify no access.**

<b>Test Case ID</b>	<b>3.10.2</b>
<b>Description</b>	Some system collections, such as Roles, are not available for replication and synchronization. The correct result of this test case is an authorization error.
<b>Actors</b>	Any: Public, GDFR User, GDFR Admin.
<b>Pre-conditions</b>	The user is not authenticated.
<b>Primary functional path</b>	Execute the importSRU service on the mirror node: <MIRROR_node_base_url>/webservices/registry/importSRU/Roles Append the service arguments: baseURL=<SOURCE_node_base_url>/webservices/registry/importSRU/Roles since=1900
	Authenticate as GDFR User.
	Execute the importSRU service on the mirror node: <MIRROR_node_base_url>/webservices/registry/importSRU/Roles Append the service arguments: baseURL=<SOURCE_node_base_url>/webservices/registry/importSRU/Roles since=1900
	Authenticate as GDFR Admin.
	Execute the importSRU service on the mirror node: <MIRROR_node_base_url>/webservices/registry/importSRU/Roles Append the service arguments: baseURL=<SOURCE_node_base_url>/webservices/registry/importSRU/Roles since=1900
<b>Verification</b>	All three attempts must fail with HTTP 403 Unauthorized condition.
	No records were retrieved from the source node.

**Attempt to replicate UserAccounts collection to verify no access.**

<b>Test Case ID</b>	<b>3.10.3</b>
<b>Description</b>	Some system collections, such as UserAccounts, are not available for replication and synchronization. The correct result of this test case is an authorization error.
<b>Actors</b>	Any: Public, GDFR User, GDFR Admin.
<b>Pre-conditions</b>	The user is not authenticated.
<b>Primary functional path</b>	Execute the importSRU service on the mirror node: <code>&lt;MIRROR_node_base_url&gt;/webservices/registry/importSRU/UserAccounts</code> Append the service arguments: <code>baseUrl=&lt;SOURCE_node_base_url&gt;/webservices/registry/importSRU/UserAccounts</code> <code>since=1900</code>
	Authenticate as GDFR User.
	Execute the importSRU service on the mirror node: <code>&lt;MIRROR_node_base_url&gt;/webservices/registry/importSRU/UserAccounts</code> Append the service arguments: <code>baseUrl=&lt;SOURCE_node_base_url&gt;/webservices/registry/importSRU/UserAccounts</code> <code>since=1900</code>
	Authenticate as GDFR Admin.
<b>Verification</b>	Execute the importSRU service on the mirror node: <code>&lt;MIRROR_node_base_url&gt;/webservices/registry/importSRU/UserAccounts</code> Append the service arguments: <code>baseUrl=&lt;SOURCE_node_base_url&gt;/webservices/registry/importSRU/UserAccounts</code> <code>since=1900</code>
	All three attempts must fail with HTTP 403 Unauthorized condition.  No records were retrieved from the source node.