

1. Fedora 3.4 Documentation	3
1.1 Downloads	4
1.2 Fedora Software Distribution	4
1.2.1 Release Notes	4
1.2.1.1 Fedora Repository 3.4.2 Release Notes	5
1.2.1.2 Fedora Repository 3.4.1 Release Notes	6
1.2.1.3 Fedora Repository 3.4 Release Notes	6
1.2.1.4 Fedora Repository 3.4-RC1 Release Notes	10
1.2.2 Installation and Configuration Guide	12
1.2.2.1 Alternative Webapp Context Configuration	18
1.2.2.2 Configuring Low Level Storage	19
1.2.2.3 HTTP Proxy Configuration	21
1.2.2.4 Installation From Source	21
1.2.2.5 Installing Java	22
1.2.2.6 Setting JAVA_HOME in Windows	23
1.2.3 Upgrade Guide	24
1.2.3.1 Upgrading from 2.x	25
1.2.3.2 Upgrading from 3.x	31
1.2.4 License and Copyright	31
1.2.5 Distribution Structure	32
1.2.6 Demonstration Objects	32
1.3 Getting Started with Fedora	34
1.4 Fedora Digital Objects	36
1.4.1 Fedora Digital Object Model	36
1.4.2 Fedora Identifiers	42
1.4.3 Digital Object Relationships	44
1.4.4 Content Model Architecture	47
1.4.5 CMA Construction Guide	50
1.4.5.1 Creating a Service Definition	51
1.4.5.2 Creating a Service Deployment	53
1.4.6 Introduction to FOXML	60
1.4.6.1 FOXML Ingest Example	61
1.4.7 Fedora Atom	63
1.4.7.1 ATOM Ingest Example	65
1.4.8 Fedora METS	68
1.4.8.1 METS Ingest Example	69
1.4.9 Ingest and Export	72
1.4.9.1 Ingest with the file URI scheme	73
1.4.10 Portable Fedora Objects	74
1.5 Fedora Repository	76
1.5.1 Checksums	76
1.5.2 Command-Line Utilities	77
1.5.3 Messaging	79
1.5.4 Replication and Mirroring	84
1.5.4.1 Journaling	85
1.5.5 Resource Index	94
1.5.5.1 Triples in the Resource Index	99
1.5.6 Security	101
1.5.6.1 Authentication and User Attributes	104
1.5.6.2 Fedora Security Layer (FeSL)	105
1.5.6.2.1 FeSL Installation	105
1.5.6.2.2 FeSL Authentication	106
1.5.6.2.3 FeSL Authorization	108
1.5.6.3 XACML Policy Enforcement	109
1.5.6.3.1 Fedora XACML Policy Writing Guide	119
1.5.6.3.2 XACML Vocabulary and Examples	130
1.5.7 Service Framework	132
1.5.8 Versioning	133
1.5.9 Web Service Interfaces	137
1.5.9.1 API-A	137
1.5.9.2 API-A-LITE	141
1.5.9.3 API-M	147
1.5.9.4 API-M-LITE	153
1.5.9.5 REST API	155
1.5.9.6 Basic Search	176
1.5.9.7 Basic OAI-PMH Provider	179
1.5.9.8 Resource Index Search	179
1.6 Fedora Framework Services	183
1.7 Fedora Clients	183
1.7.1 Client Command-line Utilities	183
1.7.2 Example SOAP Client	189
1.7.3 Fedora Administrator	190
1.7.3.1 Batch Processing	206
1.7.4 Fedora Web Administrator	223
1.7.5 Messaging Client	227

1.7.6 REST API Java Client	227
1.8 Fedora Local Services	227
1.8.1 FOP Service	227
1.8.2 Image Manipulation Service	228
1.8.3 SAXON XSLT Processor Service	228
1.9 Frequently Asked Questions	230
1.9.1 Administration FAQ	231
1.9.1.1 How do I configure Fedora to allow API-M access from remote hosts?	231
1.9.1.2 How do I turn off API-M access?	231
1.9.1.3 How do I undo SSL-only access to API-M functions?	232
1.9.2 Backup And Rebuild FAQ	233
1.9.2.1 How can I backup my Fedora Repository without having to rebuild ?	233
1.9.2.2 Is it possible to rebuild without taking my repository offline?	234
1.9.2.3 What do I need to backup in order to be able to rebuild my Fedora Repository ?	236
1.9.3 Development FAQ	237
1.9.3.1 How do you configure Eclipse for Fedora with Maven?	237
1.9.3.2 Where are Fedora's bugs reported?	255
1.9.3.3 Where is the source code ? How can it be accessed ?	256
1.9.4 Installation FAQ	256
1.9.4.1 How do I the get Fedora running under the JBoss Application Server?	257
1.9.4.2 How do I uninstall Fedora Commons ?	257
1.9.4.3 Which Linux Distribution is best for Fedora Commons?	257
1.9.5 Usage FAQ	257
1.9.5.1 "Policy blocked datastream resolution" error while adding datastream	258
1.9.5.2 How can I implement collections in Fedora Commons?	258
1.9.5.3 How can I retrieve the audit datastream ?	259
1.9.5.4 How do I customize the HTML views produced by the REST API?	259
1.9.5.5 How do I set my own PIDs (at runtime) for Fedora objects?	259
1.9.5.6 Mime type for SDep Output	259
1.9.5.7 Must I provide DC, RELS-EXT and RELS-INT as Inline XML Metadata?	260
1.9.5.8 Resource Index date comparison with ITQL	260
1.9.5.9 Where is the list of Dublin Core and FOXML properties?	260

Fedora 3.4 Documentation

Downloads

Fedora Software Distribution

- Release Notes
- Installation and Configuration Guide
- Upgrade Guide
- License and Copyright
- Distribution Structure
- Demonstration Objects

Getting Started with Fedora

Fedora Digital Objects

- Fedora Digital Object Model
- Fedora Identifiers
- Digital Object Relationships
- Content Model Architecture
- CMA Construction Guide
- Introduction to FOXML
- Fedora Atom
- Fedora METS
- Ingest and Export
- Portable Fedora Objects

Fedora Repository

- Checksums
- Command-Line Utilities
- Messaging
- Replication and Mirroring
- Resource Index
- Security
- Service Framework
- Versioning
- Web Service Interfaces

Fedora Framework Services

Fedora Clients

- Client Command-line Utilities
- Example SOAP Client
- Fedora Administrator
- Fedora Web Administrator
- Messaging Client
- REST API Java Client

Fedora Local Services

- FOP Service
- Image Manipulation Service
- SAXON XSLT Processor Service

Frequently Asked Questions

- [Administration FAQ](#)
- [Backup And Rebuild FAQ](#)
- [Development FAQ](#)
- [Installation FAQ](#)
- [Usage FAQ](#)

Downloads

Fedora

Fedora 3.4.2 Installer	110mb	MD5 Checksum	PGP Signature
Fedora 3.4.2 Source Code	14mb	MD5 Checksum	PGP Signature
Fedora 3.4 Messaging Client	7mb	MD5 Checksum	PGP Signature
Fedora 3.4 Journal Receiver	0.5mb	MD5 Checksum	PGP Signature

Compatible Services

GSearch 2.2	19mb	Documentation
OAI Provider 2.1	6mb	Documentation
SWORD-Fedora 1.2		Documentation
DirIngest 1.2	8mb	Documentation

Related Software

Please see [Fedora Create](#) for a complete list of Fedora-related software and projects.

Fedora Software Distribution

[Release Notes](#)

[Installation and Configuration Guide](#)

[Upgrade Guide](#)

[License and Copyright](#)

[Distribution Structure](#)

[Demonstration Objects](#)

Release Notes

[Fedora Repository 3.4.2 Release Notes](#)

[Fedora Repository 3.4.1 Release Notes](#)

[Fedora Repository 3.4 Release Notes](#)

[Fedora Repository 3.4-RC1 Release Notes](#)

Fedora Repository 3.4.2 Release Notes

Introduction

Release Date: January 19th, 2011






























Fedora 3.4.2 is a bugfix release that addresses several memory and stability problems with the core repository service.

Many thanks to those in the community who contributed to this release; particularly Janna Wemekamp (FCREPO-774 and FCREPO-823), Jörg Panzer (FCREPO-774), Nigel Thomas (FCREPO-846) and Mark Roy (FCREPO-820). Your help in identifying bugs, submitting patches, and road-testing fixes is very much appreciated.

We rely on continuing feedback from the community to ensure the ongoing quality and improvement of the Fedora Repository. Contributions are always welcome, particularly comments and feedback on your use of Fedora, reporting of any bugs you discover, code contributions and patches, and assistance with testing. Please let us have your thoughts via the [Fedora Commons Users mailing list](#). Please see our [Getting Involved](#) page for more information about code contributions.

See below for details about each change in this release.

Bug Fixes and Improvements

(16 issues)			
Type	Key	Summary	Priority
	FCREPO-819	Memory leak due to Mulgara bug	
	FCREPO-811	POLICY datastreams (possibly others) subject to race conditions when control group is "R"	
	FCREPO-835	Migrate Fedora Source Code to GitHub	
	FCREPO-822	Ingesting with "M" datastream contentLocation set to a file fails with Akubra	
	FCREPO-845	Memory leak in OwlApi used by ECM	
	FCREPO-774	REST Servlet throws exception under access load.	
	FCREPO-843	Ban all known slf4j bindings except logback	
	FCREPO-842	If a previous "add" to the policy index (dbxml) fails, subsequent updates fail as the existing doc can't be deleted	
	FCREPO-837	Standardize on LF for all text-based formats in the git repository	
	FCREPO-820	modifyObject API-M with no state change causes XACML policy to be removed from index.	
	FCREPO-785	ECM validator reports invalid, if no form declaration is specified	
	FCREPO-784	Fedora object 3.0 declare wrong format uri for the POLICY datastream	
	FCREPO-841	FeSL FedoraRIAttributeFinder attempts to locate RI attributes using hierarchical resource paths	
	FCREPO-780	NPE thrown on disseminations if external web service response does not contain a Content-type header	
	FCREPO-846	Concurrency issues with DBXML	
	FCREPO-823	Ingesting object with managed content DC datastream fails if content specified using a file:/// URL	

Known Issues

Please see the [bug tracker](#) for the latest list of outstanding issues.

Migration to GitHub

This is the first release of Fedora since the source code was migrated from SourceForge to [GitHub](#). The Fedora Repository GitHub page can be found [here](#). Instructions on accessing the source code can be found at [\[here\]](#).

Previous Release Notes

All release notes for Fedora 3.4.x can be found [here](#).

Fedora Repository 3.4.1 Release Notes

About This Release

Release Date: October 19th, 2010

This bugfix release FCREPO-798, a “Denial of Service” (DOS) vulnerability affecting all prior versions of Fedora 2.x and 3.x, and [FCREPO-790](#), a bug that could lead to the operating system running out of file handles.

FCREPO-798 was discovered during a code review and verified in testing. However, there have been no known attacks on any public or private Fedora repository. Our review indicates this vulnerability can corrupt the Fedora database in a way that will cause failure of your operating repository. However, **it cannot be used to damage your archival storage**. Fortunately, the repository may be recovered through the use of the rebuilder utility but until your system is patched it could be subject to additional DOS attacks.

A set of patches for Fedora 3.3 and Fedora 3.4 as well as a full release of Fedora 3.4.1 in which the issue is fixed has been posted on SourceForge. We ask you contact your repository operator immediately about the issue. If you are using Fedora 3.0 through 3.2, we urge you to update to patched copies of Fedora 3.3 or 3.4, or the 3.4.1 release at your earliest opportunity. The security releases may be found at:

- <http://sourceforge.net/projects/fedora-commons/files/fedora/3.3.1/>
- <http://sourceforge.net/projects/fedora-commons/files/fedora/3.4.1/>

The instructions for installation may be found in the README files at the above locations along with the downloads.

Unfortunately, Fedora 2 repositories remain vulnerable; a patch to Fedora 2, whose code base was declared at “end-of-life” two years ago, has proven beyond our resources at this time. Because of this, we will not be providing details about potential exploits in the near term. Fedora 2 installations are still of great concern to the Fedora committers since we know there are many installations in our community who may not be in a position to update to the latest Fedora release. We are seeking resources or volunteers to fix Fedora 2 but, at this time, we are not able to commit to a timeline for this work.

If you cannot update soon please read the following section containing suggestions that may help mitigate the vulnerability of your repository. Your installation may have minimal risk if Fedora is not directly exposed to un-trusted users. You should:

- Restrict access to Field Search including for front applications which pass unmodified query parameter text directly from users
- Restrict access from anonymous users for:
 - API-A Lite “get” operations
 - REST API “get” operations
 - REST API “findObjects” operations
- Restrict ingest of new digital objects from un-trusted users

If you have front-end applications (like Islandora or Muradora) which control access, the format of queries, or FOXML ingest or modifications your risks are mitigated. It is best if direct access to Fedora is hidden from users and only your front-end applications are exposed. In all cases, we recommend close monitoring of your repository.

This notification is to warn operators of production Fedora repositories. Please notify us if you have a sudden, unexplained failure of your system. As with all software, security issues may arise. We are collecting contact information for a responsible person for each production Fedora systems to help the notification process. Could you or your repository administrator please provide us with a suitable contact? If you know of any other production Fedora repositories, could you provide a suitable contact for it?

If you have any questions or are operating a Fedora system in production please contact [ddavis at duraspace dot org](#) or [cwilper at duraspace dot org](#).

Previous Release Notes

All release notes for Fedora 3.4.x can be found [here](#).

Fedora Repository 3.4 Release Notes

Introduction

Release Date: August 23, 2010

We are proud to announce the release of Fedora 3.4. Although a minor release, this version includes a number of exciting new features and bug fixes that make Fedora an ever more compelling repository platform.

You are encouraged to download this new release and give it a spin. Please let us have your comments and feedback via the mailing lists, and of course please let us know of any problems you discover.

Some of the important new features are:

- **DC, RELS-EXT, RELS-INT as Managed Content:** The Dublin Core and Relationships datastreams can now be stored as Managed Content, improving performance particularly when these datastreams are large. A migration tool is included to migrate existing inline XML datastreams to managed content datastreams
- **REST API relationships methods:** New methods in the REST API for adding and manipulating relationships in RELS-EXT and RELS-INT
- **Enhanced Content Models:** Including the ability to validate objects against their content models, and support for optional datastreams
- **Optimistic Locking:** The REST API now provides support for optimistic locking to ensure no one else has made a change to an object since you started editing it
- **FeSL Authentication:** FeSL Authentication can now be used independently of FeSL's experimental authorization mechanism, and is now the default authentication mechanism (although the old mechanism can be specified during installation). FeSL Authorization is still disabled by default.
- **FeSL policies as Fedora Objects:** XACML policies are now managed in FeSL directly through the Fedora API by manipulating Fedora objects containing a FESLPOLICY datastream
- **Logging reconfiguration without restart:** Using the new SLF4J and Logback logging framework, logging configuration changes now become effective without having to restart the server
- **Akubra low-level storage:** Akubra is now considered production-ready and is the default low-level storage module
- **REST API improvements and bug fixes:** further stabilizing the REST API
- **Deprecation of "LITE" APIs:** As of this release, the API-A-LITE and API-M-LITE APIs are deprecated, and will be removed in a future release. You are encouraged to migrate any code using these APIs to use the new REST API.

As of version 3.4, we recommend the [MediaShelf Java Client Library](#).. The existing client libraries supplied with the server distribution are now considered deprecated, and will no longer be maintained.



Java package naming and Java version

- As of version 3.4, Java 5 (JDK 1.5) is no longer supported. Java 6 (JDK 1.6) or later is required both to build and run Fedora.
- Java package names have undergone a thorough revision for this release. If you have your own code built against Fedora libraries and you update these to use the libraries from this version you will need to update your code to use the new package names.
- **If you have built your own storage plugins you will need to modify these to work with the new package names and some minor changes to the ILowLevelStorage interface**



Akubra Low-level Storage

- As of version 3.4, Akubra is the default low-level storage module, replacing the legacy file storage module
- Akubra is not backwards-compatible with object and datastream file storage from pre 3.4 repositories (these will have been using the legacy file storage module)
- **If you are upgrading an existing pre-3.4 repository please ensure you select the legacy-fs option for Low Level Storage when installing** (or use `lstore.type=legacy-fs` in your `install.properties` file)



Enhanced Content Models

- Version 3.4 includes Enhanced Content Models (ECM)
- This is the first version of Fedora with ECM, and we welcome feedback from the community on these new features
- Please note that some features in ECM are liable to change based on this feedback



REST API

- In version 3.4 many bug-fixes and improvements have been made to the REST API
- Previously the output of some REST API methods was incorrect and invalid against the schemas
- **If your client applications use the REST API, you may need to update your handling of some REST API responses**, particularly if you use namespace-aware parsing (default namespaces have been added to some method responses which were previously missing these)

For a detailed list of features and bugfixes comprising this release see the tables below.





































































Features and Improvements





(25 issues)			
Type	Key	Summary	Priority
	FCREPO-705	Extend Config A test coverage to include REST API support	
	FCREPO-683	Enhanced Content Models	
	FCREPO-428	Extend fedora-validate-objects to include referential integrity reporting	
	FCREPO-531	CMA and server should support optional datastreams sensibly	
	FCREPO-551	Akubra replaces LLStore as primary blob storage abstraction	
	FCREPO-385	Extend Validator to Support CMA-defined integrity checks	
	FCREPO-603	Standalone, reusable, Java client library for Fedora	
	FCREPO-492	Allow DC, RELS-EXT, etc, to be Managed Content	
	FCREPO-445	Improved CMA modeling capabilities	
	FCREPO-730	Provide link to datastream history from the "datastream profile view"	
	FCREPO-685	Validate API method	
	FCREPO-622	Java repackaging: org.fcrepo	
	FCREPO-630	Switch to SLF4J with Logback as primary logging framework	
	FCREPO-689	Enable optimistic locking for modify operations	
	FCREPO-621	Move to Java 6+ (Java 5 no longer supported)	
	FCREPO-639	Allow log re-configuration without restart	
	FCREPO-733	Get validate to also validate the ontologies for RELS-INT	
	FCREPO-669	Do a Java version sanity check at install time	
	FCREPO-670	Better error reporting when database connection test fails at install time	
	FCREPO-671	Allow use of FeSL AuthN without FeSL AuthZ during install	
	FCREPO-686	Migration tool for migrating datastreams from one type (control group) to another	
	FCREPO-577	Make XACML policies first-class Fedora digital objects	
	FCREPO-552	Configurable parameters for the Fedora internal http client	
	FCREPO-646	Remove undocumented/unused/broken ReportServlet	
	FCREPO-647	Remove unused/unnecessary ThreadMontitor	

Bug Fixes

--	--	--	--

(44 issues)

Type	Key	Summary	Priority
	FCREPO-753	Credentials not passed through for API-A REST calls when FeSL AuthN enabled	
	FCREPO-703	REST API calls to getDatastreamDissemination fails when XACML enabled, API-A auth disabled	
	FCREPO-701	REST API purgeDatastream - invalid date format for start time purges all versions of a datastream	
	FCREPO-696	Adding datastream with managed content throws Exception when providing a checksum	
	FCREPO-623	Journaling RMI transport writer does not send close() events	
	FCREPO-680	REST API throws NPE when adding new datastream without providing media- and mime type	
	FCREPO-699	PUT /objects/{pid}/datastreams/{dsid} sets versionable to true if it is not specified (REST API)	
	FCREPO-627	Fedora REST-style object viewer does not properly handle encoded slashes in the pid value.	
	FCREPO-641	AuthFilterJAAS prevents the server from starting when FEDORA_HOME is not set	
	FCREPO-712	Date strings with milliseconds are parsed incorrectly	
	FCREPO-736	System test failures when fesl authn enabled (fesl authz disabled)	
	FCREPO-754	No authentication for UserServlet with FeSL AuthN and API-A auth turned off	
	FCREPO-509	REST API does not properly handle ingests with FOXML that does not contain a PID	
	FCREPO-638	Fedora basic search interface returns results with old-style "/fedora/get" URLs instead of new REST interface URLs	
	FCREPO-660	Regression: setting FEDORA_HOME via context-param fails	
	FCREPO-566	Trunk fails to build on Windows if path contains spaces	
	FCREPO-704	Ingesting a managed content datastream via REST fails with files larger than 2GB	
	FCREPO-802	external JMS broker unavailable at startup halts startup	
	FCREPO-628	FieldSearch query with bad syntax does not immediately release connection to pool	
	FCREPO-64	Datastream SIZE attribute not fully implemented	
	FCREPO-615	With FeSL enabled, authentication is required for every resource	
	FCREPO-732	Parameter datastreamDefaultFilename mis-spelt as datstreamDefaultFilename in fedora.fcfg	
	FCREPO-708	REST API purgeDatastream returns HTTP 204 No Content instead of the dates of the deleted versions	
	FCREPO-758	FeSL policy validation issues	
	FCREPO-698	info:fedora registration contains outdated information	
	FCREPO-620	Tomcat 6 server.xml:SSLEnabled defaults to false with non-bundled Tomcat	
	FCREPO-616	Installer-generated web.xml drops namespace and schemaLocation declarations	
	FCREPO-607	REST getObjectProfile method has a datetime attribute in the return value, but this is not present in the schema	
	FCREPO-771	FeSL - conversion of policy "name" to PID incorrect	
	FCREPO-766	System test failure on Windows with Akubra: TestCommandLineFormats.testExportATOM_ZIP	
	FCREPO-700	REST api export specifies the wrong media type for atomZip archives	
	FCREPO-609	Purge object will throw general exception if force=true	
	FCREPO-587	DOValidatorSchematron error on Fedora startup with WebLogic	
	FCREPO-610	ObjectHistory only includes datastream timestamps, and only from currently active datastreams	

	FCREPO-618	Content-Disposition tests in fedora.test.api.TestRESTAPI fail on WebLogic	
	FCREPO-666	Maven build warnings for \${version} \${pom.version} \${artifactId}	
	FCREPO-640	Some IPv6 loopback addresses fail to match for default policies	
	FCREPO-665	DefaultDOManager logging WARNings for valid dsLocations	
	FCREPO-617	"Other" servlet container option does not provide local services webapps	
	FCREPO-613	Purge datastream will throw a general exception if force=true	
	FCREPO-673	REST getNextPID with invalid PID namespace throws error but results in update to PIDGEN table	
	FCREPO-614	ConfigC test failures/errors	
	FCREPO-612	Datastream Profile in /objects/{pid}/datastreams/{dsId}?format=xml use attribute datetime, which is not in the schema	
	FCREPO-727	Provide a link to datastream content from the "list datastreams" view	

Previous Release Notes

All release notes for Fedora 3.4.x can be found [here](#).

Fedora Repository 3.4-RC1 Release Notes

Introduction

We are proud to announce the release of Release Candidate 1 of Fedora 3.4. Although a minor release, this version includes a number of exciting new features and bug fixes that make Fedora an ever more compelling repository platform. We aim to release the final version of Fedora 3.4 in August, and in the meantime you are encouraged to give this Release Candidate a try. Please let us have any comments and any problems you find via the mailing list.

Some of the important new features are:

- **DC, RELS-EXT, RELS-INT as Managed Content:** The Dublin Core and Relationships datastreams can now be stored as Managed Content, improving performance particularly when these datastreams are large
- **REST API relationships methods:** New methods in the REST API for adding and manipulating relationships in RELS-EXT and RELS-INT
- **Enhanced Content Models:** Including the ability to validate objects against their content models, and support for optional datastreams
- **Optimistic Locking:** The REST API now provides support for optimistic locking to ensure noone else has made a change to an object since you started editing it
- **FeSL Authentication:** FeSL Authentication can now be used independently of FeSL's experimental authorization mechanism, and is now the default authentication mechanism (although the old mechanism can be specified during installation). FeSL Authorization is still disabled by default.
- **Logging reconfiguration without restart:** Using the new SLF4J and Logback logging framework, logging configuration changes now become effective without having to restart the server
- **REST API improvements and bug fixes:** further stabilizing the REST API
- **Deprecation of "LITE" APIs:** As of this release, the API-A-LITE and API-M-LITE APIs are deprecated, and will be removed in a future release. You are encouraged to migrate any code using these APIs to use the new REST API.



Java package naming and Java version

- As of version 3.4, Java 5 (JDK 1.5) is no longer supported. Java 6 (JDK 1.6) or later is required both to build and run Fedora.
- Java package names have undergone a thorough revision for this release. If you have your own code built against Fedora libraries and you update these to use the libraries from this version you will need to update your code to use the new package names.

For a detailed list of features and bugfixes comprising this release see the tables below.

Features and Improvements

(20 issues)			
Type	Key	Summary	Priority

	FCREPO-705	Extend Config A test coverage to include REST API support	
	FCREPO-428	Extend fedora-validate-objects to include referential integrity reporting	
	FCREPO-531	CMA and server should support optional datastreams sensibly	
	FCREPO-551	Akubra replaces LLStore as primary blob storage abstraction	
	FCREPO-385	Extend Validator to Support CMA-defined integrity checks	
	FCREPO-603	Standalone, reusable, Java client library for Fedora	
	FCREPO-492	Allow DC, RELS-EXT, etc, to be Managed Content	
	FCREPO-445	Improved CMA modeling capabilities	
	FCREPO-622	Java repackaging: org.fcrepo	
	FCREPO-630	Switch to SLF4J with Logback as primary logging framework	
	FCREPO-689	Enable optimistic locking for modify operations	
	FCREPO-621	Move to Java 6+ (Java 5 no longer supported)	
	FCREPO-639	Allow log re-configuration without restart	
	FCREPO-733	Get validate to also validate the ontologies for RELS-INT	
	FCREPO-669	Do a Java version sanity check at install time	
	FCREPO-670	Better error reporting when database connection test fails at install time	
	FCREPO-671	Allow use of FeSL AuthN without FeSL AuthZ during install	
	FCREPO-552	Configurable parameters for the Fedora internal http client	
	FCREPO-646	Remove undocumented/unused/broken ReportServlet	
	FCREPO-647	Remove unused/unnecessary ThreadMonitor	

Bug Fixes

(35 issues)			
Type	Key	Summary	Priority
	FCREPO-753	Credentials not passed through for API-A REST calls when FeSL AuthN enabled	
	FCREPO-701	REST API purgeDatastream - invalid date format for start time purges all versions of a datastream	
	FCREPO-696	Adding datastream with managed content throws Exception when providing a checksum	
	FCREPO-623	Journaling RMI transport writer does not send close() events	
	FCREPO-680	REST API throws NPE when adding new datastream without providing media- and mime type	
	FCREPO-699	PUT /objects/{pid}/datastreams/{dsid} sets versionable to true if it is not specified (REST API)	
	FCREPO-627	Fedora REST-style object viewer does not properly handle encoded slashes in the pid value.	
	FCREPO-641	AuthFilterJAAS prevents the server from starting when FEDORA_HOME is not set	
	FCREPO-736	System test failures when fesl authn enabled (fesl authz disabled)	
	FCREPO-754	No authentication for UserServlet with FeSL AuthN and API-A auth turned off	
	FCREPO-638	Fedora basic search interface returns results with old-style "/fedora/get" URLs instead of new REST interface URLs	
	FCREPO-660	Regression: setting FEDORA_HOME via context-param fails	
	FCREPO-566	Trunk fails to build on Windows if path contains spaces	

	FCREPO-704	Ingesting a managed content datastream via REST fails with files larger than 2GB	
	FCREPO-802	external JMS broker unavailable at startup halts startup	
	FCREPO-628	FieldSearch query with bad syntax does not immediately release connection to pool	
	FCREPO-64	Datastream SIZE attribute not fully implemented	
	FCREPO-615	With FeSL enabled, authentication is required for every resource	
	FCREPO-732	Parameter datastreamDefaultFilename mis-spelt as datstreamDefaultFilename in fedora.fcfg	
	FCREPO-708	REST API purgeDatastream returns HTTP 204 No Content instead of the dates of the deleted versions	
	FCREPO-698	info:fedora registration contains outdated information	
	FCREPO-620	Tomcat 6 server.xml:SSLEnabled defaults to false with non-bundled Tomcat	
	FCREPO-616	Installer-generated web.xml drops namespace and schemaLocation declarations	
	FCREPO-607	REST getObjectProfile method has a datetime attribute in the return value, but this is not present in the schema	
	FCREPO-700	REST api export specifies the wrong media type for atomZip archives	
	FCREPO-609	Purge object will throw general exception if force=true	
	FCREPO-587	DOValidatorSchematron error on Fedora startup with WebLogic	
	FCREPO-610	ObjectHistory only includes datastream timestamps, and only from currently active datastreams	
	FCREPO-618	Content-Disposition tests in fedora.test.api.TestRESTAPI fail on WebLogic	
	FCREPO-666	Maven build warnings for \${version} \${pom.version} \${artifactId}	
	FCREPO-640	Some IPv6 loopback addresses fail to match for default policies	
	FCREPO-665	DefaultDOManager logging WARNings for valid dsLocations	
	FCREPO-617	"Other" servlet container option does not provide local services webapps	
	FCREPO-673	REST getNextPID with invalid PID namespace throws error but results in update to PIDGEN table	
	FCREPO-727	Provide a link to datastream content from the "list datastreams" view	

Previous Release Notes

All release notes for Fedora 3.4.x can be found [here](#).

Installation and Configuration Guide

Introduction

This guide will show you how to install a new Fedora Repository using the installer, or from source code. If you are upgrading from a previous release, please see [Upgrading from 2.x](#) or [Upgrading from 3.x](#)

Prerequisites



Download Fedora 3.4.2

- [Fedora 3.4.2 Installer \(110M\)](#)
- [Fedora 3.4.2 Source Code \(14M\)](#)

Java SE Development Kit (JDK) 6.

Whether installing a binary or source distribution, JDK 6 is required. The JDK should be installed on the machine you intend to use as the Fedora server. It is available from <http://java.sun.com/>. Look [here](#) for more information on installing Java.

Database

Fedora uses a relational database to support some of its functions. To simplify installation, the Fedora installer includes and can configure an embedded instance of the [Derby SQL Database 10.5.3](#). Fedora supports four external databases: [MySQL](#), [Oracle](#), [PostgreSQL](#) and [Microsoft SQL Server](#). The embedded Derby database should only be used for evaluation and development purposes; Derby should not be used for any production repository. It is recommended that you use one of the supported external databases which must be installed, configured and running before proceeding with the installation. To configure Fedora to use an external database, please see the [Database](#) section below for further instructions.

Application Server

The Fedora Repository installer includes Tomcat 6.0.20. Optionally, Fedora may be installed into any existing application server that implements Servlet 2.5/JSP 2.1 or higher. At this time, Fedora has chiefly been tested with Tomcat 5.x and Tomcat 6.0.x but users have reported running Fedora successfully with Jetty and JBoss.

Maven 2

Fedora uses Maven for its build environment. Maven2 is available from <http://maven.apache.org/>.

Prepare Environment Variables

The following environment variables must be correctly defined:

JAVA_HOME

This should point to the base directory of your Java installation. On Windows systems, this might be `C:\java`. For UNIX derivatives, this might be something like `/usr/local/jdk1.6.0_17`.

FEDORA_HOME

This is the directory where Fedora will be installed, for example, `C:\fedora` (Windows) or `/usr/local/fedora` (UNIX derivatives). Note: This is only required when running the Fedora client command line utilities. The server also requires this information at run time, but can accept it from the following sources (listed in order of preference):

- The `fedora.home` init-param in the Fedora webapp's `web.xml` file (Fedora 3.2+ only). The installer will automatically include the correct path in your `web.xml` at installation time, so if you move your Fedora Home directory later, you will need to also modify this file and restart the webapp container.
- The `fedora.home` system property, configured as appropriate for your web application server of choice.
- The `FEDORA_HOME` environment variable, as available when the web application server starts.

PATH

This must include the Java and Fedora bin directories. For UNIX derivatives, this will be `$FEDORA_HOME/server/bin`, `$FEDORA_HOME/client/bin` and usually `$JAVA_HOME/bin`. For Windows, this will be `%FEDORA_HOME%\server\bin`, `%FEDORA_HOME%\client\bin` and usually `%JAVA_HOME%\bin`.

If you will be building from source, Maven should also be on your path.

JAVA_OPTS

If Fedora is configured to use SSL, `JAVA_OPTS` must include the `javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword` properties. See the SSL section below for more information.

CATALINA_HOME

If Fedora is configured to use Tomcat, `CATALINA_HOME` must be set before starting Fedora. If using the quick install option, `CATALINA_HOME` should be set to `$FEDORA_HOME/tomcat` (or `%FEDORA_HOME%\tomcat` in Windows).

DISPLAY (Unix-only)

When running a Fedora server in a Unix-like operating system (Linux, Solaris, OS X, etc), you should ensure that this environment variable is NOT set by the user who will be running the application server in which Fedora is installed (e.g. Tomcat). *Background:* Fedora and the included web applications are designed to run without access to a graphics output device. Although rare, having this environment variable set has been reported to cause stability problems in certain installations of Fedora. Since a graphic output device should never be needed by the Fedora server, it is safest to ensure this environment variable is not set.

The Fedora Installer provides three installation options: *quick*, *custom*, and *client*.

To start the installer, change to the directory where you downloaded the installer and at a command prompt, enter:

```
java -jar fcrepo-installer-3.4.2.jar
```



Tip

Fedora can also be installed in non-interactive mode by specifying an `install.properties` file as an argument to the installer. After installing interactively, you will find an `install.properties` file in your `$FEDORA_HOME/install/` directory. You can use this file as a template for future, non-interactive installations. **Take care if you are using an `install.properties` file from an earlier release as the file may not contain some properties defined in the later release, default settings will be used in this case, which may or may not be appropriate for your installation**

Please ensure that the user account that is running the installer has sufficient permissions to write to the directories where Fedora will be installed (if deploying to an existing Tomcat installation, this includes permissions to the Tomcat directory). Installer created files will usually be owned by the user running the installer. Consequently, for example, after installation users of the Fedora Admin client will need write permissions to the log files defined by `FEDORA_HOME/client/log4j.xml`.

Quick Install

The quick option is designed to get Fedora up and running as quickly as possible, with a minimum of advanced options. The quick install will automatically install Tomcat pre-loaded with the Fedora Repository and the Derby database. Neither SSL support nor XACML policy enforcement is enabled by the quick install.

Custom Install

The custom option provides the most flexibility in configuring an installation. Options include the choice of servlet container, database, the host, ports and application server context Fedora will be running on, enabling optional services, as well as security options including SSL, XACML policy enforcement, and FeSL.

Servlet Container

The installer will automatically configure and deploy to Tomcat 5.0.x, 5.5.x, and 6.0.x servlet containers. However, if an existing Tomcat installation (as opposed to the Tomcat bundled with the installer) was selected, the installer will not overwrite your existing `server.xml`, but rather, place a modified copy at `FEDORA_HOME/install` so that you may review it before installing it yourself.

Other servlet containers will require manual deployment of the `war` files located at `FEDORA_HOME/install`.

Application Server Context

The installer provides the option to enter an application server context name under which Fedora will be deployed. The context name defaults to `Fedora` (resulting in `http[s]://host:port/fedora`), however any other valid context name can be supplied. The installer will name the resulting `war` file according to the supplied context name (defaults to `fedora.war`). Please ensure that the servlet container configuration reflects the name of the Fedora context name in case it needs to be configured explicitly. For further details see [Alternative Webapp Context Configuration](#).

SSL

Configuring SSL support for Fedora's API-M interface is an optional feature. It is strongly recommended for production environments if Fedora is exposed to unsecured application and users. However, if your installation is within a managed data center with firewall services, you may choose to provide SSL using a software or hardware front-end instead. For example, a reverse proxy implemented using the [Apache HTTP Server](#) and hiding Fedora generally provides better SSL performance.

If the Tomcat servlet container is selected, the installer will configure `server.xml` for you. However, as noted above, if an existing Tomcat installation was selected, the installer will not overwrite your existing `server.xml`.

Please consult your servlet container's documentation for certificate generation and installation. (In particular, the example certificate provided by the installer for Tomcat should not be used in a production environment).

If Fedora is configured to use SSL, the `JAVA_OPTS` environment variable must include the `javax.net.ssl.trustStore` and `javax.net.ssl.trustStorePassword` properties. The value of `javax.net.ssl.trustStore` should be the location of the truststore file and the value of `javax.net.ssl.trustStorePassword` is the password for the keystore. The following values may be used with the sample keystore included with the installer:

```
-Djavax.net.ssl.trustStore=$FEDORA_HOME/server/truststore -Djavax.net.ssl.trustStorePassword=tomcat
```

FeSL

The [Fedora Security Layer](#) is an experimental feature introduced from Fedora 3.3. FeSL Authentication is now the default authentication mechanism, however FeSL Authorization is still considered experimental. Enabling FeSL Authorization will disable the legacy policy enforcement. See [FeSL Installation](#) for more information about FeSL requirements that must be satisfied prior to installation.

Resource Index

If the [Resource Index](#) is enabled, Fedora will use [Mulgara](#) as its underlying triplestore, with full-text indexing disabled.

Messaging

If [Messaging](#) is enabled, Fedora will create and send a message via JMS whenever an API-M method is called.

Client Install

Both the quick and custom options will install the Fedora client software in addition to the Fedora server. The client option, however, will install only the Fedora client software.

Running the Fedora Server

If you selected the quick install option, you will find Tomcat installed in `FEDORA_HOME/tomcat`. To run Fedora, start Tomcat by entering:

```
$FEDORA_HOME/tomcat/bin/startup.sh
```

(or for Windows)

```
"%FEDORA_HOME%\tomcat\bin\startup.bat"
```

If you selected the custom install option, ensure that your database server is running (unless you selected the included Derby option which will be automatically started when the first database connection is made).

Demo Objects

If you just started Fedora for the first time, it's a good idea to check out the demonstration objects to get an idea of how Fedora works. See the [Demonstration](#) documentation for complete descriptions.

NOTE: If, during a custom install, you entered values other than the defaults for *fedoraServerHost* (localhost) or *fedoraServerPort* (8080), you must run the **demo object converter utility** script to change the host and/or port in the demonstration object ingest files. The demonstration object conversion is only required if you are ingesting demonstration objects. If the demonstration objects are already ingested into the repository (e.g. from a previous installation), there is no need for conversion. The demonstration objects are shipped with references to "localhost:8080" and these references must reflect the new values of *fedoraServerHost* *fedoraServerPort*. Refer to the [Command-line Utilities](#) documentation for additional details on running the demo object converter.

To ingest the demonstration objects, at a command prompt, enter:

```
fedora-ingest-demos.sh [hostname] [port] [username] [password] [protocol]
```

(or for Windows)

```
fedora-ingest-demos.bat [hostname] [port] [username] [password] [protocol]
```

For additional information on the `fedora-ingest-demos` command, see the documentation for the [Client Command-line Utilities](#). Please note that the demonstration objects must be ingested before they can be discovered using the default search interface.

Database

Fedora is designed to be RDBMS-independent. Fedora has been tested with Derby, McKoi, MySQL, Oracle, PostgreSQL and Microsoft SQL Server. The embedded version of Derby included with the installer is provided as a convenience; Derby is not recommended for use in production repositories. If you choose to use any database other than the embedded Derby provided by the Fedora Installer, you must install that database first.

Follow the instructions below for the RDBMS of your choice in order to create the user and tables required by Fedora.

MySQL

Please note that the MySQL JDBC driver provided by the installer requires MySQL v3.23.x or higher.

The MySQL commands listed below can be run within the `mysql` program, which may be invoked as follows:

```
mysql -u root -p
```

Create the database. For example, to create a database named "fedora3", enter:

```
CREATE DATABASE fedora3;
```

Set username, password and permissions for the database. For example, to set the permissions for user fedoraAdmin with password fedoraAdmin on database "fedora3", enter:

```
GRANT ALL ON fedora3.* TO fedoraAdmin@localhost IDENTIFIED BY 'fedoraAdmin';  
GRANT ALL ON fedora3.* TO fedoraAdmin@%' IDENTIFIED BY 'fedoraAdmin';
```

MySQL 4.1.x users must also specify the default character set for the Fedora database as "utf8" and the default collation as "utf8_bin". For example, to set the default character set and collation on a database named "fedora3", enter:

```
ALTER DATABASE fedora3 DEFAULT CHARACTER SET utf8;  
ALTER DATABASE fedora3 DEFAULT COLLATE utf8_bin;
```

Oracle

To prepare Oracle for use with Fedora, the following steps should be taken by an administrative user. First, using the Database Configuration Assistant, ensure that the database you'll be using is created with the UTF8 charset. Next, you'll need to create a Fedora tablespace and user in the database. Assuming the administrative user is sys and the SID is fedora3, log in using SQL*Plus using the following command:

```
sqlplus sys/PASSWORD@fedora3 as sysdba
```

To create a tablespace named "fedora_tbsp" with data in /var/lib/oracle, enter the following:

```
CREATE TABLESPACE fedora_tbsp  
DATAFILE '/var/lib/oracle/fedora_tbsp.dat' SIZE 1024M REUSE  
AUTOEXTEND ON NEXT 256M MAXSIZE UNLIMITED  
SEGMENT SPACE MANAGEMENT AUTO;
```

To create a user "fedoraAdmin" with password "fedoraAdmin", using the "fedora_tbsp", enter the following:

```
CREATE USER fedoraAdmin IDENTIFIED BY fedoraAdmin  
DEFAULT TABLESPACE fedora_tbsp;
```

Using the GRANT command, make sure the user has permission to connect, create, alter, and drop tables, sequences, triggers, and indexes in this tablespaces. For example:

```
GRANT ALL PRIVILEGES TO fedoraAdmin;
```

NOTE: Due to distribution license restrictions, the Fedora Installer does not include the Oracle JDBC driver. Oracle JDBC drivers are available from http://technet.oracle.com/software/tech/java/sqlj_jdbc/content.html. The installer will prompt you for the location of the driver on your filesystem. Also, if you run Fedora in Java 6 as is required by Fedora 3.4, you will need an Oracle Java 6 jdbc jar such as ojdbc6.jar.

PostgreSQL

Please consult the documentation at <http://www.postgresql.org/docs/> for more detailed information about configuring PostgreSQL.

Launch the PostgreSQL interactive terminal, psql, (optionally appending the -U argument to connect as a different user).

```
psql -d postgres
```

To create a user "fedoraAdmin" with password "fedoraAdmin" and database named "fedora3", enter the following:

```
CREATE ROLE "fedoraAdmin" LOGIN PASSWORD 'fedoraAdmin';  
CREATE DATABASE "fedora3" WITH ENCODING='UTF8' OWNER="fedoraAdmin";
```


Microsoft SQL Server

David Handy has contributed a [guide for interfacing Fedora with MS SQL Server](#)

Other Databases

To use a database other than Derby, McKoi, MySQL, Oracle, PostgreSQL and Microsoft SQL Server, the database must support common SQL-92 syntax and you must have a JDBC version 3 driver available.

The JDBC driver will need to be installed manually. For most containers, the driver may be placed in the Fedora webapp's `WEB-INF/lib` directory. For Tomcat 5.0.x, however, the driver should be installed to `TOMCAT_HOME/common/lib`. The JDBC URL will need to be configured appropriately in the Fedora Server Configuration File.

Upon startup, Fedora checks the database for all required tables. If the tables do not exist, Fedora will create them. Because table creation is much less standardized task across RDBMSs than SQL queries you must do one of the following:

1. Create the tables and indexes and auto-increments yourself in your own database (see the file: `src/dbspec/server/fedora/server/storage/resources/DefaultDOManager.dbspec` in the source distribution for the RDBMS-neutral table specifications).
2. Write a subclass of `fedora.server.utilities.DDLConverter` for your database software, include it in the Fedora `WEB-INF/classes` directory or in a jar file in the Fedora `WEB-INF/lib` directory, and associate it with the JDBC driver inside the `FEDORA_HOME/server/config/fedora.fcfg` file (see how it's done by looking at the `MySQLDDLConverter` and `DerbyDDLConverter` associations with their respective drivers in the `fedora.fcfg` file, and the classes' implementations in the source distribution). If you choose option #2, please tell us about it, as it will be useful for other users of Fedora. Option 2 is harder, but it will make future installations of new versions of Fedora (where the db schema will likely change) much easier for you if you plan on using that database later.

Configuring the Fedora Server

fedora.fcfg

The Fedora Server's configuration is chiefly governed by the Fedora Server Configuration File, `fedora.fcfg`, located at `FEDORA_HOME/server/config/fedora.fcfg`.

The Fedora server configuration file contains:

- Global parameters for the Fedora server
- Configuration parameters for each server module
- Configuration parameters for each persistent data store

The configuration file has a simple schema. It starts with a `server` element, under which a series of parameter elements occur, followed by a series of module elements, followed by a series of `datastore` elements. The parameter elements directly following the root `server` element are used to control what are considered generic server functionality; for example: the port on which the server is exposed.

The module elements are used to configure specific parts of Fedora. For instance, the module with the role attribute `fedora.server.search.FieldSearch` is used to configure the field-searching component of the server. Inside the module element, several `param` elements are included. These are specific to that module's implementation. Descriptions of each parameter can currently be found in the configuration file itself.

The `datastore` elements are used to configure various databases that might be used by the system. Although the sample configuration file holds several, you will typically only need one. The `datastore` elements are associated with the modules by means of a parameter inside the associated module. In the sample configuration file, for example, the `poolNames` parameter of the `fedora.server.storage.ConnectionPoolManager` module refers to one of the `datastore` elements in its value.

There are many other parameters you can configure with Fedora. Refer to the Fedora Server Configuration File itself (`fedora.fcfg`) for internal documentation on all the parameters.

Logging in Fedora

Fedora uses the Simple Logging Facade for Java (SLF4J) framework for logging with Logback as the actual logging implementation. For detailed information about using SLF4J, consult the SLF4J Manual: <http://www.slf4j.org/manual.html>, and for information about using Logback consult the Logback manual: <http://logback.qos.ch/manual/index.html>.

The log configuration file is located at `FEDORA_HOME/server/config/logback.xml`. One of the benefits of using SLF4J and Logback is that configuration changes take effect without needint to restart the server.

Normally, coarse-grained logs for Fedora are written to `FEDORA_HOME/server/logs/fedora.log`. The following examples show the kinds of configuration changes you can make to aid in debugging.

To change the level to `DEBUG` for all Fedora classes, change the `logger name="org.fcrepo"` line to the following:

```
<logger name="org.fcrepo" additivity="false" level="DEBUG">
```

To change the level to DEBUG for just one class, add the following lines:

```
log4j.logger.fedora.server.utilities.SQLUtility = DEBUG, FEDORA  
log4j.additivity.fedora.server.utilities.SQLUtility = false
```

To change the level to DEBUG for a whole package, add the following lines:

```
<logger name="org.fcrepo.server.resourceIndex" additivity="false" level="DEBUG">  
<appender-ref ref="FEDORA"/>
```

Related Topics

[Installation From Source](#)

Alternative Webapp Context Configuration

Introduction

This optional Fedora server configuration provides for the ability to deploy a Fedora server under a different webapp context than the default 'fedora'.

For example, instead of running the server at

- www.example.com/fedora

It is now possible to run at

- www.example.com/newContextName

One immediate opportunity offered by the decoupling of the server from a particular context path (namely, 'fedora'), is the capability to deploy multiple Fedora servers within a single application server.

Configuration

Installer

This is the step where the webapp context is actually set. If the 'custom' option is chosen during the execution of fcrepo-installer-3.3.jar, the option to specify "Fedora application server context" is presented. Simply specify the desired '[new context]' at this point if something other than the default ('fedora') is wanted.

Server Command-line Scripts

The server command-line scripts need to know the name of the app server context when they are run. By default, they assume 'fedora', but this assumption may be changed by defining the environment variable WEBAPP_NAME:

```
WEBAPP_NAME =newContextName
```

where newContextName is the fedora appserver context specified at installation.

Client Command-line Scripts

By default, the client command-line scripts assume 'fedora' as the app server context. This assumption may be changed by specifying the new context as an additional argument. See [Client Command-line Utilities](#) for the exact usage information for each script.

System Tests

This is relevant to the fedora development process, and not a concern for normal fedora operation. In the source distribution, live junit system

tests are defined in the maven pom file of the integration test subproject (fcrepo-integrationtest/pom.xml). When running these tests while using an alternate fedora app server context, the fedora.appserver property must be set to the new app server context. If it is not, certain tests for command line utilities will fail.

```
<properties>
...
<fedora.appServerContext>newContextName</fedora.appServerContext>
...
</properties>
```

Known Issues

When using an alternate context, the 'view' dissemination from the demo:SmileyStuff object will contain broken image links. This is a problem with the demo object itself, and does not represent user or system error. You can fix this manually by modifying the image-collection-demo/SmileyStuff-ViewStylesheet.xml stylesheet within the fedora-demo webapp. Simply change occurrences of "/fedora/get" to "/newContextName/get".

Configuring Low Level Storage

Introduction

Often abbreviated "LLStore", the *Low Level Storage Interface* is a critical component of Fedora. It stores and provides access to the authoritative copy of all digital object XML (FOXML) and datastreams managed by a Fedora repository.

LLStore is an internal Java interface and is not intended to be accessed directly by user applications. Instead, applications are expected to interact with Fedora through the web-based repository APIs.

Fedora's default LLStore module stores digital object XML and datastreams as individual files in a regular filesystem. However, there are several other options that can be configured.

General Configuration

If you don't want to use the default implementation or configuration options, you must configure the LLStore module before starting Fedora for the first time. Configuration is done within the server/config/fedora.fcfg file, by modifying the **class** and **param** values as appropriate for the LLStore implementation you are plugging in:

```
<module role="fedora.server.storage.lowlevel.ILowlevelStorage"
  class="org.example.SomeLLStoreModule">
  <param name="someParam" value="someValue"/>
  <!-- etc -->
</module>
```

Consult the documentation for each plug-in listed below for the specific class name and configuration options supported. Depending on the plug-in you are installing, you may also need to add one or more .jar files to the Fedora webapp's classpath (e.g. WEB-INF/lib).

Plug-Ins Provided by Fedora Commons

Akubra Low Level Storage (New)

[Akubra](#) is a newly-developed, generic "blob" storage API. The [AkubraLowlevelStorage](#) plug-in is included with Fedora 3.2 and acts as a bridge between Fedora and Akubra, so that Akubra implementations can be used to support low-level storage in Fedora.

Because it offers an improved file storage abstraction and is pluggable itself, we plan to use Akubra as the default low-level storage option for future releases of Fedora. However, it is also a very new API; we would like to gain more experience with it and hear more user feedback before making it the default option.

To use this plug-in, replace the existing LowlevelStorage module in fedora.fcfg with the following:

```
<module role="fedora.server.storage.lowlevel.ILowlevelStorage"
  class="fedora.server.storage.lowlevel.akubra.AkubraLowlevelStorageModule"/>
```

Then modify the akubra-llstore.xml file as appropriate. This is a Spring bean configuration file and is where you tell Fedora which Akubra BlobStore implementation to use and how it should be configured.

Default Akubra-LLStore Configuration

The default Akubra-LLStore configuration is similar to the Filesystem Low Level Storage module traditionally used by Fedora in that it stores object xml and datastreams as individual files on your local filesystem. However, it differs in a couple important ways:

- It does not require the use of database tables to "look up" the path to each file.
- It stores files in a deterministic location based on an md5 hash of a the unique id of each file.

Customizing the Akubra-LLStore Configuration

By editing the [default akubra-llstore.xml](#) file in a couple simple ways, you can control:

- The *base directory* of the object xml and datastream storage areas. Unless you are just testing out Fedora with Akubra, you should definitely change these values to a permanent storage location. Look for /tmp/datastreamStore and /tmp/objectStore in the file and change the values as desired.
- The *shape of the directory tree* in which to store the files. You control this by changing the "##" values in the configuration file. The default values of "##" ensure that there are at most 256 directories in each store, all at the top level, and files will be distributed fairly evenly among them. For more information on what "##" means and how it controls the directory structure, see [HashPathIdMapper](#) in the Fedora 3.2 Javadocs.

For more extensive information on Akubra, please refer to the [Akubra Wiki Space](#).

Filesystem Low Level Storage (Mature)

This is Fedora's default storage option and requires no additional setup to use. It comes pre-configured with your Fedora installation to store all objects in \$FEDORA_HOME/data/objects/, and all datastreams in \$FEDORA_HOME/data/datastreams/. Paths are allocated based on the date the item was first created. For example, a datastream created on May 8th, 2009 might be located in the 2009/0508/20/48/ directory.

To change these locations, consult the fedora.fcfg file that came with your Fedora installation and change the indicated param values as desired.

S3 Low Level Storage (Experimental)

This plugin stores all object XML and datastream content on Amazon's Simple Storage System (S3) and serves as a good example of how a LowlevelStorage implementation can be built. However, we do not recommend the use of this plugin in production environments because:

- It does not support rebuilds (see "Rebuilder Support" below)
- We plan to replace it with an Akubra S3 implementation in the near future.

For an overview of how this plugin works and how it can be configured to work with Fedora, [please see this document](#).

The S3 Low Level Storage is not distributed in binary form. You may obtain it from our legacy Subversion repository via:

```
svn co https://svn.fedora-commons.svn.sourceforge.net/svnroot/fedora-commons/incubator/AmazonS3Storage
```

Third-Party Plug-Ins

iRODs Low Level Storage

This plugin was developed by the [DICE Group](#) and allows Fedora to use iRODs to store digital objects and datastreams. It is based on the SRB plugin developed by the DART Project (see below).

For more information on this plugin, please visit <https://www.irods.org/index.php/Fedora>

SRB Low Level Storage

This plugin was developed by the [DART Project](#) and allows Fedora to use the [Storage Resource Broker \(SRB\)](#) to store digital objects and datastreams.

For more information on this plugin, please visit <http://www.itee.uq.edu.au/~eresearch/projects/dart/outcomes/FedoraDB.php>

Sun Honeycomb Low Level Storage

This plugin was developed by [Sun Microsystems](#) and allows Fedora to use the [StorageTek 5800 System](#) to store digital objects and datastreams.

For more information on this plugin, please visit <http://opensolaris.org/os/project/honeycomb/>

Rebuilder Support

Fedora's Rebuild Utility ([fedora-rebuild](#)) provides repository administrators with an automated way to reconstitute Fedora's higher-level indexes (the SQL database and the Resource Index) when upgrading Fedora, migrating to another SQL database, or recovering from inconsistencies.

Historically, the only Low Level Storage implementation that supported this utility was the Filesystem Low Level Storage module that came bundled with Fedora. As of Fedora 3.2, the rebuilder now works with any LLStore implementation that supports the new [IListable interface](#).

Currently, the only LLStore implementations that support this interface (and thus can be used with the rebuild utility) are the Akubra and Filesystem LLStore modules released with Fedora 3.2.

HTTP Proxy Configuration

Http Proxy Usage

For networks within an organization, the access to the public domain Internet is often routed through a web proxy and therefore only accessible by using this proxy. Fedora >= v3.3 allows the usage of a http proxy. This means that you can for example have objects with external referenced content that is only reachable through a proxy server.

Proxy properties

There are five properties that can be used for configuring the http proxy:

Parameter	Description
<code>http.proxyHost</code>	The hostname or IP address of the machine the proxy server is running on.
<code>http.proxyPort</code>	The port of the proxy. Defaults to 80.
<code>http.nonProxyHosts</code>	A list of hosts that should be reached directly, bypassing the proxy. This is a list of regular expressions separated by ' '. Any host matching one of these regular expressions will be reached through a direct connection instead of through a proxy. If the nonProxyHosts list is invalid, the Fedora web client dismisses it, writes the error to the logfile and continues its operation.
<code>http.proxyUser</code>	Should the http proxy require credentials in the form of username and password, you must provide them with these properties. Note: if the proxyUser is set the proxyPassword must not be empty, otherwise the username is not being used by Fedora's web client.
<code>http.proxyPassword</code>	The password for the http proxy.



Note

Note: https proxies are currently not supported.

For more detailed information see [the official java documentation](#).

Tomcat Setup

The proxy setup can be done in basically two ways. Either by using the JAVA_OPTS environment variable:

```
JAVA_OPTS="-DproxyHost=host -DproxyPort=3128 -DnonProxyHosts=host1|host2 -DproxyUser=fedora -DproxyPassword=secret"
```

Or by supplying these values directly to Tomcat (`$CATALINA_HOME/bin/catalina.sh` for *nix or `%CATALINA_HOME%/bin/catalina.bat` for Windows). Please consult the official Tomcat documentation for further details.

Installation From Source

Installation from Source



Fedora now builds with Maven

As of version 3.3, Fedora is now built with Maven instead of Ant. This document describes how to build Fedora 3.3 from source -- instructions for building Fedora 3.2.x are still available [here](#).

To build the installer, at a command prompt, enter:

```
mvn clean install -P fedora-installer
```

Once the build has been run with the fedora-installer profile (-P), you will be able to run the Fedora installer application. To start the installer, at a command prompt, enter:

```
java -jar fcrepo-installer/target/fcrepo-installer-VERSION.jar
```

Please ensure that the user account that is running the installer has sufficient permissions to write to the directories where Fedora will be installed (if deploying to an existing Tomcat installation, this includes permissions to the Tomcat directory). For more information on the Installer, see the [Installation](#) instructions in the [Installation and Configuration Guide](#).

Other useful build targets in the source distribution include:

1. mvn clean install
 - builds all source code
 - runs all unit & integration tests
2. mvn install -Dintegration.test.skip=true
 - runs all unit tests
 - skips all integration tests
3. mvn install -Dmaven.test.skip=true
 - skips all unit tests
4. mvn integration-test -P config[A|B|C|Q]
 - runs system tests per given configuration. For details on what each system test does, see the README at the root of the source distribution.

[Back to Installation and Configuration Guide](#)

Installing Java

This page is part of the [Installation and Configuration Guide](#).



The Fedora Repository release 3.4 requires Java 6 (JDK 1.6)

The Fedora Repository needs JDK 1.6 to be installed on your computer. JDK 1.5 (and below) is no longer supported. This release has only been tested with the Sun JDK though other distributions may work.

- A JRE (Java Runtime Environment) is not enough.
- JDK 1.6 is the supported platform and Fedora has been tested against this platform.
- JDK 1.5 is no longer supported in Fedora Repository 3.4 or later.

1. If you are not sure whether you have JDK installed correctly, please confirm by doing the following:
 - Open a command prompt.
 - On Windows: Open your 'Start' menu and select 'Run', then type `cmd` and click 'OK'. Alternately, you can open a Command Prompt window from the Accessories menu item.
 - Type the following in the command prompt and then press Enter:
 - On Windows: `echo %JAVA_HOME%`
 - On Unix: `echo $JAVA_HOME`
 - View the result:
 - If a line is displayed such as `C:\Progra~1\Java\jdk1.6.0_17`, please check that the letters just before the final numbers are '**jdk**'. If you see those letters, the JDK is installed.
 - If nothing is displayed, or you do not see '**jdk**' plus some numbers, the JDK is not installed.
2. If you need to install the JDK, follow these instructions:
 - Go to the [Java Sun download page](#).
 - Download the version entitled 'JDK 6 Update XX', where 'XX' stands for some number. (Sun will provide the latest version on that page.)
 - When the download has finished, run the Java installer. At one point, you will be asked to choose a directory to install to. Copy or write this directory down for use later.




Java 6

The Fedora Repository can be compiled using the Java SE 6 JDK. The Sun Java 1.5 has been set to EOL (end of life) in 2009 by Sun Microsystems. As of release 3.4 Fedora has moved to Java 6 as the standard platform and Java 5 is no longer supported. We would appreciate feedback from installations using Java EE application servers.

3. On Windows: Please follow [these instructions](#) to set your `JAVA_HOME` environment variable to the directory you where you have just installed the JDK. By default, this directory is under `C:\Program Files\Java`. Please note that some scripts on Windows may not work correctly from the Windows command line when directories having spaces in their names are encountered. Usually, putting the

command in double quotation marks will fix the problem. The JDK can also be installed anywhere in the file system such as `C:\Java` avoiding spaces in directory names.

Setting JAVA_HOME in Windows

 This information is only relevant if you are installing the Fedora Repository on a Windows server.


After you have installed the [Java Development Kit](#) in Windows, you must set the `JAVA_HOME` variable to the installation directory.



Please check that you have a JDK or SDK — Java JRE is not enough


A common problem is that people have only installed the Java Runtime Environment (JRE). You need either a Java Development Kit (JDK) or J2SE Software Development Kit (SDK). To confirm that you have the right version, you can check the Java installation path. Unless you changed the path during installation, Java will be installed to a subdirectory under `C:\Program Files\Java`, for example `C:\Program Files\Java\jdk1.5.0_17`


Open `C:\Program Files\Java` and confirm the installation path is for a JDK or SDK. JRE installations are *not suitable*, and have an installation directory beginning with `jre`. The numbers after the `jre` are not relevant. Example JREs are:

 `jre1.5.0_16`

 `jre6`

SDK and JDK installations are suitable. Their installation directory begins with `jdk` or `j2sdk`, the numbers at the end are not relevant. Example JDK and SDKs are:

 `jdk1.5.0_17`

 `jdk1.6.0_07`

If you cannot see an installed JDK or SDK, install the JDK now (see [Installing Java](#)). It is perfectly fine to leave your currently installed JRE in place. Windows will generally use its registry to support Java functions. You can install an additional JDK within `C:\Program Files\Java` or in a separate directory for use with the Fedora Repository. We will set the `JAVA_HOME` environment variable to provide compatibility.

Step 1. Locate the JDK Installation Directory

If you already know the installation path for the Java or Software Development Kit, go to *Step 2* below. Otherwise, find the installation path by following these instructions:

1. Unless you changed the installation path for the Java Development Kit during installation, it will be in a directory under `C:\Program Files\Java`. Using File Explorer, open the directory `C:\Program Files\Java`.
2. Inside that path will be one or more sub-directories such as `jdk1.5.0_17`. If you have just installed the Java Development Kit, it will be installed to the newest directory, which you can find by sorting by date. For example, it may be installed in `C:\Program Files\Java\jdk1.5.0_17`. This is the installation path.

Step 2. Set the JAVA_HOME Variable

Once you have identified the JDK installation path:

1. Right-click the **My Computer** icon on your desktop and select '**Properties**'.
2. Click the '**Advanced**' tab.
3. Click the '**Environment Variables**' button.
4. Under '**System Variables**', click '**New**'.
5. Enter the variable name as `JAVA_HOME`.
6. Enter the variable value as the installation path for the Java Development Kit.
7. Click '**OK**'.
8. Click '**Apply Changes**'.
9. If you are running the Fedora Repository as a WAR rather than the Standalone, you may need to restart your application server.

This diagram shows setting the `JAVA_HOME` variable to an installation path of `C:/Java/jdk1.5.0_12`:



If you came here from Installing Standalone, go back and begin *Step 3*.

RELATED TOPICS

Installing Java

Upgrade Guide

Migration Guides

- [Upgrading from 2.x](#)
- [Upgrading from 3.x](#)

Fedora 3 Migration Support

If you are still using a 2.x Fedora installation, it is highly recommended that you move to 3.2.

This page is meant to be a starting point to provide extra information to ease the migration process from Fedora versions 2.x to versions 3.x. We are enlisting your participation in creating a community of support to not only ensure successful migration experiences, but to provide meaningful feedback and patches to help improve the migration utilities software.

- If you have recently migrated to 3.x, you are encouraged to note your experiences, successes 😊, and otherwise 😞 below.
- If you are planning (or would like to start planning) a migration, please do not hesitate to let us know. Users are encouraged to add questions, resources or their contact information if they are willing to help. This information is all meant to supplement the official migration guides listed above.

Contacts

The following individuals have volunteered to share their experience and try to answer questions and offer support for those having difficulty migrating their repository. If information cannot be found here, or in the mailing list archives, users are encouraged to e-mail the mailing list to maximize the number of individuals that can help and so that others may benefit from the exchange.

- [Michael Durbin](#)

Migration FAQ

Many questions about migration have been posted to the Fedora mailing lists. The most common themes are linked below, though a full listing or search can be performed here: http://sourceforge.net/mail/?group_id=177054

Questions encountered when migrating from 2.x to 3.2

- none yet

Questions encountered when migrating from 2.x to 3.1

- What do I do when the Analyzer/Generator/Transformer fails because of an exception "Caused by: java.lang.NoClassDefFoundError: org/jrdf/graph/PredicateNode"?
 - [Mailing list thread](#)

- [Mailing list question](#)
- What do I do when the analyzer fails because duplicate PID?
 - [Mailing list thread](#)

Experiences, Walkthroughs and Information

- [Chris Wilper's Blog posts about Fedora](#) (includes an overview of the new CMA)
- [Indiana University Digital Library Program's Migration Experience](#)
- [Open Repositories 2009 presentation: Fedora 3: A Smooth Migration](#)
- [Survey of Experiences](#)
 - results pending

Discussion and advice

Users are encouraged to add their own stories, problems, or advice here.

Upgrading from 2.x



Upgrade utility not currently compatible with version 3.4.2

The upgrader utility is not currently compatible with version 3.4.2 of Fedora, due to the Java package renaming in this release.

This issue is currently being addressed, see [the JIRA issue](#). We will announce on the mailing lists when a new version of the utility is available. In the meantime, you must upgrade to Fedora 3.3 as described below, then follow the instructions on [Upgrading from 3.x](#).

Overview

This document explains how to migrate your Fedora repository from version 2.x to 3.x. Although it is written assuming 3.3, the same instructions apply if you are upgrading to an older 3.x release. Before continuing, you should familiarize yourself with the new [Content Model Architecture](#). A basic understanding of how the CMA works will be helpful in understanding the migration process.

Throughout this guide, `OLD_FEDORA_HOME` refers to the home directory of your Fedora 2.x installation, and `NEW_FEDORA_HOME` refers to the home directory of your new Fedora 3.x installation.

NOTE: To reduce confusion during this process, if you have previously ingested the Fedora demo objects, you should purge them from your 2.x repository before starting with the migration process. The demo objects have changed significantly since 2.x, and if you'd like the new versions, we recommend that you ingest them after performing a successful migration of your old repository.

1. Install, Start, and Stop Fedora 3.x

Follow the instructions in the Fedora [Installation and Configuration Guide](#). When finished, start the server and verify your installation was successful by visiting the `describeRepository` page (e.g. <http://localhost:8080/fedora/describe>). Finally, shut down Fedora. It should not be restarted again until later in the migration process.

2. Point Fedora 3.x to Existing Objects

A. Object XML (FOXML)

Determine where your old object XML is stored. If you're unsure, consult `OLD_FEDORA_HOME/server/config/fedora.fcfig` and find the value of `object_store_base`.

Edit `NEW_FEDORA_HOME/server/fedora.fcfig` and change the value of `object_store_base` the *full path* of your old object's directory.

NOTE: The upgrade process will transform the objects in `object_store_base` of `NEW_FEDORA_HOME/server/fedora.fcfig` in place. Thus, by following the above instructions, the new upgraded files will exist in their old location. If that is not what is desired, then you could copy the old files (both objects and content) to their final location, point the `NEW_FEDORA_HOME/server/fedora.fcfig` to that new location, and then continue the update process.



Warning

If you haven't already done so, make a backup of all original object XML files. These will be changed during the transformation part of the migration process, and if something goes wrong, this backup will be your only way of recovering.

B. Managed Datastreams


Determine where your old Fedora Repository's managed Datastreams are stored. Again, you can check your old `fedora.fcfg` file. Look for the value of `datastream_store_base`.

Edit `NEW_FEDORA_HOME/server/fedora.fcfg` and change the value of `datastream_store_base` the *full path* of your old Datastreams directory.

NOTE: The migration process will not make any changes to these files.

2. Install the Migration Utilities

The migration utilities come as a separate download from the Fedora installation. They are all included, with source, in a single download [fedora-migration-3.2.zip](#) at [sourceforge.net](#). Once unzipped, you will find the executable jars in the top directory.

 Although the filename is `fedora-migration-3.2.zip`, the utilities will also work for migrating to Fedora 3.3

3. Run the Analyzer


The analyzer examines all of your existing digital objects and looks for similarities. It outputs the following information in a directory you specify.

- A set of generated content models.
- A list of PIDs for each content model.
- Service Deployment (formerly known as Behavior Mechanism) information for each content model.

The analyzer does not make changes to any source objects.

A. Configuring the Analyzer

The analyzer accepts a Java properties file for configuration.

 **Tip**
In properties files, the `"\"` character must be escaped. When using Windows, this means paths like `c:\work\abc` must be written with two backslashes as a path delimiter rather than one.

Create a file (e.g. `migration.properties`) with the following content, filling in values appropriate to your environment:

```

# This is the directory where the analyzer's output files should be sent.
# If it doesn't already exist, it will be automatically created.
# If it already exists, it must be empty (to avoid accidental overwrites)
# To disable the above restriction, uncomment clearOutputDir=true below.
outputDir=c:\\fedora-migration
#clearOutputDir=true

# The Fedora 3.x home directory.
fedoraHome=c:\\fedora-3.3

# The full path to the JDBC driver Fedora is configured to use.
# NOTE: The analyzer only uses the database to aid in looking up
# the location of FOXML. It will populate the initial paths in
# the database the first time it runs, if the necessary.
jdbcJar=c:\\fedora-3.3\\tomcat\\webapps\\fedora\\WEB-INF\\lib\\postgresql-8.3-603.jdbc3.jar
# Aspects of the original objects to ignore for the purpose of
# classification. This is optional. If unspecified, the generated
# content models will be the MOST SPECIFIC POSSIBLE. If
# specified, this property must consist of a space-delimited
# list of any of the following:
# OrigContentModel
# If specified, objects that have differing values in the original
# contentModel property may be assigned to the same content
# model if they are otherwise similar.
# DatastreamIDs
# If specified, only datastreams bound to disseminators will
# be considered important for classification. Objects that have
# differing sets of UNUSED datastream IDs may be assigned to
# the same content model if they are otherwise similar.
# MIMETypes
# If specified, the MIMETYPE of each candidate datastream
# will be ignored for the purpose of classification. Objects that
# have differing MIME types for the same datastream may be
# assigned to the same content model if they are otherwise
# similar.
# FormatURIs
# This works exactly the same as MIMETypes, but applies to
# the FORMAT_URI of candidate datastreams.
#ignoreAspects=OrigContentModel DatastreamIDs MIMETypes FormatURIs

# Specific datastream IDs to ignore for the purpose of classification.
# This is optional. If specified, this property must consist of a
# space-delimited list of datastream IDs to ignore. Note: This configuration
# has no effect if DatastreamIDs is already specified as an ignoreAspect
# above.
ignoreDatastreamIDs=DC RELS-EXT RELS-INT POLICY

# Explicitly declare objects to be in the FedoraObject-3.0 content model.
# The default is 'false', or implicit. If left implicit, the objects
# will not have an explicit basic model, and it will be up to the
# system to use a default value at runtime. This option may have
# an impact on future upgrades. Future versions of Fedora may adopt a new
# basic model that has additional system methods, or require certain
# datastreams or formats. Objects that explicitly declare a 3.0
# model should behave exactly the same if Fedora is upgraded to a
# post 3.0 version. If left implicit, the objects may be interpreted
# in light of the new model, and may inherit new methods, or may
# fail validation and require updating if the new model introduces
# requirements they do not fulfill.

# Uncomment to force explicit basic model declarations in the
# upgraded objects.
# explicitBasicModel = true

```

B. Analyzer Usage

The analyzer utility is an executable jar and takes the configuration file as a parameter. For example, if your configuration is in the current directory and is named migration.properties, enter the following:

```
java -jar analyzer.jar migration.properties
```

Analysis of small repositories will finish very quickly. For repositories with millions of objects, analysis will take several hours. With modern hardware, expect a rate of about 10,000 objects per minute.



Information

If upgrading to Fedora 3.3 with an embedded Derby database (a configuration that is not recommended for production), you may notice an error similar to "java.sql.SQLException: Cannot close a connection". This does NOT indicate that the analyzer failed, and can be safely ignored.

C. Reviewing Analyzer Output

The analyzer will produce several files in the output directory.

Generated content models will be in FOXML 1.1 format and will be named `cmodel-n.xml` (where `n` is a number used for association). For each of these, there will be an associated `cmodel-n.members.txt` file containing a list of PIDs that conform to the content model.

Each content model object will contain the following inline XML datastreams:

- **CLASS-DESCRIPTION** - This is a simple human-readable log of the matching aspects of all member objects found during the analysis process. This datastream is not used or recognized by Fedora; it is only included for documentation purposes and may be removed at any time.
- **DS-COMPOSITE-MODEL** - This is a special Fedora-defined description of the datastreams (and aspects thereof) that member objects are expected to have. You'll notice that it contains a subset of the information expressed in CLASS-DESCRIPTION. This datastream is important, and should not be removed from the content model object.

If the member objects had disseminators:

- You will notice that the generated content model also has a `RELS-EXT` Datastream. This RDF Datastream points to the original Behavior Definition (now known as Service Definition) objects via a Fedora-defined `fedora-model:hasService` relationship. This relationship means that members of this content model should have the behaviors defined within the target SDef(s).
- There will also be an associated `cmodel-n-deployments.txt` file in the output directory. For each Behavior Mechanism formerly used by the objects' disseminators, this file identifies the original BMech, specifies a PID for a new, similar Service Deployment, and specifies a set of Datastream input name changes that the copy should have. This information is used by the generator to create new content-model-specific Service Deployment objects.

Three additional files will be created in the output directory:

- **sdefs.txt** - This lists all original Behavior Definition (now known as Service Definition) objects. The generator will create a stylesheet to upgrade these objects to FOXML 1.1.
- **sdeps.txt** - This lists all original Behavior Mechanism objects. Although these objects will be made obsolete by the new generated Service Deployments, the generator will create a stylesheet to upgrade them to Service Deployments in FOXML 1.1 so that you can view them in the Fedora 3.0 Repository before deciding to purge them.
- **nocmodel.txt** - This lists objects that should NOT be assigned a content model. Initially, the file is empty but it can be customized, if desired (see [Upgrading Objects Without Content Models](#) below).

D. Customizing Content Model PIDs

By default, the PIDs assigned to the generated content models are of the form, `changeme:CMODELN`. You should change these to use your own repository's namespace. You may also want to change the identifier part of the PID (e.g. `myns:Journal`).

To do this, open each `cmodel-n.xml` file in an editor and change the following as desired:

- The value of the PID attribute at the root of the document.
- If there is a `RELS-EXT` Datastream, change the part after `info:fedora/` in the `rdf:about` attribute. For example, change `rdf:about="info:fedora/changeme:CMODEL1"` to `rdf:about="info:fedora/myorg:Journal"`.

E. Customizing Service Deployment PIDs

If any `cmodel-n-deployments.txt` files were created by the analyzer, you should also change any `NEW_DEPLOYMENT` pid values specified within each before continuing. These PIDs will default to values like `changeme:CMODEL1-BMECH1`, but again, you should specify your own (e.g. `myns:Journal-DefaultImpl`).



Warning

DO NOT specify the same PID for `NEW_DEPLOYMENT` as `OLD_BMECH`, as this will cause ingest problems later.

F. Upgrading Objects Without Content Models

By default, the analyzer ensures that every data object is assigned to a content model. If you'd rather avoid assigning an explicit content model to an object, you may do so by a) removing its PID from the `cmodel-n-members.txt` file in which it resides and b) adding it to the `nocmodel.txt` file. This may be done for any number of objects.



Warning

In order for migration to work properly, the PID of every object in the source repository must occur exactly once in the set of PID list files.

5. Run the Generator

The generator reads the output of the analyzer (along with any customizations you have made) and adds the following to the same directory:

- New Service Deployment objects (as specified in each `cmodel-n-deployments.txt` file)
- Stylesheets for transforming existing objects as necessary



Information

The generator does not make changes to the source objects.

A. Configuring the Generator

Like the analyzer, the generator accepts a Java properties file for configuration.

The configuration file should have the following content, with values filled in appropriate to your environment.



Tip

Since you already entered `fedoraHome` and `jdbcJar` in `migration.properties`, you can minimize typing by just using that file, and adding the necessary `sourceDir` parameter as shown below.

```
# This is the directory containing the analyzer's output, and where the generator's
# results should be written.
sourceDir=c:\\fedora-migration

# The Fedora 3.x home directory
fedoraHome=c:\\fedora-3.3

# The full path to the JDBC driver Fedora is configured to use
jdbcJar=c:\\fedora-3.3\\tomcat\\webapps\\fedora\\WEB-INF\\lib\\postgresql-8.3-603.jdbc3.jar
```

B. Generator Usage

The generator utility also accepts the configuration file as a parameter. For example:

```
java -jar generator.jar migration.properties
```

The generator should finish very quickly, regardless of the size of the repository.

C. Reviewing Generator Output

The generator will produce several files in the output directory. These files, along with those already produced by the analyzer, will be used in the next steps of the migration process.

One stylesheet will be written for each PID list: `sdeps.xslt`, `sdefs.xslt`, `nocmodel.xslt`, and each `cmodel-n.members.xslt` file. Each of these stylesheets will include transformation rules for upgrading the objects to FOXML1.1 and adding the necessary `fedora-model:hasModel` relationship via `RELS-EXT`, if necessary.



Information

The generated stylesheet will cause a new `RELS-EXT` Datastream to be created, or will amend the content of the latest revision of the `RELS-EXT` Datastream, if it already exists.

The output will also include a new, generated Service Deployment object (`cmodel-n.deploymentN.xml`) for each one listed in the input's `cmodel-n.deployments.txt` files. This SDep will match the content of the source SDep in the repository, but it will have a different PID, will be in FOXML 1.1 format, and will use input part names consistent with the Datastream IDs specified in the associated content model's DS-COMPOSITE-MODEL Datastream.

6. Run the Transformer

The transformer applies stylesheets to FOXML stored in a Fedora repository. Although it directly accepts the output of the analyzer and generator, it can also be used outside of the migration process for making low-level changes to batches of objects. When running the transformer in this way, the repository should be shut down, and the `rebuilder` should be run immediately afterward (see section below).

Here's how the transformer works: It scans a directory for PID list files, ending with `.txt`. For each, if a `.xslt` file exists with the same name, that stylesheet is applied to the repository's FOXML for each object in the PID list.



Warning

The transformer makes changes to object XML in the repository. It is possible to damage some, or all of the objects in the repository by using the transformer, so be sure to make a backup first!

A. Configuring the Transformer

Like the analyzer and generator, the transformer accepts a Java properties file for configuration. The transformer is configured with the same properties that the generator is configured with (see above), and also takes the following:

```
# Whether to run the transformer in "dry run" mode or not.
# In dry run mode, transformation will be tested but no changes will be written
dryRun=true
```

Although not required, it is strongly recommended that you run the transformer with `dryRun=true` the first time, to ensure all transformations will fully succeed.

B. Transformer Usage

The transformer utility also takes the configuration file as a parameter. For example:

```
java -jar transformer.jar migration.properties
```

Transformation will take roughly double the time that analysis took, since it must read and write each file in the repository.

7. Run the Rebuilder

Now that all of your existing objects have been upgraded, you need to run the `rebuilder` so Fedora's database is up-to-date with the files on disk. See the [Rebuilder](#) documentation for further details.



Information

You **MUST** rebuild the SQL database. Rebuilding the Resource Index is required only if your Fedora 3.x Repository has been configured to enable the Resource Index.

Rebuilding the SQL database will take around double the time that the transformer took. Rebuilding the Resource Index may take significantly longer.

8. Ingest Generated Objects

After the rebuilder has run successfully, you should restart your Fedora 3.x instance and visit the describe page (e.g. <http://localhost:8080/fedora/describe>) again to make sure it started successfully.

Now you'll need to ingest all the new Content Model and Service Deployment objects created during the analyzer and generator steps. To do this:

- Run `fedora-admin.bat` or `fedora-admin.sh` and login to your new repository from the local machine.
- Go to File -> Ingest > Objects from Directory, and choose directory where these files were written (e.g. `C:\fedora-migration`)
- Choose FOXML as the format, and select OK.



Information

The admin client may try to ingest all files in the directory, including the non-FOXML files. While harmless, you may wish to copy all FOXML files (all files in that directory that end in .xml) to a separate directory, and ingest from there.

10. Verify Success and Clean Up

You should now verify that the upgraded objects are behaving as expected. Pick a few from each `cmodel-n.members.txt` file and do the following for each:

- Visit the object's "Object Profile" page (e.g. <http://localhost:8080/fedora/get/demo:MyPID>)
- From that page, navigate to "View Item Index", and click each Datastream, ensuring it downloads properly.
- If you had Disseminators in your original repository, you should also navigate to the "View Dissemination Index" page and make sure your old Disseminations appear, and that they execute properly.

If you had Disseminators in your original repository, you are now making use of a NEW set of Service Deployment objects. You may now want to purge the original BMechs, since they are no longer in use. The simplest way to do this is with the `fedora-admin` GUI. The PIDs of the old former BMechs are enumerated in the file `sdeps.txt`.

Upgrading from 3.x

Upgrading from 3.4.1 to 3.4.2

Version 3.4.2 is a bugfix release and does not require a database or resource index rebuild, nor does it require an upgrade to your configuration. In order to "swap in" the new version of the software, you may:

1. Shut down your old 3.4.1 repository
2. Install the newer version of Fedora in a different location, but before starting it:
 - a. Copy your old `server/config/` directory into the new installation's `server/config/` directory.
 - b. If you are using the legacy `llstore` implementation instead of Akubra, modify the `fedora.fcfg` file, ensuring that the `object_store_base` and `datastream_store_base` values point to the absolute path of these existing directories. If you'd rather keep the paths relative in the config file, move or copy the content to the matching location in the new installation instead.
 - c. If you have previously made changes to the repository-wide XACML policies, copy them into the new repository installation's `data/fedora-xacml-policies` directory (you will need to create this directory)
3. Start the new instance of Fedora for the first time.

Upgrading from 3.x to 3.4.2



Akubra low-level storage

As of Fedora 3.4, Akubra is the default low-level storage implementation. Akubra is not backwards-compatible with the datastream and object storage from previous releases, so when upgrading ensure you select the legacy-fs option for Low level Storage. If you are using an `install.properties` file ensure you specify the `llstore.type=legacy-fs` property. A migration utility to migrate from legacy low-level storage to Akubra will be provided in a future release.

1. Shut down your old 3.x repository
2. Install the newer version of Fedora in a different location, but before starting it, modify the new `fedora.fcfg` so that:
 - a. It points to your previous 3.x database
 - b. The object and datastream paths point to your previous 3.x locations
 - c. Note: Due to the Mulgara version upgrade, if you have enabled the Resource Index previously, it will need to be rebuilt. Therefore, it is unnecessary to point the resource index configuration to the old location.
3. Start the new repository for the first time.
4. Shut down the new repository.
5. If you have previously made changes to the repository-wide XACML policies, copy them into the new repository's XACML directory.
6. If you previously enabled messaging, and there were messages from your old repository that have not yet been delivered, copy its `activemq-data` directory over the new `activemq-data` directory in your new install.
7. Run the Resource Index `Rebuilder`.
8. Restart the new repository

License and Copyright

Fedora Repository Software License

Copyright © 2008-2009 Fedora Commons, Inc.
Copyright © 2002-2007 The Rector and Visitors of the University of Virginia and Cornell University

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License.
You may obtain a copy of the License at: <http://www.apache.org/licenses/>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

The Fedora Repository Software distribution includes several third-party libraries, each with their own license terms. For a complete copy of all copyright and license terms, including those of third-party libraries, please see:

- [Fedora License and Copyright Page](#)

Distribution Structure

Introduction

All of the directories named below are relative to `FEDORA_HOME`.

- **server** - Contains the server configuration, command-line utilities, and policy files.
- **client** - Contains the client classes, libraries, and utilities.
- **data** - The default parent directory for `object_store_base`, `datastream_store_base`, and XACML policies.
- **docs** - Contains the licenses, javadocs, and Fedora documentation.
- **derby** - If the included Derby database was chosen during the installation (i.e. either a "quick" install or the "included" database was selected as part of a custom install, this will contain the configuration and data for the embedded Derby instance.
- **tomcat** - If the "quick" option was selected during the installation, this will contain an installation of Tomcat 5.5.26.

Server

- **server/access** - Stylesheets for the default dissemination service. All objects in the repository automatically subscribe to this service through the system default content model. This service provides operations for listing and retrieving the components of the object.
- **server/bin** - Contains server command-line utilities, including `fedora-rebuild`, `fedora-reload-policies`, and `validate-policy`.
- **server/config** - Contains the server configuration files, including `fedora.fcfg`, `fedora-users.xml`, and `log4j.properties`.
- **server/logs** - Contains the server log files.
- **server/schematron** - Files supporting the validation of serialized digital objects.
- **server/xsd** - W3 XML schemas, for reference and validation support.

Client

- **client/bin** - Contains client command-line utilities, including `fedora-admin`, `fedora-convert-demos`, `fedora-ingest-demos` and various batch utilities.
- **client/demo** - Contains sample digital objects that can be used as examples.
- **client/fedora-client.jar** - Contains the classes necessary to run the client applications.
- **client/log4j.xml** - Log4J configuration file for client applications.

Data

- **data/objects** - The default root directory (defined in `fedora.fcfg`) for the internal storage of Fedora objects.
- **data/datastreams** - The default root directory (defined in `fedora.fcfg`) for the internal storage of Managed Content datastreams.
- **data/fedora-xacml-policies/repository-policies** - The default directory (defined in `fedora.fcfg`) for repository-wide XACML policies.

Docs

- **docs/license** - Contains license documentation for Fedora and third-party licenses and attributions.
- **docs/javadocs** - Javadoc API documentation for Fedora.

Demonstration Objects

About

This document describes the demonstration objects that are distributed with Fedora.

After installing Fedora, you'll find these objects, in several formats, in your `$FEDORA_HOME/client/demo` directory.

These objects can be ingested into the repository in one of two ways.

- Using the [Fedora Administrator GUI](#), selecting File, Ingest, Multiple Objects, From Directory and pointing to the demo/foxml directory.
- Running the `fedora-ingest-demos` command line script.

Once ingested, the demo objects can be viewed in a web browser using API-A-LITE. For example, to view the **demo:5** object:

```
http://localhost:8080/fedora/get/demo:5
```

All demo objects are intended to work when the Fedora repository server is in a stand-alone condition (e.g., if the repository is running without a network connection, or if the repository is behind a firewall and not set up to receive outside connections)

Simple Document Demo

This Fedora data object **demo:18** demonstrates the simplest Fedora digital object scenario. It is the case where we aggregate content in the Fedora object, and let Fedora's default object behaviors provide access to the content. This is an example of a Fedora digital object that only has default dissemination services. In this case, there are 3 datastreams in the object, one for each format of a particular document (in this case the Fedora paper presented at ECDL2002). We can use the basic Fedora object dissemination service (also called "datastream disseminations") which are part of the basic content model shared by all objects. The basic content model is dynamically associated with every object in the repository (though it may optionally be statically associated). It has a default service definition (sDep) which provides basic operations for every object which includes the ability to list items in the object, get an item, get the dissemination index, get the Dublin Core record, and retrieve other information about the object. The results of these operations can be returned as either HTML (method names begin with "view...") or XML (method names begin with "get..."). The end result is that the object is simply a container for content and metadata. The user can view the contents and get any item from the object. While this scenario may be easy to implement and useful, it does not take advantage of Fedora's extensible service features where custom operations can be associated with an object.

Formatting Objects Demo

There are two demonstrations of using Fedora to display XML content styled using XSL Formatting Objects. First, the Fedora data object **demo:21** shows the transformation of native formatting object document stored as an inlined XML datastream into PDF. Second, the Fedora data object **demo:26** shows the use of formatting objects to process TEI documents.

Simple Image Demo

The Fedora data object **demo:5** demonstrates the UVA Simple Image behaviors by associating a simple dissemination with the object through its content model. There are 4 Datastreams in the object, one for each of four different image resolutions. The object is linked to one dissemination service which provides four behavior methods: `getVeryHigh`, `getHigh`, `getMedium`, and `getThumbnail`. The fulfillment of the service contract entails the Fedora HTTP Image Getter resolving the URL of the appropriate datastream for each of the UVA Simple Image behaviors. There are no transformations performed on the datastreams. This object shows how a service definition can be used to create a normalized set of methods for a particular type of object, an image object in this case, which is defined by a content model. The idea here is that the Simple Image service definition provides a standard set of dissemination services that can be used on any image object that conforms to the standard image content model. As we will see later, different variants of image objects can subscribe to the same service definition, and in some cases the datastreams will be dynamically transformed by a service to provide the appropriate image disseminations. This demo shows a simple one-to-one mapping of the datastreams in the object to the behavior methods.

Document Transformation Demo

The Fedora data object **demo:14** demonstrates the Document Transformation behaviors. There are 3 datastreams in the object, one XML source document, and two XSLT stylesheets. The object's content model provides one dissemination service which is associated with the "Document Transform" service definition and the Fedora Local Saxon Service (service deployment). Two services are available: `getDocumentStyle1` and `getDocumentStyle2`. When these methods are run the repository mediates access to the Fedora Local Saxon Service to produce the appropriate transformation on the XML source in the object. The dissemination result will be one of two document styles.

Image Collection Demo

This demo illustrates the use of the Resource Index search service to fulfill collection behaviors. *For this demo to work, the [Resource Index](#) must be enabled prior to ingesting these objects.*

A series of data objects (`demo:SmileyBucket`, `demo:SmileyKeychain`, etc.) subscribe to the image behaviors defined by the sDef object `demo:DualResImage`. Each of these image objects also use the RELS-EXT datastream to assert its membership in the `demo:SmileyStuff` collection. The `demo:SmileyStuff` collection subscribes to sDef object `demo:Collection`, which defines two methods: `list` and `view`. The collection object uses the `demo:DualResImageCollection` sDep to implement those behaviors.

To see the dynamic HTML listing of collection members in action, you can view <http://hostname/fedora/get/demo:SmileyStuff/demo:Collection/view>.

This dissemination first requests the list of members of the `demo:SmileyStuff` collection using the local [rsearch](#) service. Then it uses the local [saxon](#) service to transform the XML results into a human-readable HTML page. The query text and the stylesheet are both datastreams of the

SmileyStuff collection and act as inputs to the list and view behaviors, respectively.

Getting Started with Fedora

Getting Started with Fedora

- [The Fedora Basics](#)
- [Using the Fedora Repository Software](#)
- [Search and Discovery](#)
- [Tutorials](#)
- [Fedora-based Applications](#)

The Flexible Extensible Digital Object Repository Architecture is a conceptual framework that uses a set of abstractions about digital information to provide the basis for software systems that can manage digital information. It provides the basis for ensuring long-term durability of the information, while making it directly available to be used in a variety of ways. It is very important to understand that Fedora provides a foundation upon which to build a variety of information management schemes for different use cases, not a full solution for a specific use case. The Fedora software that DuraSpace distributes has been designed to provide many different possibilities for a large array of applications.

Fedora has a very active developer community, both contributing to the core software development process and developing complete applications on top of Fedora that address particular use cases or application areas. This guide is designed to give you a basic understanding of the Fedora architecture and the core repository management software, and to give you some general ideas about how to use it. Whether you want to look at adopting one of the existing Fedora-based solutions or develop your own, this general introduction should be useful to you.

The Fedora Basics

In a Fedora repository, all content is managed as data objects, each of which is composed of components ("datastreams") that contain either the content or metadata about it. Each datastream can be either managed directly by the repository or left in an external, web-accessible location to be delivered through the repository as needed. A data object can have any number of data and metadata components, mixing the managed and external datastreams in any pattern desired.

Each object can assert relationships to any number of other objects, providing a way to represent complex information as a web of significant meaningful entities without restricting the parts to a single context.

Each data object is represented by an XML file that is managed in the file system, which contains information about how to find all of the components of the object, as well as important information needed to ensure its long-term durability. The system keeps an audit trail of actions that have affected the object, any formal policies may be asserted about the object and its use, and things like checksums, all within that XML file. As long as both the XML files and the content files that are managed by the repository are backed up properly, the entire running instance of the repository can be reconstructed from the XML files. There is no dependence upon any software to do so, no relational database that cannot be completely reconstructed from the files.

Fedora also provides a way to define any number of views of the digital object as a set of virtual datastreams or behaviors of the object, some of which can be created on the fly. This allows the object to present a set of virtual data products on the front end that are derived from the actual data that is being managed on the backend.

While Fedora can easily be used to model digital collections of surrogates of traditional, catalog-based collections, it has been designed to be able to support durable web-like information architectures. Because each object completely contains all of the content, metadata and attributes of a unit of content, and can assert any number of relationships to any other object, it is easy to support schemes in which objects have multiple contexts with no dependencies upon each other. A Fedora repository does not have a particular catalog. Any number of indices, designed for specific purposes can be applied to any pattern of components of objects desired.

- [The Fedora Abstractions](#) - A description of the essential abstractions defined by the Fedora architecture that are implemented in DuraSpace's Fedora software. (*coming soon*)
 - [An introduction to Fedora digital objects](#) - A detailed description of the four types of Fedora objects and what they can do.
 - [An introduction to FOXML](#) - The details about the XML encoding of Fedora objects.
 - [Relationships using RDF](#) - A description of how Fedora can use the Resource Description Framework (RDF) to represent object relationships.
 - [Content modeling](#) - A general introduction to using Fedora to manage different kinds of information. (*coming soon*)
-

Using the Fedora Repository Software

We provide a test repository instance that starts up your own instance of Fedora in the cloud. You can use this to play with the web-based administrator client to get a feel for making objects and managing a repository. Note that the instance of Fedora that is started up for you will stay active for one hour, at which time it will be terminated, removing all objects that you have created.

When you are first getting started with setting up a Fedora repository on your own machine, a quick start option is provided that makes it easy to

get going. This type of installation does not use any of the security features that Fedora provides, eliminating much of the complexity that often trips up new users. It is recommended that you start with that, then turn on the security features as you get comfortable with the basics.

There are two ways to create objects in your Fedora repository. You can use the web-based client to create them interactively one at a time, or you can construct your own workflows that create FOXML files which can be ingested into Fedora, either singly or as batch from a single directory.

When you are ready to start using service objects, we provide an application called EZService that makes it easy to create basic service objects from some XML template files. When you are ready to get more into the details, take a look at the Content Model Architecture (CMA) Construction Guide to get more of the fine details of what is possible.

When you are getting started with Fedora it is usually best to keep the security and policy enforcement functionality turned off. When you are interested in using that functionality, Fedora provides complete policy expression and enforcement systems that allow you to write policies that can be applied repository wide, to any object or any component of any object.

- [Use our test repository](#)- We provide instance of a Fedora repository running in the cloud that you can use without having to download and configure software.
 - [Installation and Configuration on your machine](#) - This is the starting point for setting up a Fedora repository on your server, which includes a quick start option.
 - [The Fedora Web-based Administrator](#) - You can use this to interactively create and update objects, as well as to administer your repository.
 - [Creating and Updating Data Objects](#) - The complete guide to building objects.
 - [EZService](#) - A quick start for creating service objects.
 - [The CMA Construction Guide](#) - A detailed guide to creating content model and service objects
 - [Security](#)
-

Search and Discovery

The nature of an object-based repository, like Fedora, is to manage all information in the most modular manner, in a way that is as independent of any particular software as possible. There is no database that holds metadata fields. There are services, such as GSearch and PrOAI, that harvest content and metadata from objects in various ways for various purposes. The best practice for building access systems for a Fedora repository is to use such services to build one or more indexes that are tailored to your needs.

There is a built-in search, Basic Search, that was included so that repository managers would have something to use to help them manage their repository. It indexes the required DC datastream, which is either an in-line XML datastream or a managed content datastream. If you provide this information as in-line XML it is best to keep this datastream as small as possible, as the data is actually stored in the FOXML file. If your FOXML files average larger than about 20 k in size, performance can be affected in situations where you have many simultaneous users. It is best to put basic info in the DC datastream that is useful in repository management, but not elaborate descriptive info. If you want a rich Dublin core record it is best to put it in a managed content datastream and index it using GSearch.

Both GSearch and PrOAI are designed to let you be selective in which objects you want included and to let you specify which datastreams you want to be included, either actual datastreams or virtual ones that result from a service call. GSearch lets you use one or more search engines that are already included to define different kinds of indexes. It also provides a way for you to create a plugin for your favorite search engine, if it is not already included. PrOAI lets you selectively expose your repository under the Open Archives Initiative (OAI) scheme.

- [Basic Search](#) - The built-in search that is intended for repository managers use, not intended to be exposed externally.
 - [GSearch](#) - The Generic Search Service that makes Fedora's features useful to different search engines.
 - [PrOAI](#) - The OAI provider service that is designed to take advantage of Fedora's features.
 - [The RDF-based Resource Index](#) - This is Fedora's built-in semantic store.
-

Tutorials

- [Tutorial 1 - Introduction to Fedora](#)
 - [Tutorial 2 - Creating Fedora Objects](#)
-

Fedora-based Applications

Below is a list of applications that run on the current 3.x versions of Fedora (or will soon be available). For a more complete community software registry that includes applications that run on earlier generations of Fedora, or are other useful tools and utilities, see our [Community Software Registry](#) .

- [ActiveFedora](#)- Built on RubyFedora, this ruby gem provides an active record oriented way of working with objects in Fedora
- [ESciDoc](#) - An eResearch environment developed specifically for use by scientific and scholarly communities.
- [The Fascinator](#) - A front end to Fedora commons repository that uses Solr to handle all browsing, searching, and security.
- [Hydra](#) - Will provide a "Lego Set" of web services and templates that can be used for a wide range of content management workflows. (Not released yet but coming soon.)

- [Islandora](#) - A Drupal module that allows users to view and manage objects stored in Fedora.
- [RODA](#) - An OAI-compliant, service-oriented digital repository system designed to preserve government authentic digital objects.

Fedora Digital Objects

[Fedora Digital Object Model](#)

[Fedora Identifiers](#)

[Digital Object Relationships](#)

[Content Model Architecture](#)

[CMA Construction Guide](#)

[Introduction to FOXML](#)

[Fedora Atom](#)

[Fedora METS](#)

[Ingest and Export](#)

[Portable Fedora Objects](#)

Fedora Digital Object Model

- [The Fedora Digital Object](#)
- [The Fedora Digital Object Model](#)
- [Datastreams](#)
- [Digital Object Model - Access Perspective](#)
- [Four Types of Fedora Digital Objects](#)
 - [Data Object](#)
 - [Service Definition Object](#)
 - [Service Deployment Object](#)
 - [Content Model Object](#)

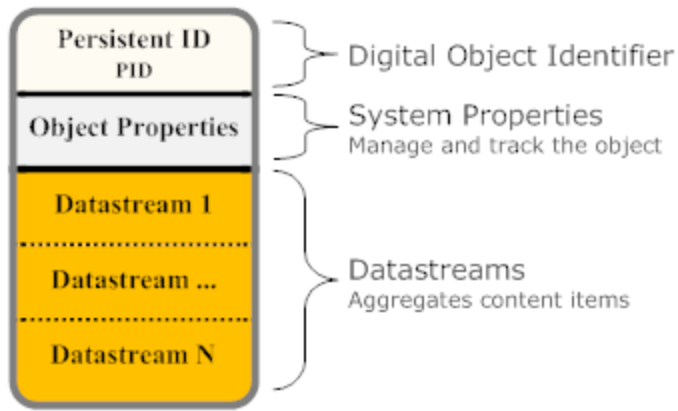
The Fedora Digital Object

Fedora defines a generic digital object model that can be used to persist and deliver the essential characteristics for many kinds of digital content including documents, images, electronic books, multi-media learning objects, datasets, metadata and many others. This digital object model is a fundamental building block of the Content Model Architecture and all other Fedora-provided functionality.

The Fedora Digital Object Model

Fedora uses a "compound digital object" design which aggregates one or more content items into the same digital object. Content items can be of any format and can either be stored locally in the repository, or stored externally and just referenced by the digital object. The Fedora digital object model is simple and flexible so that many different kinds of digital objects can be created, yet the generic nature of the Fedora digital object allows all objects to be managed in a consistent manner in a Fedora repository.

A good discussion of the Fedora digital object model (for Fedora 2 and prior versions) exists in a recent paper ([draft](#)) published in the [International Journal of Digital Libraries](#). While some details of this paper have been made obsolete by the CMA (e.g. Disseminators), the core principles of the model are still part of the CMA. The Fedora digital object model is defined in XML schema language (see [The Fedora Object XML - FOXML](#)). For more information, also see the [Introduction to FOXML](#) in the Fedora System Documentation.



The basic components of a Fedora digital object are:

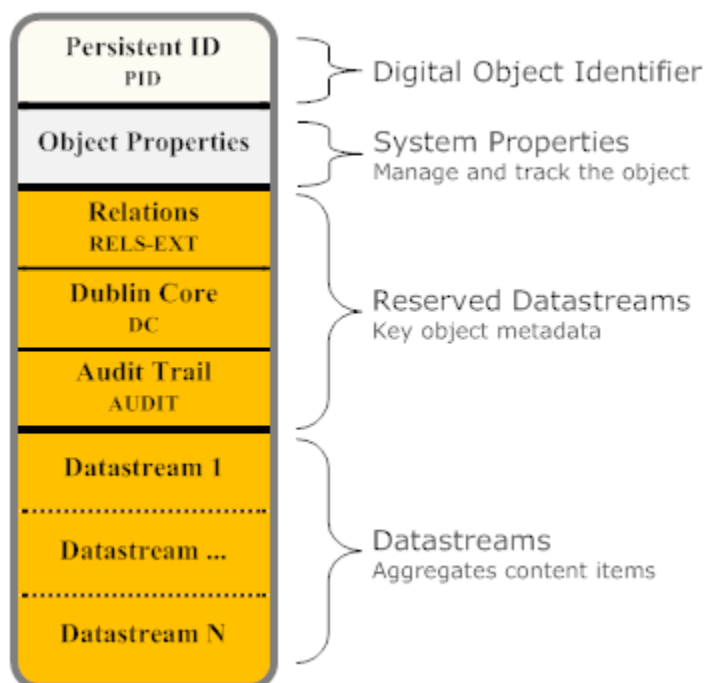
- **PID:** A persistent, unique identifier for the object.
- **Object Properties:** A set of system-defined descriptive properties that are necessary to manage and track the object in the repository.
- **Datastream(s):** The element in a Fedora digital object that represents a content item.

Datastreams

A Datastream is the element of a Fedora digital object that represents a content item. A Fedora digital object can have one or more Datastreams. Each Datastream records useful attributes about the content it represents such as the MIME-type (for Web compatibility) and, optionally, the URI identifying the content's format (from a format registry). The content represented by a Datastream is treated as an opaque bit stream; it is up to the user to determine how to interpret the content (i.e. data or metadata). The content can either be stored internally in the Fedora repository, or stored remotely (in which case Fedora holds a pointer to the content in the form of a URL). The Fedora digital object model also supports versioning of Datastream content (see the [Fedora Versioning Guide](#) for more information).

Each Datastream is given a Datastream Identifier which is unique within the digital object's scope. Fedora reserves four Datastream Identifiers for its use, "DC", "AUDIT", "RELS-EXT" and "RELS-INT". Every Fedora digital object has one "DC" (Dublin Core) Datastream by default which is used to contain metadata about the object (and will be created automatically if one is not provided). Fedora also maintains a special Datastream, "AUDIT", that records an audit trail of all changes made to the object, and can not be edited since only the system controls it. The "RELS-EXT" Datastream is primarily used to provide a consistent place to describe relationships to other digital objects, and the "RELS-INT" datastream is used to describe internal relationships from digital object datastreams. In addition, a Fedora digital object may contain any number of custom Datastreams to represent user-defined content.

Decisions about what to include in a Fedora digital object and how to configure its Datastreams are choices as you develop content for your repository. The examples in this tutorial demonstrate some common models that you may find useful as you develop your application. Different patterns of datastream designed around particular "genre" of digital object (e.g., article, book, dataset, museum image, learning object) are known as "content models" in Fedora.



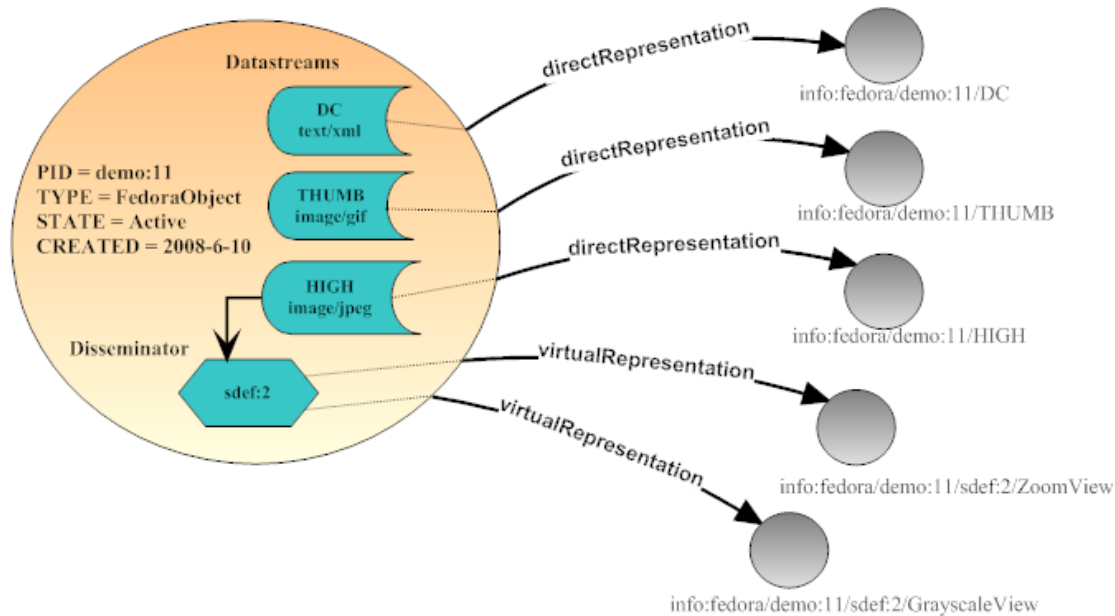
The basic properties that the Fedora object model defines for a Datastream are as follows:

- **Datastream Identifier:** an identifier for the datastream that is unique within the digital object (but not necessarily globally unique)
- **State:** the Datastream's state: Active, Inactive, or Deleted
- **Created Date:** the date/time that the Datastream was created (assigned by the repository service)
- **Modified Date:** the date/time that the Datastream was modified (assigned by the repository service)
- **Versionable:** an indicator (true/false) as to whether the repository service should version the Datastream (by default the repository versions all Datastreams)
- **Label:** a descriptive label for the Datastream
- **MIME Type:** the MIME type of the Datastream (required)
- **Format Identifier:** an optional format identifier for the Datastream such as emerging schemes like PRONOM and the Global Digital Format Registry (GDRF)
- **Alternate Identifiers:** one or more alternate identifiers for the Datastream (such identifiers could be local identifiers or global identifiers such as Handles or DOI)
- **Checksum:** an integrity stamp for the Datastream content which can be calculated using one of many standard algorithms (MD5, SHA-1, etc.)
- **Bytestream Content:** the content (as a stream resource) represented or encapsulated by the Datastream (such as a document, digital image, video, metadata record)
- **Control Group:** the approach used by the Datastream to represent or encapsulate the content as one of four types or control groups:
 - **Internal XML Content** - the content is stored as XML in-line within the digital object XML file
 - **Managed Content** - the content is stored in the repository and the digital object XML maintains an internal identifier that can be used to retrieve the content from storage
 - **Externally Referenced Content** - the content is stored outside the repository and the digital object XML maintains a URL that can be dereferenced by the repository to retrieve the content from a remote location. While the datastream content is stored outside of the Fedora repository, at runtime, when an access request for this type of datastream is made, the Fedora repository will use this URL to get the content from its remote location, and the Fedora repository will mediate access to the content. This means that behind the scenes, Fedora will grab the content and stream in out the the client requesting the content as if it were served up directly by Fedora. This is a good way to create digital objects that point to distributed content, but still have the repository in charge of serving it up.
 - **Redirect Referenced Content** - the content is stored outside the repository and the digital object XML maintains a URL that is used to redirect the client when an access request is made. The content is not streamed through the repository. This is beneficial when you want a digital object to have a Datastream that is stored and served by some external service, and you want the repository to get out of the way when it comes time to serve the content up. A good example is when you want a Datastream to be content that is stored and served by a streaming media server. In such a case, you would want to pass control to the media server to actually stream the content to a client (e.g., video streaming), rather than have Fedora in the middle re-streaming the content out.

Digital Object Model - Access Perspective

Below is an alternative view of a Fedora digital object that shows the object from an access perspective. The digital object contains Datastreams and a set of object properties (simplified for depiction) as described above. A set of access points are defined for the object using the methods described below. Each access point is capable of disseminating a "representation" of the digital object. A representation may be considered a defined expression of part or all of the essential characteristics of the content. In many cases, direct dissemination of a bit stream is the only required access method; in most repository products this is the only supported access method. However, Fedora also supports disseminating virtual representations based on the choices of content modelers and presenters using a full range of information and processing resources. The diagram shows all the access points defined for our example object.

For the access perspective, it would be best if the internal structure of digital object is ignored and treated as being encapsulated by its access points. Each access point is identified by a URI that conforms to the [Fedora "info" URI scheme](#) . These URIs can be easily converted to the URL syntax for the Fedora REST-based access service (API-A-LITE). It should be noted that Fedora provides a several protocol-based APIs to access digital objects. These protocols can be used both to access the representation and to obtain associated metadata at the same access point.



By default, Fedora creates one access point for each Datastream to use for direct dissemination of its content. The diagram shows how these access points map to the Datastreams. The example object aggregates three Datastreams: a Dublin Core metadata record, a thumbnail image, and a high resolution image. As shown, each Datastream is accessed from a separate URI.

Custom access points are created using the Content Model Architecture by defining control objects as described below. Behind the scenes, custom access points connect to services that are called on by the repository to produce representations of the object. Custom access points are capable of producing both virtual and direct representations (though they are likely to provide slower performance). Content in the Datastreams may be used as input as well as caller-provided parameters. A "virtual representation" is produced at runtime using any resource the service can access in conjunction with content generated in its code. In this example, there is one service that contains two operations, one for producing zoomable images and one for producing grayscale images. These operations both require a jpeg image as input, therefore the Datastream labeled "HIGH" is used by this service. Fedora will generate one access point for each operation defined by the service. The control objects contains enough information so that a Fedora repository can automatically mediate all interactions with the associated service. The Fedora repository uses this information to make appropriate service calls at run time to produce the virtual representation. From a client perspective this is transparent; the client just requests a dissemination from the desired access point.

Four Types of Fedora Digital Objects

Although every Fedora digital object conforms to the Fedora object model, as described above, there are four distinct types of Fedora digital objects that can be stored in a Fedora repository. The distinction between these four types is fundamental to how the Fedora repository system works. In Fedora, there are objects that store digital content entities, objects that store service descriptions, objects used to deploy services, and objects used to organize other objects.

Data Object

In Fedora, a Data object is the type of object used to represent a digital content entity. Data objects are what we normally think of when we imagine a repository storing digital collections. Data objects can represent such varied entities as images, books, electronic texts, learning objects, publications, datasets, and many other entities. One or more Datastreams are used to represent the parts of the digital content. A Datastream is an XML element that describes the raw content (a bitstream or external content). In the CMA, Disseminators, a metadata construct used to represent services, are eliminated though their functionality is still provided in other ways.

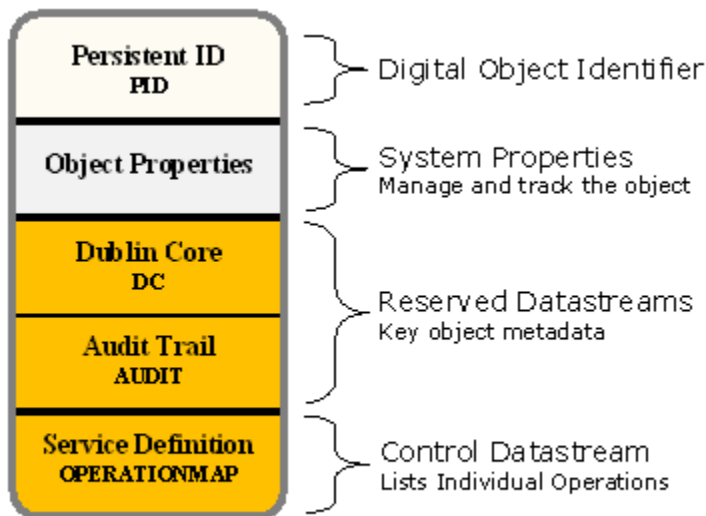
The Data object, indeed all Fedora digital objects, now consists of the FOXML digital object encapsulation (`foxml:digitalObject`) and two fundamental XML elements: Object Properties (`foxml:objectProperties`) and Datastreams (`foxml:datastream`). The Data object is the simplest, most common of all the specialized object types and is identical to the digital object described in the Fedora Digital Object Model section above.

Data objects can now be freely shared between Fedora repositories. If a federated identifier-resolver system, such as the Handle System™, or any authoritative name registry system is used, the Data object will have the same identifier for each copy of itself in each participating repository. Sharing Data objects while keeping the same identifier in each copy greatly simplifies replication, and enables many business processes and services that are needed for large scale repository installations integrated within the Fedora Framework. Data objects can still be shared between repositories by including both the original identifier and alternate identifiers as part of the object's metadata.

Service Definition Object

In Fedora, a *Service Definition object* or `sDef` is a special type of *control object* used to store a model of a Service. A Service contains an integrated set of *Operations* that a Data object supports. In object-oriented programming terms, the `sDef` defines an "interface" which lists the operations that are supported but does not define exactly how each operation is performed. This is also similar to approaches used in Web

(REST) programming and in SOAP Web services. In order to execute an operation you need to identify the Data object, the SDef, and the name of the Operation. Some Operations use content from Datastreams (supplied by the Data object) and, possibly, additional parameters supplied by the client program or browser requesting the execution.



Conceptually an Operation is called using the following form (the specifics vary with the actual Fedora interface being used but all will contain some form of this information):

```
Repository : Get : Data object PID : SDef PID : Operation Name : Optional Parameters
```

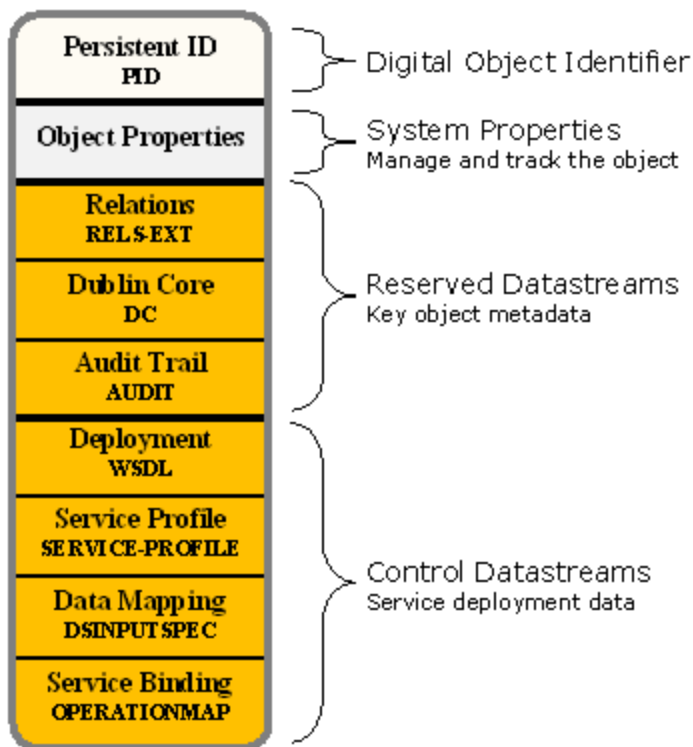
A SDef is a building block in the CMA that enables adding customized functionality for Data objects. Using a SDef is a way of saying "this Data object supports these operations." Essentially, a SDef defines a "behavior contract" to which one or more Data objects may "subscribe." In repositories, we usually create a large number of similar Data objects and want them all to have the same functionality. To make this approach flexible and easier to use, the CMA uses the Content Model (CModel) object (described below) to contain the model for similar Data objects. Instead of associating the SDef directly with each Data object, the relation *hasService* is asserted to the CModel object. By following the relation between the Data object to the CModel object, and then from the CModel object to the SDef object, we can determine what Operations the Data object can perform. Also note that a Data object (through its CModel object) may support more than one Service (by having multiple SDef relations).

SDef objects can now be freely shared between Fedora repositories. If a federated identifier-resolver system, such as the Handle System™, or any authoritative name registry system is used the SDef object will have the same identifier for each copy of itself in each participating repository. Sharing SDef objects while keeping the same identifier in each copy greatly simplifies replication, and enables many business processes and services that are needed for large scale repository installations integrated within the Fedora Framework. SDef objects can still be shared between repositories by including both the original identifier and alternate identifiers as part of the object's metadata. The best results will be gained by sharing the Data object, SDef objects, and Content Model object as a group maintaining the same original identifiers. By using the CMA in this fashion, you transfer a significant unit of the data and metadata that documents the expression pattern for your intellectual work. While this is, by itself, not everything needed, it is a big step forward for creating a durable content repository.

It is worth noting that Service Definition objects conform to the basic Fedora object model. Also, they are stored in a Fedora repository just like other Fedora objects. As such, a collection of SDef objects in a repository constitutes a "registry" of Service Definitions.

Service Deployment Object

The Service Deployment object is a special type of control object that describes how a specific repository will deliver the Service Operations described in a SDef for a class of Data objects described in a CModel. The SDep is executable code but instead it contains information that tells the Fedora repository how and where to execute the function that the SDep represents. In the CMA, the SDep acts as a deployment object only for the specific repository in which it is ingested; each repository is free to provide functionality in a different way. For example, one Fedora repository may choose to use a Servlet and another may use a SOAP Web service to perform the same function. As another example, individual repository implementations may need to provide the functionality at different end points. Or perhaps, a specific installation may use a dynamic end point resolution mechanism to permit failover to different service providers.



Since the SDep operates only within the scope of an individual repository, the operators of that repository are free to make changes to the SDep or the functionality it represents at any time (except for temporarily making the object's services unavailable while the change is being made). This approach permits the system operators to control access to services called by the Fedora repository to institute security or policies as their organization determines. It enables Fedora-called services to be managed using the same principles and tools for the deployment of any distributed system. It also enables the system operators to reconfigure their systems quickly without having to change any part of their content except the SDep object.

The SDep stores concrete service binding metadata. A SDep uses a *isDeploymentOf* relation to a SDef as its way of saying "I am able to perform the service methods described by that SDef." A SDep object is related to a SDef in the sense that it defines a particular concrete implementation of the abstract operations defined in a SDef object. The SDef also uses a *isContractorOf* relation to a CModel as a way of saying "Use me to do the service operations for any Data objects conforming to that CModel."

A SDep object stores several forms of metadata that describe the runtime bindings for invoking service methods. The most significant of these metadata formats is service binding information encoded in the Web Services Description Language (WSDL). The Fedora repository system uses the WSDL at runtime to dispatch service method requests in fulfilling client requests for "virtual representations" of a Data object (i.e., via its Operations). This enables Fedora to talk to a variety of different services in a predictable and standard manner. A SDep also contains metadata that defines a "data contract" between the service and a class of Fedora Data objects as defined in the CModel. For the initial deployment of the CMA a simple data contract mechanism was chosen. Since the Datastream IDs are specified in the CModel and the SDep is now a deployment control object only for a specific repository, the SDep is able to uniformly bind directly to these IDs. In the future a more abstract binding mechanism may be used but this approach is simple and clear, though it may require the creation of a small number of additional SDep objects.

A major aspect of the CMA redesign is that there is no requirement that conformance to a Content Model or that referential integrity between objects be checked at ingest time. This may result in a run-time error if the repository cannot find referenced objects, interpret the Content Model or if there are any conformance problems.

It is worth noting that SDep objects conform to the basic Fedora object model. Also, they are stored in a Fedora repository just like other Fedora objects. As such, a collection of SDep objects in a repository constitutes a "registry" of service deployments that can be used with Fedora objects. In the CMA, SDep objects are not freely sharable across repositories. They represent how a specific repository implements a service. However, SDep objects can be shared if the operator of the system modifies them for local deployment. Because of this, SDep objects should not be automatically replicated between repositories without considering the affect.

Content Model Object

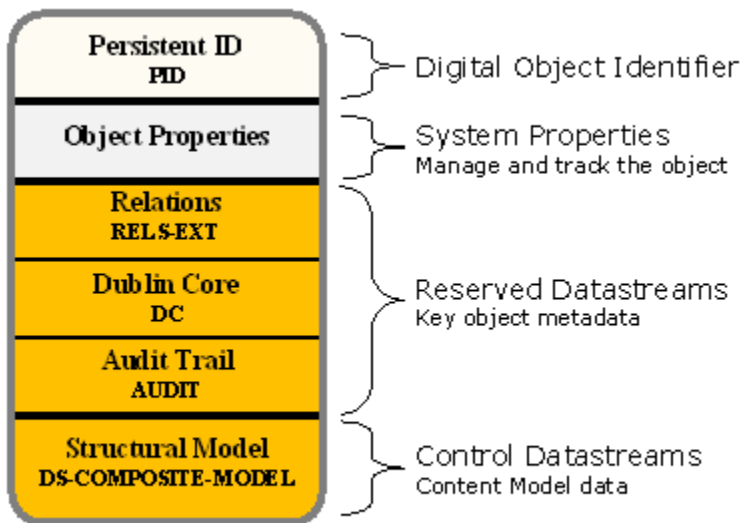
The Content Model object or CModel is a new specialized control object introduced as part of the CMA. It acts as a container for the Content Model document which is a formal model that characterizes a class of digital objects. It can also provide a model of the relationships which are permitted, excluded, or required between groups of digital objects. All digital objects in Fedora including Data, SDef, SDep, and CModel objects are organized into classes by the CModel object. In this section, we will primarily discuss the relationship between the Data and CModel objects.

To create a class of Data objects, create a CModel object. Each Data object belonging to the class asserts the relation *hasModel* using the identifier of the CModel as the object of the assertion. The current CModel object contains a structural model of the Data object. Over time there

will be additional elements to the Content Model document but this initial implementation is sufficient to describe the Datastreams which are required to be present in each Data object in the class. The other key relation is to the SDef objects. You can assert zero or more *hasService* relations in the CModel to SDef objects.

A Data object may assert a *hasModel* relationship to multiple CModel objects. Such a Data object should conform to all of its Content Models, containing an aggregation of all the Datastreams defined by the Content Models. If two or more Content Models define Datastreams which have the same name but different characteristics, no well-formed Data object can be constructed and likely the repository will be unable to deliver its content or services. Fedora automatically assumes that all objects conform to a system-defined "Basic Content Model." There is no need to assert a relation to this content model explicitly but, if the Data object asserts other relations, it is a good practice to make the assertion to the Basic Content Model explicit. Regardless, the repository will behave the same whether the relation is asserted or not. Along with the Basic Content Model, the repository defines a "Basic Service Definition" which supplies Operations common to all objects. One such service provides direct access to the Datastreams.

Because of the Basic Content Model and the Basic Service Definition, nothing needs to be added to a Data object if the user only wants to store and disseminate Datastreams by name. However, without an explicit Content Model you cannot validate whether the Data object is correctly formed. In the CMA, if the repository cannot find and interpret all the control objects related to a Data object, or cannot interpret the Content Model, it will issue a runtime error when the Data object is accessed. Note that the repository will always be able to perform basic Datastream operations because they are a part of the Basic Content Model and Basic Service Definition. Other than conformance to the rules for a properly formed digital object, there is no warning or error issued on ingest or modification of an object in the CMA.



CModel objects can now be freely shared between Fedora repositories. If a federated identifier-resolver system, such as the Handle System™, or any authoritative name registry system is used the CModel object will have the same identifier for each copy of itself in each participating repository. Sharing CModel objects while keeping the same identifier in each copy greatly simplifies replication, and enables many business processes and services that are needed for large scale repository installations integrated within the Fedora Framework. CModel objects can still be shared between repositories by including both the original identifier and alternate identifiers as part of the object's metadata. The best results will be gained by sharing the Data object, SDef objects, and CModel objects as a group maintaining the same original identifiers. By using the CMA in this fashion, you transfer a significant unit of the data and metadata that documents the expression pattern for your intellectual work. While this is, by itself, not everything needed, it is a big step forward for creating a durable content repository. Over time, Content Model languages can be developed that permit describing an ever larger portion of the essential characteristics of the content and its behaviors.

It is worth noting that Content Model objects conform to the basic Fedora object model. Also, they are stored in a Fedora repository just like other Fedora objects. As such, a collection of Content Model objects in a repository constitutes a "registry" of Content Models.

Fedora Identifiers

PIDs

A PID is a unique, persistent identifier for a Fedora digital object. PIDs may be user-defined or automatically assigned by a repository. In this section we describe the syntactic and normalization considerations for PIDs.

Syntax

PIDs are case-sensitive and consist of a namespace prefix and a simple string identifier. The syntax is described below using [augmented BNF](#):

```
object-pid    = namespace-id ":" object-id
namespace-id  = 1*( ALPHA / DIGIT / "-" / "." )
object-id     = 1*( ALPHA / DIGIT / "-" / "." / "~" / "_" / escaped-octet )
escaped-octet = "%" HEXDIG HEXDIG
```

The maximum length of a PID is 64 characters.

For convenience, we provide the following single regular expression, which can be used to validate a normalized PID string:

```
^[A-Za-z0-9]|\.\.+:(((A-Za-z0-9)|-\.\.|\~|_|(%[0-9A-F]{2}))+)$
```

Examples

- demo:1
- demo:A-B.C_D%3AE
- demo:MyFedoraDigitalObject

Normalization

HEXDIG characters may occur in lowercase, but should be capitalized for normalization purposes. The separator character may occur as "%3A" or "%3a", but should be changed to a colon ":" for normalization purposes.

Datastream IDs

Datastreams IDs may consist only of [XML NCName characters](#) and must not exceed 64 characters in length.

URIs for Objects

It is often useful to have Uniform Resource Identifiers ("URIs") that refer to Fedora Objects. For instance, semantic web technologies require the use of a URI to identify a subject. Other benefits of exposing and using URIs are described in [Section 2 of the W3C's Architecture of the World Wide Web](#).

Every Fedora object has an implicit URI associated with it. These identifiers exist within the "fedora" namespace of the "info" URI scheme. We chose this URI scheme due to its resolution protocol independence and syntactic freedom.

Syntax

The URI for a Fedora object is constructed simply by appending the PID to the string "info:fedora/".

Examples

- info:fedora/demo:1
- info:fedora/demo:A-B.C_D%3AE
- info:fedora/demo:MyFedoraDigitalObject

Normalization

To normalize an object URI, normalize the PID part as described above.

URIs for Disseminations

Every dissemination of an object also has an implicit URI associated with it. This is useful when describing or referring to the representations provided by a digital object.

Syntax

Dissemination URIs take one of two forms. In the case of a method call the URI indicates the service definition and the method (along with any parameters). In the case of a datastream dissemination, the URI indicates the Datastream id.

```
dissemination-uri = "info:fedora/" pid "/" ( method-call / datastream-id )
method-call       = sDef-pid "/" method-name [ "?" param *( "&" param ) ]
param             = paramName "=" paramValue
```

Note: Although datastream-ids and method-names may consist of XML NCName characters. NCName characters that are not URI-safe must be escaped using one to four escaped UTF-8 octets per character, each of the form "%" HEXDIG HEXDIG.

Examples

- info:fedora/demo:1/demo:MySDef/method
- info:fedora/demo:1/demo:MySDef/method?param1=value1
- info:fedora/demo:1/title.jpg
- info:fedora/demo:1/DC

Normalization

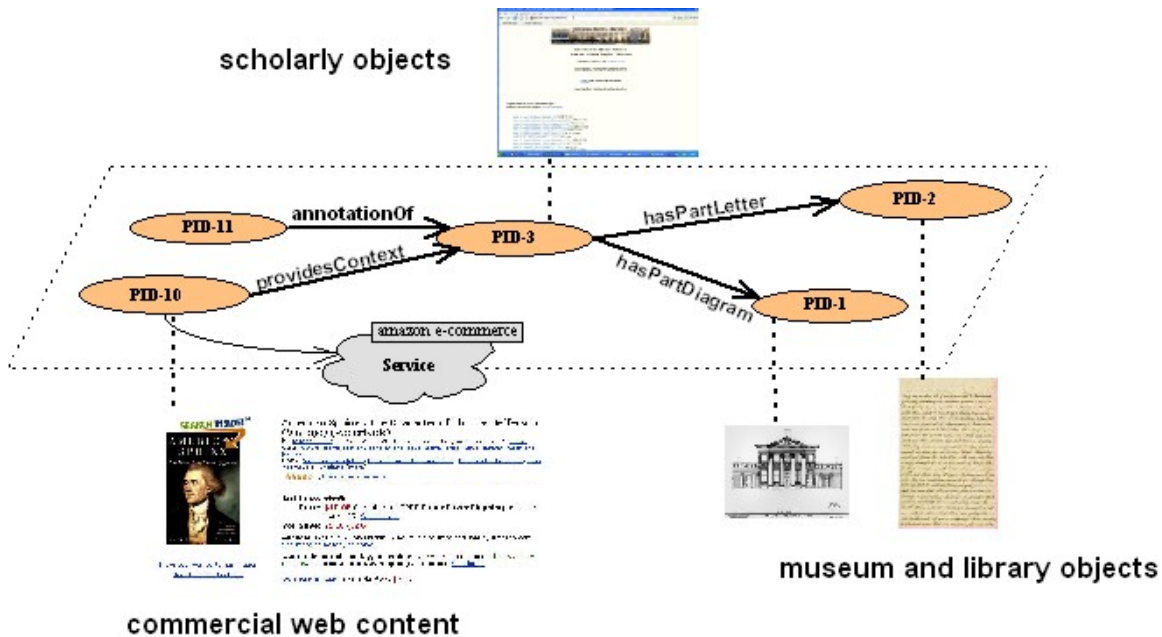
To normalize a dissemination URI:

1. Normalize the PID portion(s) of the URI.
2. Un-escape any URI-escaped characters that do not need escaping according to the definition of the "info" scheme.
3. Make all remaining escaped octets use UPPERCASE (%ff becomes %FF).
4. Parameters should be alphabetized in order by name, then by value. The order should be according to occurrence in UTF-8.

Digital Object Relationships

What are Fedora Digital Object Relationships?

Fedora digital objects can be related to other Fedora objects in many ways. For example there may be a Fedora object that represents a collection and other objects that are members of that collection. Also, it may be the case that one object is considered a part of another object, a derivation of another object, a description of another object, or even equivalent to another object. For example, consider a network of digital objects pertaining to Thomas Jefferson, in which scholarly works are stored as digital objects, which are related to other digital objects representing primary source materials in libraries or museums. The composite scholarly objects can be considered a graph of related digital objects. Other types of objects can also be related to the scholarly object over time, for instance annotations about the scholarly object can be created by others and related to the original object. Also, digital objects can be created to act as "surrogates" or "proxies" for dynamically produced web content such as an Amazon page for a book relevant to the scholarly object. Such a network of digital objects can be created using Fedora, which in the abstract, would look like this:



Digital object relationship metadata is a way of asserting these various kinds of relationships for Fedora objects. A default set of common relationships is defined in the [Fedora relationship ontology](#) (actually, a simple RDF schema) which defines a set of common generic relationships useful in creating digital object networks. These relationships can be refined or extended. Also, communities can define their own ontologies to encode relationships among Fedora digital objects. Relationships are asserted from the perspective of one object to another object as in the following general pattern:

```
<subjectFedoraObject> <relationshipProperty> <targetFedoraObject>
```

The first Fedora object is considered the "subject" of the relationship assertion. The relationship, itself, is considered a property of the subject. The

target Fedora object is the related object. Thus, a valid relationship assertion as an English-language sentence might be:

```
<MyCatVideo> <is a member of the collection> <GreatCatVideos>
```

Using RELS-INT, relationships can also be asserted from the datastream of one object to another object or datastream.

```
<MyCatPicture/Thumbnail> <is thumbnail of> <MyCatPicture/FullSizeImage>
```

Why are Fedora Digital Object Relationships Important?

The creation of Fedora digital object relationship metadata is the basis for enabling advanced access and management functionality driven from metadata that is managed within the repository. Examples of the uses of relationship metadata include:

- Organize objects into collections to support management, OAI harvesting, and user search/browse
- Define bibliographic relationships among objects such as those defined in [Functional Requirements for Bibliographic Records](#)
- Define semantic relationships among resources to record how objects relate to some external taxonomy or set of standards
- Model a network overlay where resources are linked together based on contextual information (for example citation links or collaborative annotations)
- Encode natural hierarchies of objects
- Make cross-collection linkages among objects (for example show that a particular document in one collection can also be considered part another collection)

Where is Digital Object Relationship Metadata Stored?

Object-to-Object relationships are stored as metadata in digital objects within special Datastreams. These Datastreams are known by the reserved Datastream identifiers of "RELS-EXT" (which stands for "Relationships-External") and "RELS-INT" (which stands for "Relationships-Internal"). Each digital object can have one RELS-EXT datastream which is used exclusively for asserting digital object relationships and one RELS-INT datastream which is used exclusively for asserting relationships from datastreams of the digital object.

RELS-EXT and RELS-INT Datastreams can be provided as part of a Fedora ingest file. Alternatively, they can be added to an existing digital object via component operations of the Fedora management service interface (i.e., `addDatastream`, `addRelationship`). Refer to the [FOXML reference example](#) to see an example of the RELS-EXT Datastream in context. Modifications to the RELS-EXT and RELS-INT Datastreams are made via the Fedora management interface (i.e., `modifyDatastream`).

The RELS-EXT and RELS-INT Datastreams should be encoded as either Inline XML Datastreams, meaning that the relationships metadata is expressed directly as XML within the digital object XML file, or as Managed Content datastreams. The datastream control group used when Fedora creates new relationships datastreams (eg as part of an `addRelationship` API method) is specified in the `DOManager` section of `fedora.fcfig` using the property `defaultRELSControlGroup`.



Ensure DC, RELS-EXT and RELS-INT are versionable if using Managed Content

Due to an outstanding bug [FCREPO-849](#), if you use Managed Content for DC, RELS-EXT or RELS-INT then please make sure these datastreams are versionable (the default setting for versionable is "true", so if you haven't specified this datastream property then you are safe).

How is Digital Object Relationship Metadata Encoded?

Fedora object-to-object metadata is encoded in XML using the [Resource Description Framework \(RDF\)](#). The relationship metadata must follow a prescribed RDF/XML authoring style where the subject is encoded using `<rdf:Description>`, the relationship is a property of the subject, and the target object is bound to the relationship property using the `rdf:resource` attribute. The subject of a relationship assertion must be a URI that identifies either a Fedora digital object for RELS-EXT, or a Fedora digital object datastream for RELS-INT. These URIs are based on Fedora object PIDs and conform to the syntax described for the [Fedora "info" URI scheme](#).

The syntax for asserting RELS-EXT relationships in RDF is as follows:

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:fedora="info:fedora/fedora-system:def/relations-external#"
  xmlns:myns="http://www.nsd1.org/ontologies/relationships#">
  <rdf:Description rdf:about="info:fedora/demo:99">
  <fedora:isMemberOfCollection rdf:resource="info:fedora/demo:c1"/>
  <myns:isPartOf rdf:resource="info:fedora/mystuff:100"/>
  <myns:owner>Jane Doe</myns:owner>
  </rdf:Description>
</rdf:RDF>

```

The above RDF fragment indicates that the Fedora digital object identified as "info:fedora/demo:99" is the subject that is being described (as specified by the `rdf:about` attribute). This object has two relationships to other digital objects. It has an "isMemberOfCollection" relationship to the object identified as "info:fedora/demo:c1" which is a Fedora object that represents a collection. This collection object is considered the target of the relationship assertion. The second relationship assertion that is just like the first one, except that it asserts that the subject is a part of another Fedora digital object, "info:fedora/mystuff:100". The third relationship asserts that the digital object has owner "Jane Doe". Note that the object of this relationship is a text literal, not a URI.

The syntax for asserting RELS-INT relationships in RDF is as follows:

```

<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:fedora="info:fedora/fedora-system:def/relations-external#"
  xmlns:myns="http://www.nsd1.org/ontologies/relationships#">
  <rdf:Description rdf:about="info:fedora/demo:99/Thumbnail">
  <myns:isThumbnailOf rdf:resource="info:fedora/demo:99/FullSizeImage"/>
  </rdf:Description>
  <rdf:Description rdf:about="info:fedora/demo:99/FullSizeImage">
  <myns:hasImageSize>1600 x 900</myns:hasImageSize>
  </rdf:Description>
</rdf:RDF>

```

The above RDF fragment indicates that the Fedora digital object datastream identified as "info:fedora/demo:99/Thumbnail" is the subject that is being described (as specified by the `rdf:about` attribute) in the first `rdf:Description` element. This datastream has a relationship to another datastream, in this case within the same digital object. It has an "isThumbnailOf" relationship to the datastream identified as "info:fedora/demo:99/FullSizeImage", asserting that it is a thumbnail of the full-sized image.

The second `rdf:Description` element has the datastream "info:fedora/demo:99/FullSizeImage" as the subject, and it asserts that the image size is 1600 x 900 pixels. Note that the object of this relationship is a text literal, not a URI.

Unlike RELS-EXT, the RDF for RELS-INT can have multiple `rdf:Description` elements, with each having an `rdf:about` attribute identifying a datastream within this Fedora digital object, and each `rdf:Description` element can contain one or more relationships with specified datastream as the subject.

To ensure the integrity of relationship metadata so that it can be properly indexed by Fedora, the Fedora repository service validates all RELS-EXT and RELS-INT Datstreams and enforces the following constraints on the RDF.

RELS-EXT Validation

1. The subject must be encoded as an `<rdf:Description>` element, with an "rdf:about" attribute containing the URI of the digital object in which the RELS-EXT Datastream resides. Thus, relationships are asserted about this object only. Relationship directionality is from *this* object to other objects.
2. The relationship assertions must be RDF properties associated with the `<rdf:Description>`. Relationship assertions can be properties defined in the default [Fedora relationship ontology](#), or properties from other namespaces.
3. Prior to 2.1, the objects of relationships were restricted to other Fedora digital object URIs. This has since been relaxed so that a relationship property may reference any URI or literal, with the following exception: a relationship may not be self-referential, `rdf:resource` attribute must not point to the URI of the digital object that is the subject of the relationship.
4. There must be only one `<rdf:Description>` in the RELS-EXT datastream. One description can have as many relationship property assertions as necessary.
5. There must be no nesting of assertions. Specifically, there cannot be an `<rdf:Description>` within an `<rdf:Description>`. In terms of XML "depth," the RDF root is considered at the depth of zero. There must be one `<rdf:Description>` element that must exist at the depth of one. The relationship assertions are RDF properties of the `<rdf:Description>` that exist at a depth of two.

6. Assertions of properties from certain namespaces for forbidden in RELS-EXT. There must NOT be any assertion of properties from the Dublin Core namespace or from the FOXML namespace. This is because these assertions exist elsewhere in Fedora objects and may conflict if asserted in two places. The RELS-EXT Datastream is intended to be dedicated to solely object-to-object relationships and not used to make general descriptive assertions about objects.

RELS-INT Validation

1. The subject must be encoded as an `<rdf:Description>` element, with an "rdf:about" attribute containing the URI of a digital object datastream in which the RELS-INT Datastream resides. Thus, relationships are asserted about datastreams in this object only. Relationship directionality is from *this* object's datastreams to other objects and datastreams. (Note that the existence of the datastream is not checked, the URI must be syntactically valid for a datastream in this digital object, but the datastream does not actually have to exist.)
2. The relationship assertions must be RDF properties associated with the `<rdf:Description>`. Relationship assertions can be properties defined in the default [Fedora relationship ontology](#), or properties from other namespaces.
3. A relationship property may reference any URI or literal.
4. There may be more than one `<rdf:Description>` elements. One `<rdf:Description>` should be included for each datastream to be described, and each must have an "rdf:about" attribute identifying the datastream being described. One description can have as many relationship property assertions as necessary.
5. There must be no nesting of assertions. Specifically, there cannot be an `<rdf:Description>` within an `<rdf:Description>`. In terms of XML "depth," the RDF root is considered at the depth of zero. There must be one `<rdf:Description>` element that must exist at the depth of one. The relationship assertions are RDF properties of the `<rdf:Description>` that exist at a depth of two.
6. Assertions of properties from certain namespaces is forbidden in RELS-INT. There must NOT be any assertion of properties from the Fedora object properties namespaces (model and view). This is because these assertions exist elsewhere in Fedora objects and may conflict if asserted in two places.

Resource Index - RDF-based Indexing and Searching for Digital Objects

Yes! The Fedora repository service automatically indexes the RELS-EXT and RELS-INT Datastreams for all objects as part of the [RDF-based Resource Index](#).

This provides a unified "graph" of all the objects in the repository and their relationships to each other. The Resource Index graph can be queried using SPARQL or iTQL which are SQL-like query languages for RDF. The Fedora repository service exposes a web service interface to search the Resource Index. Please refer to the Resource Index documentation for details.

Content Model Architecture

Audience

If you are only interested in using a Fedora Repository to store your content (and associated metadata) and access the content exactly in the format you stored it, you do not need to read this section. Fedora will use the Content Model Architecture behind the scenes and you do not have to do anything different from earlier versions of Fedora. However, if you want to use Fedora's full capability the CMA is the gateway. We will try to lead you through steps to understanding the CMA and what it can potentially do for you.

Introduction

A major goal of the Fedora architecture has been to provide a simple, flexible and evolvable approach to deliver the "essential characteristics" for enduring digital content. Whenever we work with digital content, it is with an established set of expectations for how an intellectual work may be expressed. With experience we develop "patterns of expression" that are the best compromise we can craft between the capabilities of our digital tools and the intellectual works we create in digital form. We store our digital content with the expectation that all the important characteristics of our intellectual works will be intact each and every time we return to access them, whether it has been a few minutes or many years.

We also want to communicate our intellectual works effectively to others. To do this, we have an expectation that the important aspects of our digital works are delivered accurately to users accessing them. Often we teach the same patterns of expression used to create our works to aid our users in comprehending them. And the same patterns may be used once again to enable collaboration for creating new or derived works.

Librarians, archivists, records managers, media producers and the myriad other author and publishers of intellectual works have long used content patterns calling them, for example: books, journals, articles, collections, installations. The term "content model" originated with the publishing community to describe the physical structure of the intellectual work. Users and collectors of intellectual works often add value by organizing works, annotating them, and producing ways to find them or information within them.

With the development of digital technology, publishers, users and collectors have applied these same patterns to this new media with some initial success. But like many advancements, digital technology enables ways to create and use content in ways that cannot be achieved with physical media. While it is beyond the scope of this document to discuss the many emerging aspects of digital content technology and its use, there are two key subjects that help in understanding the requirements of the architecture presented here.

First, reusable patterns such as content models can reduce the effort to create or capture, ingest, store, manage, preserve, transform, and access digital content. For example, content models can be used for content classification to facilitate discovery. Other uses include content validation usually at ingest or modification. Content models can be, for example, used as a template when content is created to generate user interfaces, drive workflows, describe content components, or to manage policy enforcement.

Second, digital content is not defined by its format or technology, and may also incorporate functions as a part of its nature. For example, when we access a digital picture we experience its resolution and color fidelity; the technology that delivers that experience is irrelevant. A spreadsheet or a video game is hardly the same thing if there is no software to make it function. Those features which must be present to provide an authentic experience of the digital content are called the "essential characteristics" and they can be captured in a content model to ensure durability of the experience as format and technology changes over time. Those same characteristics also facilitate sharing by describing the nature of the content and the ways it can be used.

Similar needs have been noted in the software engineering community for the development of complex computer systems. Often, organizational information outlasts the technology used to create and access it. Corporate mergers and breakups raise havoc with the integration of company information technology infrastructures. The same concepts that have been developed to satisfy agile IT infrastructures can help provide solutions for creating, accessing and preserving content. In this document, we introduce the fundamental concepts of Fedora's Content Model Architecture or CMA. The CMA builds upon the basic building blocks established by previous versions of the Fedora architecture, restructuring them to both simplify use while unlocking their potential.

We use the term "content model" to mean both:

1. Content structure as used by publishers and other traditional content-related professions
2. A computer model describing an information representation and processing architecture

By combining these very different views, CMA has the potential to provide a way to build an interoperable repository for integrated information access in our organizations and to provide durable access to our intellectual works.

As we introduce CMA concepts, we will discuss the rationale behind the design decisions. This is only the first generation of the CMA and, like the rest of Fedora, we expect it to evolve. An understanding of design decisions behind this "first-generation" CMA is a key element for community participation in future generations of CMA development. Most important is an understanding of three significant and interrelated developments in software engineering: (1) object-oriented programming, (2) design patterns, and (3) model-driven architectures. It is beyond the scope of this document to discuss any of these developments in detail but we will make reference in this document to aspects of them which inform the design of the CMA.

Content Model Architecture Overview

The Content Model Architecture (CMA) describes an integrated structure for persisting and delivering the essential characteristics of digital objects in Fedora. In this section we will describe the key elements of the architecture, how they relate, and the manner in which they function. The original motivation for the CMA was to provide a looser binding for Disseminators, an element of the Fedora architecture used to stream a representation to a client. However, the CMA as described in this document has encompassed a far greater role in the Fedora architecture, in many ways forming the over-arching conceptual framework for future development of the Fedora Repository.

The Fedora application community developed a number of clever approaches to add content model-like capabilities to Fedora. The CMA formalizes some of the "best of breed" techniques gained from further research and from our application community. Two primary ways of thinking about content models have emerged and both are supported by the CMA. The first approach is focused on using complex single-object models and is commonly called "compound." The second approach is to use multi-object models and is commonly called "atomistic" or "linked."

Prior implementations of the Fedora Repository utilized a set of specialized digital objects as a functional and persistence framework. All of these objects conform to the same basic object model. Digital objects in CMA are conceptually similar in prior versions of Fedora though some important implementation details have changed. Fedora still implements a compound digital object design consisting of an XML encapsulation (now FOXML 1.1) and a set of bitstreams identified by the "Datastream" XML element. We can also assemble multi-object groups of related digital objects as before using semantic technologies.

In the CMA, the "content model" is defined as a formal model that describes the characteristics of one or more digital objects. A digital object may be said to conform to a content model. In the CMA, the concept of the content model is comprehensive, including all possible characteristics which are needed to enable persistence and delivery of the content. This can include structural, behavioral and semantic information. It can also include a description of the permitted, excluded, and required relationships to other digital objects or identifiable entities.

The content model is expressed in a modeling language and stored in one or more bitstreams like any other kind of content. It may be useful to think of the content model as one or more closely related documents that contain a machine processable model.

There will likely be more than one content modeling language; the CMA does not specify that there be only a single standardized one. One of the key aspects of Fedora is the expectation that over time all things will change and the same will be true of content modeling languages. There are many valid approaches to content modeling and we wish to enable innovation in this area. Guided by these observations, the CMA is designed to be a framework for developing and deploying content model-driven repository infrastructures. However, while it is a Fedora first principle to minimize required elements, there are a small set of features, described below, that the content modeling language must provide in order for Fedora to function.

Since we anticipate that a large number of digital objects will conform to the same content model, they may be treated as a class. While we need to be careful about analogies between "class" in CMA and "class" in object oriented programming languages or in semantic technologies, exploiting the notion of "class" is a major objective of CMA.

We must have a unique, unambiguous method to identify the class. For this purpose, in the CMA we have defined the "Content Model object," a digital object that both represents the notion of the class and can contain the content model. We use the identifier of the content model object (or

CModel) as the class identifier. Following the rules of Fedora identifiers, the identifier of the CModel object can be encoded within a URI. We will describe the rationale for this decision in a later section but this approach provides two immediate benefits: (1) it provides a scheme which works within the Fedora architecture with minimal impact, and (2) it is compatible with the Web architecture, RDF and OWL. We can even build functionality using just the knowledge of the identifier without creating a content model. Having a uniform method for identifying a digital object's class maximizes interoperability.

The CMA does not require that digital objects explicitly conform to its architecture or explicitly declare any of its metadata elements beyond providing well-formed Fedora digital objects - unless you want to use the advanced features provided by the Fedora repository. The CMA uses a "descriptive" approach where the Fedora Repository will issue a run-time error for any operation it cannot perform. In most cases, you should still be able to disseminate bitstreams exactly as they are stored. CMA's dissemination approach is more consistent with the Web architecture, and provides a better balance between durable access to content and future innovations.

The minimum requirement to participate in the CMA is for a digital object to assert a relation to record its class' identity. Digital objects that do not explicitly identify their class are assumed to belong to a system-defined "Default Content Model" which has a repository-defined reserved identifier. In CMA, you should use a digital object (see CModel below) as a way to register a "class" in a repository federation.

The remaining functionality is enabled by creating the content model and storing it within Fedora digital objects like any other content. This permits applications or the repository itself to disseminate the content model, interpret it and use it to provide functionality for the set of objects it describes. Content designers are free to develop content models (or even content modeling languages). Content Models and Content Model objects are designed to be shared. Digital objects having the same content model automatically form an interoperable information community. If the whole Content Model object is shared, keeping the object's identifier the same across multiple repositories, the community can easily extend across multiple organizations.

There are two basic approaches to using content models. First, the content and content models can be disseminated to applications able to interpret them to deliver the essential characteristics of the content. Disseminating the content model to external services can be also be used when creating new digital objects for ingest, validation, transformation and replication.

Alternately, the Fedora Repository can be used as a content mediation layer which interprets the content model to disseminate the content correctly. To accomplish this, the Fedora Repository must have access to code compatible with the Content Model. The compatible code may be added to the Fedora Repository using its plug-in architecture in combination with service mechanisms. While all the tools needed to plug in your own Content Model mediation are not complete in Fedora 3.0, this feature of the CMA will permit serving several generations of digital objects in the same repository reducing the need to update objects whenever there are new releases of Fedora. This will increase the durability of collections and enable longer periods between migrations of Fedora digital objects to new formats.

While the CMA does not force you to use a specific content modeling language, Fedora 3.0 contains a reference implementation that enables the Fedora Repository to operate much as it did in prior versions. The following sections describe CMA in more detail and provide instructions on how to use the reference content modeling language so you can create your own CMA compatible objects immediately. Over time Fedora Commons will support the development of one or more content modeling languages as part of solution bundles that may be used by the community with minimum effort.

Specializing Digital Objects

One of the basic elements of the Fedora architecture is the Fedora digital object. Every digital object stored in a Fedora repository is some variation of the same basic Fedora digital object model. In Fedora, digital objects containing data (Data object) utilize an XML element called the Datastream to describe the raw content (a bitstream or external content). In Fedora 2 and prior versions, digital objects containing data may also have contained Disseminators. The Disseminator is a metadata construct used by Fedora to describe how a client can access content within the digital object or remotely referenced by the digital object. If you only needed to access the raw content, default functionality was provided by the Fedora Repository which did not require that a Disseminator be explicitly added to the digital object. Unfortunately, the older design meant that the Disseminator was repeated in every Data object.

In Fedora 2 and prior versions, three specializations of the Fedora digital object were used: the Data object, the Behavior Definition (*BDef*) and the Behavior Mechanism (*BMech*). The BDef contained a description of the access interface which, in turn, was implemented by the BMech. In combination, the Fedora digital object model provides a uniquely flexible way to persist and access digital content. However, these features were not as simple to use as they could be. In particular, repeating the Disseminator in every Data object made changes to the access interface very difficult since every object had to be changed.

In the CMA, the Fedora digital object remains the model for all digital objects just as it was in prior versions of Fedora. However, for the CMA we define four specialized variations of the Fedora digital object (see Table 1).

Object Type	Code	Description
Data	Data	A container for content
Service Definition	SDef	A container for the service definitions, an element of a content model
Service Deployment	SDep	A container for service deployment bindings
Content Model	CModel	A container for content models

Table 1 - Fundamental Fedora Object Types

Each of these specialized digital objects will be described in much greater detail below. The BDef and BMech have long been recognized as having reserved functions in Fedora and we often called them "control" objects because they contain data used to control object-specific functionality. CMA adds a new control object, the CModel. However, some of the data needed for object-specific functionality was located in the Data object (often just labeled generically as the digital object) in prior Fedora versions. In particular, Disseminators were defined in the Data object. The CMA eliminates the Disseminator and redistributes control data in a more logical and reusable fashion among the "control" objects.

Care should be taken in equating the names of object types in the Fedora 3.0 from the prior (Release 2 and earlier) Fedora architecture because there are some similarities to roles played by these objects in both the old and new architecture. However, the CMA is a major redesign so care should be taken not to automatically equate the roles of the control objects in the old and new implementations. In pre-release reviews with community members, we discovered that there were concerns about the potential of confusion so we have used a new naming scheme for Fedora 3.0.

Also, care should be taken not to equate the digital object as a container and any data (or metadata) it contains. Since the deployment, functional, persistence and information views of the Fedora architecture tend to intermingle more than in most architectures, it is easy to accidentally conflate characteristics of the architectural elements across views. We will try to make the distinctions clear where needed.

Table 2 lists the supported relationships between fundamental object types in the CMA.

Origin	Target	Relation	Purpose
Data	CModel	hasModel	Identifies the class and, optionally, the object containing a model of the essential characteristics of the class
CModel	SDef	hasService	Identifies the object containing a model of the functional characteristics of class members
SDep	SDef	isDeploymentOf	Identifies the object containing a model of the functions being deployed
SDep	CModel	isContractorOf	Identifies the object containing a model of the information being deployed

Table 2 - Relationships Between the Fedora Object Types

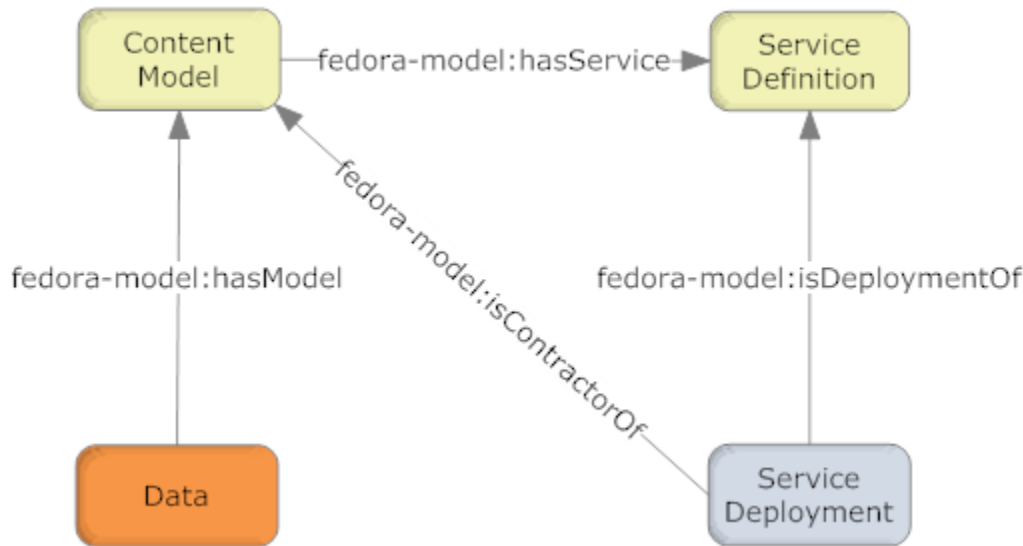


Figure 1 - Fundamental CMA Relationships

Figure 1 illustrates the required relationships between fundamental object types in the CMA. In the CMA object serialization these relations are asserted as RDF statements in the digital objects' RELS-EXT Datastream. These relations are asserted only in the object at the origin of each arrow though typically these relations will be harvested and indexed within utilities such as Semantic Triplestores or relational databases to enable fast query over them, or into caches which permit rapid access to their functionality.

The "hasModel" relation identifies the class of the Data object. There may or may not be a Fedora digital object that corresponds to the identifier. If the identifier refers to an object it must be a CModel object and contain the base content model document. It is expected that many Data objects conform to a single Content Model (and have a relation asserted to the same CModel object). The Content Model characterizes the Data objects that conform to it.

The SDef object describes a Service and the Operations it performs. Defining a Service is the means by which content developers provide customized functionality for their Data objects. A Service consists of one or more Operations, each of which is an endpoint that may be called to execute the Operation. This approach is similar to techniques found in both object-oriented programming and in Web services. The CModel object uses the "hasService" relation to assert that its' class members provides a Service (and its associated Operations). A CModel is free to assert relations to more than one Service. A Service may be related to many CModels.

Deployment of a Service in a repository is accomplished by using the "isDeploymentOf" relation to the SDef object. The Service Deployment (SDep) object is local to a Fedora repository and represents how a Service is implemented by the repository.

Finally, the SDep object asserts the "isContractorOf" to indicate the CModel (effectively the class of Data objects) for which it deploys Services. This permits the SDep to access the Datastreams in the Data object and user parameters when an Operation is called for a Data object.

To see more information about the Fedora Digital Object Model, please go to: [Fedora Digital Object Model](#)

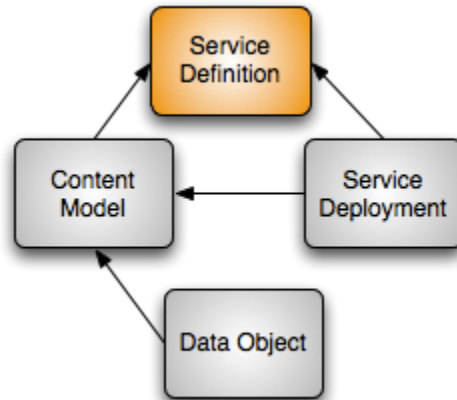
CMA Construction Guide

CMA Construction Guide

The following pages explain how to construct CMA objects for adding services to Fedora data objects. As an alternative, you might be interested in [EZService](#) which enables the creation of the FOXML for these objects from a simple XML definition of the methods and parameters using some XSLT stylesheets.

- [Creating a Service Definition](#)
- [Creating a Service Deployment](#)

Creating a Service Definition



This guide describes how *Service Definition Objects* are constructed in detail. For a conceptual overview of these and other special Fedora object types, please see the [Fedora Digital Object Model](#) document.

Introduction

In Fedora, a Service Definition ("SDef") is a special kind of object that defines a set of operations on objects in a repository. As you will see, that set of operations needs to be expressed in a particular way inside the object in order for Fedora to understand it.

Although you may author Fedora objects in other ways (e.g. by making a series of API calls to your repository), we will assume for this guide that the object is being constructed in FOXML format outside the repository, then ingested in a single step.

Persistent Identifier

As with all Fedora objects, Service Definitions must declare a persistent identifier ("PID") that is unique within the repository. There are no special restrictions on the PID beyond those described in the [Fedora Identifiers](#) document. In FOXML format, the PID is declared in the root element of the XML document as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<foxml:digitalObject xmlns:foxml="info:fedora/fedora-system:def/foxml#"
  VERSION="1.1" PID="demo:MyServiceDefinition">
  <!-- Object Properties -->
  <!-- Datastreams -->
</foxml:digitalObject>
```

Object Properties

There are also no special restrictions on the top-level object properties for Service Definitions. In the following FOXML fragment, we assert that the state of the object is *Active* and the label is *My Service Definition*.

```

<foxml:objectProperties>
  <foxml:property
    NAME="info:fedora/fedora-system:def/model#state"
    VALUE="Active" />
  <foxml:property
    NAME="info:fedora/fedora-system:def/model#label"
    VALUE="My Service Definition" />
</foxml:objectProperties>

```

DC Datastream

The DC datastream describes the object in human terms using the [Dublin Core Metadata Element Set](#). If you don't provide it yourself, Fedora will automatically add a bare-bones DC datastream for the object at ingest time. For an example DC datastream, please see the [FOXML Ingest Example](#).

RELS-EXT Datastream

The RELS-EXT datastream expresses relationships and properties of the object in RDF. An object asserts that it is a Service Definition Object through a special *hasModel* relationship to the *fedora-system:ServiceDefinition-3.0* object as shown below:

```

<foxml:datastream ID="RELS-EXT"
  CONTROL_GROUP="X" STATE="A" VERSIONABLE="true">
  <foxml:datastreamVersion ID="RELS-EXT1.0"
    MIMETYPE="application/rdf+xml"
    FORMAT_URI="info:fedora/fedora-system:FedoraRELSExt-1.0"
    LABEL="RDF Statements about this object">
  <foxml:xmlContent>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:fedora-model="info:fedora/fedora-system:def/model#">
      <rdf:Description rdf:about="info:fedora/demo:MyServiceDefinition">
        <fedora-model:hasModel
          rdf:resource="info:fedora/fedora-system:ServiceDefinition-3.0" />
      </rdf:Description>
    </rdf:RDF>
  </foxml:xmlContent>
</foxml:datastreamVersion>
</foxml:datastream>

```

METHODMAP Datastream

The METHODMAP datastream is the heart of the Service Definition object. It defines the set of methods in the [Fedora Service Definition Method Map format](#). Each method may be defined in this datastream as taking zero or more user-supplied parameters. Each parameter may be required or optional, with a default value and optionally, a set of valid values.

In the following example, we define several methods:

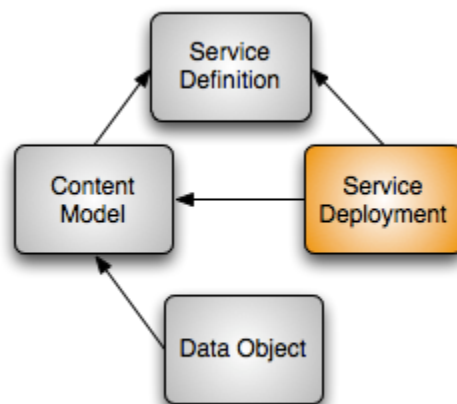
- methodOne - takes no user parameters
- methodTwo - takes one parameter:
 - parm1 - optional, with any possible value and default value of "value1"
- methodThree - takes two parameters:
 - parm1 - optional, with two possible values and a default value of "value1"
 - parm2 - required, with any possible value and a default value of "" (empty string)

```

<foxml:datastream ID="METHODMAP"
  CONTROL_GROUP="X" STATE="A" VERSIONABLE="true">
  <foxml:datastreamVersion ID="METHODMAP1.0"
    FORMAT_URI="info:fedora/fedora-system:FedoraSDefMethodMap-1.0"
    LABEL="Abstract Method Map" MIMETYPE="text/xml">
    <foxml:xmlContent>
      <fmm:MethodMap
        xmlns:fmm="http://fedora.com/nsdlib.org/service/methodmap"
        name="N/A">
        <fmm:Method operationName="methodOne" />
        <fmm:Method operationName="methodTwo">
          <fmm:UserInputParm parmName="parm1" defaultValue="value1"
            required="false" passBy="VALUE" />
        </fmm:Method>
        <fmm:Method operationName="methodThree">
          <fmm:UserInputParm parmName="parm1" defaultValue="value1"
            required="false" passBy="VALUE">
            <fmm:ValidParmValues>
              <fmm:ValidParm value="value1" />
              <fmm:ValidParm value="value2" />
            </fmm:ValidParmValues>
          </fmm:UserInputParm>
          <fmm:UserInputParm parmName="parm2" defaultValue=""
            required="true" passBy="VALUE" />
        </fmm:Method>
      </fmm:MethodMap>
    </foxml:xmlContent>
  </foxml:datastreamVersion>
</foxml:datastream>

```

Creating a Service Deployment



This guide describes how *Service Deployment Objects* are constructed in detail. For a conceptual overview of these and other special Fedora object types, please see the [Fedora Digital Object Model](#) document.

Introduction

In Fedora, a Service Deployment ("SDep") is a special kind of object that describes how the methods of a particular Service Definition are to be deployed for a particular Content Model.

Although you may author Fedora objects in other ways (e.g. by making a series of API calls to your repository), we will assume for this guide that the object is being constructed in FOXML format outside the repository, then ingested in a single step.

Persistent Identifier

As with all Fedora objects, Service Deployments must declare a persistent identifier ("PID") that is unique within the repository. There are no special restrictions on the PID beyond those described in the [Fedora Identifiers](#) document. In FOXML format, the PID is declared in the root

element of the XML document as shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<foxml:digitalObject xmlns:foxml="info:fedora/fedora-system:def/foxml#"
  VERSION="1.1" PID="demo:MyServiceDeployment">
  <!-- Object Properties -->
  <!-- Datastreams -->
</foxml:digitalObject>
```

Object Properties

There are also no special restrictions on the top-level object properties for Service Deployments. In the following FOXML fragment, we assert that the state of the object is *Active* and the label is *My Service Deployment*.

```
<foxml:objectProperties>
  <foxml:property
    NAME="info:fedora/fedora-system:def/model#state"
    VALUE="Active"/>
  <foxml:property
    NAME="info:fedora/fedora-system:def/model#label"
    VALUE="My Service Deployment"/>
</foxml:objectProperties>
```

DC Datastream

The DC datastream describes the object in human terms using the [Dublin Core Metadata Element Set](#). If you don't provide it yourself, Fedora will automatically add a bare-bones DC datastream for the object at ingest time. For an example DC datastream, please see the [FOXML Ingest Example](#).

RELS-EXT Datastream

The RELS-EXT datastream expresses relationships and properties of the object in RDF. An object asserts that it is a Service Deployment Object through a special *hasModel* relationship to the *fedora-system:ServiceDeployment-3.0* object. SDepts must also have two additional relationships asserted in RELS-EXT:

- an *isDeploymentOf* relationship to the associated Service Definition. This tells Fedora which set of methods are deployed by this SDep.
- an *isContractorOf* relationship to the associated Content Model. This tells Fedora which set of objects this SDep is capable of operating on.

```
<foxml:datastream ID="RELS-EXT"
  CONTROL_GROUP="X" STATE="A" VERSIONABLE="true">
  <foxml:datastreamVersion ID="RELS-EXT1.0"
    MIMETYPE="application/rdf+xml"
    FORMAT_URI="info:fedora/fedora-system:FedoraRELSExt-1.0"
    LABEL="RDF Statements about this object">
  <foxml:xmlContent>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:fedora-model="info:fedora/fedora-system:def/model#">
    <rdf:Description rdf:about="info:fedora/demo:MyServiceDeployment">
      <fedora-model:hasModel
        rdf:resource="info:fedora/fedora-system:ServiceDeployment-3.0"/>
      <fedora-model:isDeploymentOf
        rdf:resource="info:fedora/demo:MyServiceDefinition"/>
      <fedora-model:isContractorOf
        rdf:resource="info:fedora/demo:MyContentModel"/>
    </rdf:Description>
  </rdf:RDF>
  </foxml:xmlContent>
</foxml:datastreamVersion>
</foxml:datastream>
```

METHODMAP Datastream

The METHODMAP datastream in the Service Deployment object is similar to the one stored in the corresponding Service Definition, but it also indicates which datastreams (if any) are required as input to each deployed method. This information is specified in the [Service Deployment](#)

Method Map format, which is an extension of the *Service Definition Method Map* format.

When creating a Service Deployment method map, it is helpful to copy the method map from the corresponding Service Definition, then add information as necessary.

Continuing with the example method map set out in *Creating a Service Definition*, below we will specify that:

- *methodOne* accepts a single datastream as input, whose ID is "FOO".
- *methodTwo* accepts no datastreams as input.
 - it includes the user input parameter "param1" specified in the SDef
 - it includes a default input parameter "uri" with default value \$objuri, allowing the URI of the Fedora object to be used as a parameter when calling an external service
- *methodThree* accepts three datastreams as input, whose IDs are "FOO", "BAR", and "BAZ".
 - it also includes the user input parameters specified in the SDef
 - it also includes a default input parameter "pid" with default value \$pid, allowing the PID of the Fedora object to be used as a parameter when calling an external service



Default input parameters

Default input parameters can be specified in the method definitions. These are parameters that are then available to the WSDL's specification of the external service in a similar way to user input parameters and datastreams. The default value of the parameter can either be specified as some constant value, or the following "special" variables may be used which will be substituted when the method is invoked:

- \$pid - will be substituted with the (URL-encoded) value of the Fedora object's PID, eg `changeme:1234`
- \$objuri - will be substituted with the (URL-encoded) value of the Fedora object's URI, eg `info:fedora/changeme:1234`

Note: Differences from the METHODMAP specified in the corresponding Service Definition are indicated below with a preceding "+" sign.

```

<foxml:datastream ID="METHODMAP"
  CONTROL_GROUP="X" STATE="A" VERSIONABLE="true">
  <foxml:datastreamVersion ID="METHODMAP1.0"
+   FORMAT_URI="info:fedora/fedora-system:FedoraSDepMethodMap-1.1"
+   LABEL="Deployment Method Map" MIMETYPE="text/xml">
    <foxml:xmlContent>
      <fmm:MethodMap
        xmlns:fmm="http://fedora.com/nsdlib.org/service/methodmap"
        name="N/A">
        <fmm:Method operationName="methodOne"
+         wsdlMsgName="methodOneRequest"
+         wsdlMsgOutput="response">
          <fmm:DatastreamInputParm paramName="FOO"
+           passBy="URL_REF" required="true"/>
          <fmm:MethodReturnType wsdlMsgName="response"
+           wsdlMsgTOMIME="N/A"/>
        </fmm:Method>
        <fmm:Method operationName="methodTwo"
+         wsdlMsgName="methodTwoRequest"
+         wsdlMsgOutput="response">
          <fmm:UserInputParm paramName="parm1" defaultValue="value1"
+           required="false" passBy="VALUE"/>
          <fmm:DefaultInputParm paramName="uri" defaultValue="$objuri"
+           passBy="VALUE" required="true"/>
          <fmm:MethodReturnType wsdlMsgName="response"
+           wsdlMsgTOMIME="N/A"/>
        </fmm:Method>
        <fmm:Method operationName="methodThree"
+         wsdlMsgName="methodThreeRequest"
+         wsdlMsgOutput="response">
          <fmm:UserInputParm paramName="parm1" defaultValue="value1"
+           required="false" passBy="VALUE">
            <fmm:ValidParmValues>
              <fmm:ValidParm value="value1"/>
              <fmm:ValidParm value="value2"/>
            </fmm:ValidParmValues>
          </fmm:UserInputParm>
          <fmm:UserInputParm paramName="parm2" defaultValue=""
+           required="true" passBy="VALUE"/>
          <fmm:DefaultInputParm paramName="pid" defaultValue="$pid"
+           passBy="VALUE" required="true"/>
          <fmm:DatastreamInputParm paramName="FOO"
+           passBy="URL_REF" required="true"/>
          <fmm:DatastreamInputParm paramName="BAR"
+           passBy="URL_REF" required="true"/>
          <fmm:DatastreamInputParm paramName="BAZ"
+           passBy="URL_REF" required="true"/>
          <fmm:MethodReturnType wsdlMsgName="response"
+           wsdlMsgTOMIME="N/A"/>
        </fmm:Method>
      </fmm:MethodMap>
    </foxml:xmlContent>
  </foxml:datastreamVersion>
</foxml:datastream>

```

DSINPUTSPEC Datastream

The DSINPUTSPEC datastream specifies additional information about each datastream. It is written in the [Fedora Datastream Input Specification](#) format.


```

<foxml:datastream ID="DSINPUTSPEC"
  CONTROL_GROUP="X" STATE="A" VERSIONABLE="true">
  <foxml:datastreamVersion ID="DSINPUTSPEC1.0"
    MIMETYPE="text/xml"
    FORMAT_URI="info:fedora/fedora-system:FedoraDSInputSpec-1.1"
    Label="Datastream Input Specification">
    <foxml:xmlContent>
      <fbs:DSInputSpec
        xmlns:fbs="http://fedora.com/nsdlib.org/service/bindspec"
        label="N/A">
        <fbs:DSInput wsdlMsgPartName="FOO"
          DSMax="1" DSMin="1" DSOrdinality="false">
          <fbs:DSInputLabel/>N/A</fbs:DSInputLabel>
          <fbs:DSMIME>N/A</fbs:DSMIME>
          <fbs:DSInputInstruction>N/A</fbs:DSInputInstruction>
        </fbs:DSInput>
        <fbs:DSInput wsdlMsgPartName="BAR"
          DSMax="1" DSMin="1" DSOrdinality="false">
          <fbs:DSInputLabel/>N/A</fbs:DSInputLabel>
          <fbs:DSMIME>N/A</fbs:DSMIME>
          <fbs:DSInputInstruction>N/A</fbs:DSInputInstruction>
        </fbs:DSInput>
        <fbs:DSInput wsdlMsgPartName="BAZ" pid="demo:MyContentModel"
          DSMax="1" DSMin="1" DSOrdinality="false">
          <fbs:DSInputLabel/>N/A</fbs:DSInputLabel>
          <fbs:DSMIME>N/A</fbs:DSMIME>
          <fbs:DSInputInstruction>N/A</fbs:DSInputInstruction>
        </fbs:DSInput>
      </fbs:DSInputSpec>
    </foxml:xmlContent>
  </foxml:datastreamVersion>
</foxml:datastream>

```

Notice that the BAZ datastream input specification above has an extra attribute, "pid" specified. When present, the pid attribute tells Fedora that the datastream to be used as input is located inside a specific object. In this case, it is the BAZ datastream in the "demo:MyContentModel" object. Normally, each input datastream is expected to come from the particular data object from which the method is invoked. When a pid is specified, the associated datastream effectively serves as a constant.

WSDL Datastream

The WSDL datastream brings everything together, describing to Fedora exactly what needs to be done under the hood when a dissemination request is made. For each method, the WSDL provides enough information to Fedora so that it may do a URL replacement in order to fulfill a request.

Fedora does not work with arbitrary WSDL in this datastream; it must be in a very specific form. Notably, Fedora currently only supports performing disseminations via HTTP GET. You may use the following as a template for your own WSDL datastreams, changing the messages, port type, and binding sections as appropriate.

To customize this for your own use:

- Leave the "response" wsdl:message as-is, but replace the others with your own. You should have a wsdl:message element for each method defined by the Service Definition. By convention, the name of each message should be the method name followed by the word "Request" (e.g. "methodOneRequest"). Within each wsdl:message element, add a wsdl:part for each user and/or datastream input parameter, and specify all types as "this:inputType".
- Within the wsdl:portType section, replace each existing operation with your own. Each operation should reference the corresponding request method as the input, and point to "this:response" as the output.
- Finally, in the wsdl:binding section, replace each existing operation with your own. In this section, the location attribute is the important bit. The location should be given as a URL template, where all occurrences of (VARIABLE) are automatically replaced with the appropriate value at runtime. If VARIABLE denotes a datastream, it will be replaced with a URL reference to the datastream content at runtime. If VARIABLE denotes a user input parameter, it will be replaced with the URL-encoded value of the variable at runtime. If VARIABLE is "PID" and your METHODMAP specifies \$PID as the default value for the PID variable as in the example METHODMAP above, it will be replaced with the data object's URL-encoded PID at runtime.

```

<foxml:datastream ID="WSDL"
  CONTROL_GROUP="X" STATE="A" VERSIONABLE="true">
  <foxml:datastreamVersion ID="WSDL1.0"
    MIMETYPE="text/xml"

```

```
FORMAT_URI="http://schemas.xmlsoap.org/wsd/"
LABEL="WSDL Bindings">
<foxml:xmlContent>
  <wsdl:definitions name="definitions"
    targetNamespace="urn:thisNamespace"
    xmlns:http="http://schemas.xmlsoap.org/wsd/http/"
    xmlns:mime="http://schemas.xmlsoap.org/wsd/mime/"
    xmlns:soap="http://schemas.xmlsoap.org/wsd/soap/"
    xmlns:soapenc="http://schemas.xmlsoap.org/wsd/soap/encoding"
    xmlns:this="urn:thisNamespace"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsd/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <wsdl:types>
      <xsd:schema targetNamespace="urn:thisNamespace">
        <xsd:simpleType name="inputType">
          <xsd:restriction base="xsd:string"/>
        </xsd:simpleType>
      </xsd:schema>
    </wsdl:types>
    <wsdl:message name="methodOneRequest">
      <wsdl:part name="FOO" type="this:inputType"/>
    </wsdl:message>
    <wsdl:message name="methodTwoRequest">
      <wsdl:part name="param1" type="this:inputType"/>
      <wsdl:part name="uri" type="this:inputType"/>
    </wsdl:message>
    <wsdl:message name="methodThreeRequest">
      <wsdl:part name="param1" type="this:inputType"/>
      <wsdl:part name="param2" type="this:inputType"/>
      <wsdl:part name="pid" type="this:inputType"/>
      <wsdl:part name="FOO" type="this:inputType"/>
      <wsdl:part name="BAR" type="this:inputType"/>
      <wsdl:part name="BAZ" type="this:inputType"/>
    </wsdl:message>
    <wsdl:message name="response">
      <wsdl:part name="response" type="xsd:base64Binary"/>
    </wsdl:message>
    <wsdl:portType name="portType">
      <wsdl:operation name="methodOne">
        <wsdl:input message="this:methodOneRequest"/>
        <wsdl:output message="this:response"/>
      </wsdl:operation>
      <wsdl:operation name="methodTwo">
        <wsdl:input message="this:methodTwoRequest"/>
        <wsdl:output message="this:response"/>
      </wsdl:operation>
      <wsdl:operation name="methodThree">
        <wsdl:input message="this:methodThreeRequest"/>
        <wsdl:output message="this:response"/>
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:service name="service">
      <wsdl:port binding="this:binding" name="port">
        <http:address location="LOCAL"/>
      </wsdl:port>
    </wsdl:service>
    <wsdl:binding name="binding" type="this:portType">
      <http:binding verb="GET"/>
      <wsdl:operation name="methodOne">
        <http:operation location="(FOO)"/>
        <wsdl:input>
          <http:urlReplacement/>
        </wsdl:input>
        <wsdl:output>
          <mime:content type="N/A"/>
        </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="methodTwo">
        <http:operation
```

```
location="http://local.fedora.server/fedora/risearch?format=(parml)&type=triples&lang=spo&que
<wsdl:input>
    <http:urlReplacement/>
</wsdl:input>
<wsdl:output>
    <mime:content type="N/A"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="methodThree">
    <http:operation
```

```
location="http://example.org/service?a=(parml)&b=(parm2)&c=(parm3)&d=(FOO)&e=(BAR)&f=
<wsdl:input>
    <http:urlReplacement/>
</wsdl:input>
<wsdl:output>
    <mime:content type="N/A"/>
</wsdl:output>
</wsdl:operation>
</wsdl:binding>
</wsdl:definitions>
```

```
</foxml:xmlContent>
</foxml:datastreamVersion>
</foxml:datastream>
```

Introduction to FOXML

What is FOXML?

FOXML is a simple XML format that directly expresses the **Fedora Digital Object Model**. As of Fedora 2.0, digital objects are *stored* internally in a Fedora repository in the FOXML format. In addition, FOXML can be used for ingesting and exporting objects to and from Fedora repositories. The Fedora extension of METS will continue to be supported as an ingest and export format.

At the highest level, the FOXML XML schema defines elements that correspond directly to the fundamental Fedora digital object components (see recent [paper](#) on Fedora). Below is a brief sketch of these elements.

```
<digitalObject PID="uniqueID">

  <!-- there are a set of core object properties -->
  <objectProperties>
    <property/>
    <property/>
    ...
  </objectProperties>

  <!-- there can be zero or more datastreams -->
  <datastream>
    <datastreamVersion/>
    <datastreamVersion/>
    ...
  </datastream>

</digitalObject>
```

Why FOXML?

The introduction of FOXML was motivated by several requirements: (1) simplicity, (2) optimization and performance, and (3) flexibility in evolving Fedora. Regarding simplicity, user feedback called for a conceptually easy mapping of the Fedora concepts to an XML format. Users wanted an obvious sense of how to create Fedora ingest files, especially those who are not familiar with formats such as METS. Regarding optimization and performance, the FOXML schema was designed to improve repository performance, both at ingest and during disseminations. Overall ingest performance was positively affected with the introduction of FOXML, especially in the validation phases. Regarding flexibility, establishing FOXML as the internal storage format for Fedora objects enables easier evolution of functionality in the Fedora repository, without requiring ongoing extensions to other community formats.

What does a FOXML instance document look like?

An example is worth a thousand words. Therefore, we have provided a fully-annotated digital object encoded in FOXML. This example is presented from the perspective of how a digital object looks when it is stored inside a Fedora repository. However, the documentation also indicates how to encode an object in FOXML for ingesting into the repository. There are certain data attributes that can be omitted in ingest files since the Fedora Repository service assigns them.

To learn more, please consult the FOXML [reference example](#) now!

Where is the FOXML XML schema?

An official published version of the [FOXML XML schema](#) is also published on the Fedora Commons web site. Also, a copy of the schema is provided with the Fedora open-source distribution. The Fedora repository service validates all Fedora objects against this schema before objects are permanently stored in the repository.

Are other XML Formats supported by Fedora?

Yes! The Fedora repository service is designed to be able to accommodate different XML formats for encoding digital objects through its ingest and export operations, available via the Fedora management service interface (API-M) and command-line tools.

Currently, Fedora supports ingest and export of objects in the following formats:

- FOXML 1.1 (info:fedora/fedora-system:FOXML-1.1)
- FOXML 1.0 (info:fedora/fedora-system:FOXML-1.0)
- METS 1.1 (info:fedora/fedora-system:METSFedoraExt-1.1)
- METS 1.0 (info:fedora/fedora-system:METSFedoraExt-1.0)
- ATOM 1.1 (info:fedora/fedora-system:ATOM-1.1)
- ATOM Zip 1.1 (info:fedora/fedora-system:ATOMZip-1.1)

FOXML Ingest Example

Sample FOXML Ingest Encoding

```
<?xml version="1.0" encoding="UTF-8"?>
<!--*****
FOXML 1.1 INGEST EXAMPLE:
-->
<!-- This is an example of a FOXML object as it should be encoded for ingest in the repository. Note
that attributes -->
<!-- that are automatically assigned by the Fedora repository are omitted. Notable omissions are the
created and last -->
<!-- modified dates in the object properties, the created date, size, and versionable attributes on
datastreams. -->
<!--*****
NOTE!! Please see the FOXML Reference Example object linked from the "Introduction to FOXML" document
in the -->
<!-- Fedora System Documentation. This will give an element-by-element explanation of a FOXML
instance document. -->
<!--*****
xmlns:foxml="info:fedora/fedora-system:def/foxml#" VERSION="1.1" PID="demo:999"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="info:fedora/fedora-system:def/foxml#
http://www.fedora.info/definitions/1/0/foxml1-1.xsd">

<foxml:objectProperties>
  <foxml:property NAME="info:fedora/fedora-system:def/model#state" VALUE="A"/>
  <foxml:property NAME="info:fedora/fedora-system:def/model#label" VALUE="FOXML Reference Example"/>
</foxml:objectProperties>
<foxml:datastream ID="DC" STATE="A" CONTROL_GROUP="X">
  <foxml:datastreamVersion FORMAT_URI="http://www.openarchives.org/OAI/2.0/oai_dc/"
    ID="DC.0" MIMETYPE="text/xml"
    LABEL="Dublin Core Record for this object">

    <foxml:xmlContent>
      <oai_dc:dc xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
xmlns:dc="http://purl.org/dc/elements/1.1/">
        <dc:title>FOXML Reference Object</dc:title>
        <dc:creator>Sandy Payette</dc:creator>
        <dc:subject>Fedora documentation</dc:subject>
        <dc:description>
          FOXML showing how a digital object is encoded for persistent storage in a Fedora
repository
        </dc:description>
        <dc:publisher>Cornell CIS</dc:publisher>
        <dc:identifier>demo:999</dc:identifier>
      </oai_dc:dc>
    </foxml:xmlContent>
  </foxml:datastreamVersion>
</foxml:datastream>
<foxml:datastream ID="RELS-EXT" CONTROL_GROUP="X">
  <foxml:datastreamVersion FORMAT_URI="info:fedora/fedora-system:FedoraRELSExt-1.0"
    ID="RELS-EXT.0" MIMETYPE="application/rdf+xml"
    LABEL="RDF Statements about this object">

    <foxml:xmlContent>
      <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:fedora="info:fedora/fedora-system:def/relations-external#"
xmlns:myns="http://www.nsd.org/ontologies/relationships#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
```

```

xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/">
  <rdf:Description rdf:about="info:fedora/demo:999">
    <!-- This object ("info:fedora/demo:999") is a member of Collection #1
(info:fedora/test:collection1) -->
    <fedora:isMemberOfCollection rdf:resource="info:fedora/test:collection1"/>
    <!-- ... and it is also a member of Collection #2 (info:fedora/test:collection2) -->
    <fedora:isMemberOfCollection rdf:resource="info:fedora/test:collection2"/>
    <!-- You can also make your own relationship assertions in your own namespace...-->
    <myns:isPartOf rdf:resource="info:fedora/mystuff:100"/>
  </rdf:Description>
</rdf:RDF>
</foxml:xmlContent>
</foxml:datastreamVersion>
</foxml:datastream>
<foxml:datastream CONTROL_GROUP="E" ID="IMAGE" STATE="A">
  <foxml:datastreamVersion ID="IMAGE.0" MIMETYPE="image/x-mrsid-image"
    LABEL="Image of Pavilion III, University of Virginia">
    <foxml:contentLocation REF="http://iris.lib.virginia.edu/mrsid/mrsid_images/iva/archerp01.sid"
TYPE="URL"/>
  </foxml:datastreamVersion>
</foxml:datastream>
<foxml:datastream CONTROL_GROUP="R" ID="DRAWING-BEST" STATE="A">
  <foxml:datastreamVersion ID="DRAWING-BEST.0" MIMETYPE="image/jpeg"
    LABEL="Architectural Drawing Pavilion III (veryhigh res)">
    <foxml:contentLocation REF="http://icarus.lib.virginia.edu/images/iva/archerd05high.jpg"
TYPE="URL"/>
  </foxml:datastreamVersion>
</foxml:datastream>
<foxml:datastream CONTROL_GROUP="M" ID="DRAWING-BETTER" STATE="A">
  <foxml:datastreamVersion ID="DRAWING-BETTER.0" MIMETYPE="image/jpeg"
    LABEL="Architectural Drawing Pavilion III (med res)">
    <foxml:contentLocation REF="http://icarus.lib.virginia.edu/images/iva/archerd05medium1.jpg"
TYPE="URL"/>
  </foxml:datastreamVersion>
</foxml:datastream>
<foxml:datastream CONTROL_GROUP="M" ID="DRAWING-ICON" STATE="A">
  <foxml:datastreamVersion ID="DRAWING-ICON.0" MIMETYPE="image/jpeg" LABEL="Architectural Drawing
Pavilion III">
    <foxml:contentLocation REF="http://icarus.lib.virginia.edu/images/iva/archerd05small.jpg"
TYPE="URL"/>
  </foxml:datastreamVersion>
  <foxml:datastreamVersion ID="DRAWING-ICON.1" MIMETYPE="image/jpeg"
    LABEL="Architectural Drawing Pavilion III (thumbnail icon)"
ALT_IDS="doi:10.1234/123">
    <foxml:contentLocation REF="http://icarus.lib.virginia.edu/images/iva/archerd05small.jpg"
TYPE="URL"/>
  </foxml:datastreamVersion>
</foxml:datastream>
<foxml:datastream ID="UVATECH" STATE="A" CONTROL_GROUP="X">
  <foxml:datastreamVersion ID="UVATECH.0" MIMETYPE="text/xml"
FORMAT_URI="info:fedora/format:xml:uvalibadmin"
    LABEL="UVA Technical Metadata Record">
    <foxml:xmlContent>
      <uvalibadmin:admin xmlns:uvalibadmin="http://virginia.lib.edu/uvalibadmin:tech">
        <uvalibadmin:technical>
          <uvalibadmin:format>image/jpeg</uvalibadmin:format>
          <uvalibadmin:compression>LZW</uvalibadmin:compression>
          <uvalibadmin:bitDepth BITS="" />
          <uvalibadmin:colorSpace/>
          <uvalibadmin:colorProfile CPLOCAT="" CPFILE="" />
          <uvalibadmin:resolution>600</uvalibadmin:resolution>
        </uvalibadmin:technical>
      </uvalibadmin:admin>
    </foxml:xmlContent>
  </foxml:datastreamVersion>
</foxml:datastream>

```

```
</foxml:datastreamVersion>
</foxml:datastream>
</foxml:digitalObject>
```

Fedora Atom



Experimental

This is an experimental feature and subject to change in future versions.

Introduction

Fedora Atom is a serialization of a Fedora Digital Object using the Atom Syndication Format [RFC 4287](#) in conjunction with the Atom Threading Extensions [RFC 4685](#).

Many of the programming languages and platforms used with Fedora already provide libraries and tools for working with Atom feeds. By offering an Atom serialization of Fedora objects, these now become tools and libraries for authoring, browsing and validating Fedora digital objects as well.

Serialization Formats

Fedora Atom

A Fedora Digital Object is represented as an atom:feed element and Datastreams are represented as an atom:entry elements.

```
<feed xmlns="http://www.w3.org/2005/Atom">
  <id>info:fedora/demo:5</id>
  <title type="text">Simple Image Demo</title>
  <updated>2008-07-02T05:09:42.015Z</updated>
  <author><name>fedoraAdmin</name></author>
  <category scheme="info:fedora/fedora-system:def/model#state"
    term="Active"/>
  <category scheme="info:fedora/fedora-system:def/model#createdDate"
    term="2008-07-02T05:09:42.015Z"/>
```

Object properties are represented using various atom:feed elements. In the abbreviated example above, the object's pid, label, ownerId and lastModifiedDate are represented using the feed's id, title, author, and updated elements respectively. atom:category elements are used to represent object properties such as state and createdDate.

```

<entry>
  <id>info:fedora/demo:5/DC</id>
  <title type="text">DC</title>
  <updated>2008-07-02T05:09:43.375Z</updated>
  <link href="info:fedora/demo:5/DC/2008-07-02T05:09:43.375Z" rel="alternate"/>
  <category scheme="info:fedora/fedora-system:def/model#state"
    term="A"/>
  <category scheme="info:fedora/fedora-system:def/model#controlGroup"
    term="X"/>
  <category scheme="info:fedora/fedora-system:def/model#versionable"
    term="true"/>
</entry>

<entry xmlns:thr="http://purl.org/syndication/thread/1.0">
  <id>info:fedora/demo:5/DC/2008-07-02T05:09:43.375Z</id>
  <title type="text">DC1.0</title>
  <updated>2008-07-02T05:09:43.375Z</updated>
  <thr:in-reply-to ref="info:fedora/demo:5/DC"/>
  <category scheme="info:fedora/fedora-system:def/model#formatURI"
    term="http://www.openarchives.org/OAI/2.0/oai_dc/" />
  <category scheme="info:fedora/fedora-system:def/model#label"
    term="Dublin Core Record for this object"/>
  <content type="text/xml">
    <oai_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/">
      <dc:title>Coliseum in Rome</dc:title>
    </oai_dc:dc>
  </content>
</entry>

```

The hierarchy of Datastreams and their Datastream versions is represented via the Atom Threading Extensions. For convenience, a Datastream entry references its latest Datastream version entry with an `atom:link` element. In the example above, the DC datastream entry refers to its most recent version as follows:

```
<link href="info:fedora/demo:5/DC/2008-07-02T05:09:43.375Z" rel="alternate"/>
```

Each Datastream version refers to its parent Datastream via a `thr:in-reply-to` element. In the example above, the entry for the DC Datastream version refers to its parent as follows:

```
<thr:in-reply-to ref="info:fedora/demo:5/DC"/>
```

Fedora Atom Zip

Fedora Atom Zip is a serialization of a Fedora digital object using the ZIP file format and a Fedora Atom manifest document. Inline and managed datastream content are packaged in the ZIP archive as individual files.

The manifest must be a Fedora Atom document named "atommanifest.xml".

Format URIs

Fedora Atom and Fedora Atom Zip are identified respectively with the following URIs:

- `info:fedora/fedora-system:ATOM-1.1`
- `info:fedora/fedora-system:ATOMZip-1.1`

Examples

A complete set of demonstration objects in both Fedora Atom and Fedora Atom Zip are included in the Fedora distribution. Please see the [Demonstrations](#) documentation for more information.

References

- [RFC 4287] – [The Atom Syndication Format](#), Mark Nottingham, Robert Sayre, 2005.
- [RFC 4685] – [Atom Threading Extensions](#), James Snell, 2006.

ATOM Ingest Example

Sample ATOM Ingest Encoding

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <id>info:fedora/demo:5</id>
  <title type="text">Data Object (Coliseum) for Local Simple Image Demo</title>
  <updated>2008-07-02T05:09:42.015Z</updated>
  <author>
    <name>fedoraAdmin</name>
  </author>
  <category scheme="info:fedora/fedora-system:def/model#state" term="Active"/>
  <category scheme="info:fedora/fedora-system:def/model#createdDate" term="2008-07-02T05:09:42.015Z"/>
  <icon>http://www.fedora-commons.org/images/logo_vertical_transparent_200_251.png</icon>
  <entry>
    <id>info:fedora/demo:5/DC</id>
    <title type="text">DC</title>
    <updated>2008-07-02T05:09:43.375Z</updated>
    <link href="info:fedora/demo:5/DC/2008-07-02T05:09:43.375Z" rel="alternate"/>
    <category scheme="info:fedora/fedora-system:def/model#state" term="A"/>
    <category scheme="info:fedora/fedora-system:def/model#controlGroup" term="X"/>
    <category scheme="info:fedora/fedora-system:def/model#versionable" term="true"/>
  </entry>
  <entry xmlns:thr="http://purl.org/syndication/thread/1.0">
    <id>info:fedora/demo:5/DC/2008-07-02T05:09:43.375Z</id>
    <title type="text">DC1.0</title>
    <updated>2008-07-02T05:09:43.375Z</updated>
    <thr:in-reply-to ref="info:fedora/demo:5/DC"/>
    <category scheme="info:fedora/fedora-system:def/model#formatURI"
term="http://www.openarchives.org/OAI/2.0/oai_dc/" />
    <category scheme="info:fedora/fedora-system:def/model#label" term="Dublin Core Record for this
object"/>
    <content type="text/xml">
      <oai_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/">
        <dc:title>Coliseum in Rome</dc:title>
        <dc:creator>Thornton Staples</dc:creator>
        <dc:subject>Architecture, Roman</dc:subject>
        <dc:description>Image of Coliseum in Rome</dc:description>
        <dc:publisher>University of Virginia Library</dc:publisher>
        <dc:format>image/jpeg</dc:format>
        <dc:identifier>demo:5</dc:identifier>
      </oai_dc:dc>
    </content>
  </entry>
  <entry>
    <id>info:fedora/demo:5/RELS-EXT</id>
    <title type="text">RELS-EXT</title>
    <updated>2008-07-02T05:09:43.375Z</updated>
    <link href="info:fedora/demo:5/RELS-EXT/2008-07-02T05:09:43.375Z" rel="alternate"/>
    <category scheme="info:fedora/fedora-system:def/model#state" term="A"/>
    <category scheme="info:fedora/fedora-system:def/model#controlGroup" term="X"/>
    <category scheme="info:fedora/fedora-system:def/model#versionable" term="true"/>
  </entry>
  <entry xmlns:thr="http://purl.org/syndication/thread/1.0">
    <id>info:fedora/demo:5/RELS-EXT/2008-07-02T05:09:43.375Z</id>
    <title type="text">RELS-EXT1.0</title>
    <updated>2008-07-02T05:09:43.375Z</updated>
    <thr:in-reply-to ref="info:fedora/demo:5/RELS-EXT"/>
    <category scheme="info:fedora/fedora-system:def/model#formatURI"
term="info:fedora/fedora-system:FedoraRELSExt-1.0"/>
    <category scheme="info:fedora/fedora-system:def/model#label" term="RDF Statements about this
object"/>
    <content type="application/rdf+xml">
      <rdf:RDF xmlns:fedora-model="info:fedora/fedora-system:def/model#"

```

```
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="info:fedora/demo:5">
    <fedora-model:hasModel rdf:resource="info:fedora/demo:UVA_STD_IMAGE_1"/>
  </rdf:Description>
</rdf:RDF>
</content>
</entry>
<entry>
  <id>info:fedora/demo:5/THUMBRES_IMG</id>
  <title type="text">THUMBRES_IMG</title>
  <updated>2008-07-02T05:09:43.375Z</updated>
  <link href="info:fedora/demo:5/THUMBRES_IMG/2008-07-02T05:09:43.375Z" rel="alternate"/>
  <category scheme="info:fedora/fedora-system:def/model#state" term="A"/>
  <category scheme="info:fedora/fedora-system:def/model#controlGroup" term="M"/>
  <category scheme="info:fedora/fedora-system:def/model#versionable" term="true"/>
</entry>
<entry xmlns:thr="http://purl.org/syndication/thread/1.0">
  <id>info:fedora/demo:5/THUMBRES_IMG/2008-07-02T05:09:43.375Z</id>
  <title type="text">THUMBRES_IMG1.0</title>
  <updated>2008-07-02T05:09:43.375Z</updated>
  <thr:in-reply-to ref="info:fedora/demo:5/THUMBRES_IMG"/>
  <category scheme="info:fedora/fedora-system:def/model#label" term="Thorny's Coliseum thumbnail jpg image"/>
  <summary type="text">THUMBRES_IMG1.0</summary>
  <content
    src="http://localhost:8080/fedora-demo/simple-image-demo/coliseum-thumb.jpg" type="image/jpeg"/>
</entry>
<entry>
  <id>info:fedora/demo:5/MEDRES_IMG</id>
  <title type="text">MEDRES_IMG</title>
  <updated>2008-07-02T05:09:43.375Z</updated>
  <link href="info:fedora/demo:5/MEDRES_IMG/2008-07-02T05:09:43.375Z" rel="alternate"/>
  <category scheme="info:fedora/fedora-system:def/model#state" term="A"/>
  <category scheme="info:fedora/fedora-system:def/model#controlGroup" term="M"/>
  <category scheme="info:fedora/fedora-system:def/model#versionable" term="true"/>
</entry>
<entry xmlns:thr="http://purl.org/syndication/thread/1.0">
  <id>info:fedora/demo:5/MEDRES_IMG/2008-07-02T05:09:43.375Z</id>
  <title type="text">MEDRES_IMG1.0</title>
  <updated>2008-07-02T05:09:43.375Z</updated>
  <thr:in-reply-to ref="info:fedora/demo:5/MEDRES_IMG"/>
  <category scheme="info:fedora/fedora-system:def/model#label" term="Thorny's Coliseum medium jpg image"/>
  <summary type="text">MEDRES_IMG1.0</summary>
  <content
    src="http://localhost:8080/fedora-demo/simple-image-demo/coliseum-medium.jpg"
type="image/jpeg"/>
</entry>
<entry>
  <id>info:fedora/demo:5/HIGHRES_IMG</id>
  <title type="text">HIGHRES_IMG</title>
  <updated>2008-07-02T05:09:43.375Z</updated>
  <link href="info:fedora/demo:5/HIGHRES_IMG/2008-07-02T05:09:43.375Z" rel="alternate"/>
  <category scheme="info:fedora/fedora-system:def/model#state" term="A"/>
  <category scheme="info:fedora/fedora-system:def/model#controlGroup" term="M"/>
  <category scheme="info:fedora/fedora-system:def/model#versionable" term="true"/>
</entry>
<entry xmlns:thr="http://purl.org/syndication/thread/1.0">
  <id>info:fedora/demo:5/HIGHRES_IMG/2008-07-02T05:09:43.375Z</id>
  <title type="text">HIGHRES_IMG1.0</title>
  <updated>2008-07-02T05:09:43.375Z</updated>
  <thr:in-reply-to ref="info:fedora/demo:5/HIGHRES_IMG"/>
  <category scheme="info:fedora/fedora-system:def/model#label" term="Thorny's Coliseum high jpg image"/>
  <summary type="text">HIGHRES_IMG1.0</summary>
  <content
    src="http://localhost:8080/fedora-demo/simple-image-demo/coliseum-high.jpg" type="image/jpeg"/>
</entry>
<entry>
  <id>info:fedora/demo:5/VERYHIGHRES_IMG</id>
```

```
<title type="text">VERYHIGHRES_IMG</title>
<updated>2008-07-02T05:09:43.375Z</updated>
<link href="info:fedora/demo:5/VERYHIGHRES_IMG/2008-07-02T05:09:43.375Z" rel="alternate"/>
<category scheme="info:fedora/fedora-system:def/model#state" term="A"/>
<category scheme="info:fedora/fedora-system:def/model#controlGroup" term="M"/>
<category scheme="info:fedora/fedora-system:def/model#versionable" term="true"/>
</entry>
<entry xmlns:thr="http://purl.org/syndication/thread/1.0">
  <id>info:fedora/demo:5/VERYHIGHRES_IMG/2008-07-02T05:09:43.375Z</id>
  <title type="text">VERYHIGHRES_IMG1.0</title>
  <updated>2008-07-02T05:09:43.375Z</updated>
  <thr:in-reply-to ref="info:fedora/demo:5/VERYHIGHRES_IMG"/>
  <category scheme="info:fedora/fedora-system:def/model#label" term="Thorny's Coliseum veryhigh jpg
image"/>
  <summary type="text">VERYHIGHRES_IMG1.0</summary>
  <content
    src="http://localhost:8080/fedora-demo/simple-image-demo/coliseum-veryhigh.jpg"
```

```
type=" image/jpeg "/>
</entry>
</feed>
```

Fedora METS

Table of Contents

1. Introduction
2. Object Encoding Rules

Introduction

As usual an example is worth a thousand words. So, please refer to the sample Fedora object that is encoded for ingest in METS 1.1: [mets-ingest-example.xml](#).

Fedora supports ingest of objects in a Fedora-specific extension of Metadata Encoding and Transmission Standard (METS). More information on METS can be found at <http://www.loc.gov/standards/mets/>. For specific information about the Fedora extension to METS, please see the [METS 1.1 schema](#).

Since METS was designed to be very generic and support a variety of uses, the rules of the METS Schema are very general-purpose. Fedora objects must conform to other rules that are beyond the scope of what is expressed in the METS schema. Therefore, the Fedora Object XML submissions will also be validated against a set of Fedora-specific rules that are expressed using the Schematron language. Internally, the repository will use Schematron to enforce these rules on incoming XML submission packages. The Schematron rules are expressed in XML and can be found in the Fedora server distribution at: `%FEDORA_HOME%\server\schematron\metsExtRules1-1.xml`.

For convenience and ease of understanding we have enumerated the Fedora rules in plain English below.

Object Encoding Rules

Encoding by hand requires a pretty good understanding of METS, although it can be done by following the patterns in the demo objects that come with the Fedora distribution. Demo objects are located at: `%FEDORA_HOME%\client\demo`.

General attributes

- On METS root element, the OBJID attribute will represent the Fedora object PID. Normally, this should be left empty so that the Fedora repository can generate a new PID. However, you can assign test/demo PIDs by inserting a value in OBJID that begins with "demo:" or "test:" for example, "demo:100"
- On METS root element, the value of the EXT_VERSION attribute must be "1.1".
- On METS root element, the value of LABEL serves as the official description of the object. If there is no Dublin Core record present in the object, the Fedora repository will use this label to populate the title element of a baseline Dublin Core record for the object.
- On METS root element, the PROFILE element can be used by institutions to classify different types of Fedora data objects.
- On the METS:metsHdr element the CREATEDATE attribute should be omitted since the Fedora repository will assign this at ingest time. Fedora dates are in the ISO 8601 format in milliseconds and with UTC time as follows: yyyy-MM-ddTHH:mm:ss.SSSZ. The same thing goes for LASTMODDATE.
- On the METS:metsHdr element the RECORDSTATUS should be set to "A", "I", or "D" to indicate that the object is in the "Active", "Inactive", or "Deleted" state. The usual state is "A" (Active). These states may be used by policy enforcement, for example, to prevent access to items in non-Active states. They may also be used by external tools, for example, to indicate whether an object's data should be indexed or not.

Datastreams

- To create a proper section for Datastreams in the METS file, the METS:fileSec must have a single child METS:fileSec element whose ID attribute has the value "DATASTREAMS"
- Datastreams that are encoded in the METS:fileSec must follow the following pattern to establish proper version groups and datastream IDs. Each datastream has its own METS:fileGrp whose ID attribute is the official datastream ID. The recommended convention is ID="DSn" where n is a number (for example ID="DS1" or ID="DS2")."
- Within a METS:fileGrp, there can be one or more METS:file elements to represent different versions of a datastream. As of Fedora 1.2, versioning of data objects is supported. The METS:file element for the datastream must have an ID attribute that represent the version number relative to the datastream ID set in the METS:fileGrp. The recommended convention is ID="DSn.v" where n is the number of the datastream and v is the version number (for example ID=DS1.0 or ID=DS1.1).
- The METS:file element for a datastream must have a MIMETYPE.
- The METS:file element for a datastream must have an OWNERID attribute. In Fedora, the OWNERID attribute is used to encode the Datastream Control Group. The following are valid values:
 - "M" - Managed Content. This tells the repository to store the datastream's content byte stream inside the repository. When the METS:file contains "M" on the OWNERID, the repository will resolve the URL associated with the METS:file element and pull the

content into the repository for permanent storage. Fedora will establish its own local identifier for retrieving the content, and disregard the original URL that came in on the METS submission package.

- "E" - External Referenced Content. This tells the repository to store the URL for the datastream content, not the content byte stream itself. For this type of datastream, Fedora does not actually manage or have custodianship of the content, but it manages the link to the content and some basic metadata about it.
- "R" - Redirected Content. Like "E" this tells the repository to store the URL for the datastream content, not the content byte stream itself. More importantly, it tells the repository not to mediate or proxy this content at runtime. Instead, the repository will redirect to the URL at run time. This is desirable when a datastream points to a streaming media source, or to a complex web page where some components are lost during proxying.

Inline XML Datastreams

- Datastreams can also be encoded in the METS:dmdSecFedora and METS:amdSec. These are considered "inline XML datastreams" in Fedora. The METS:dmdSecFedora and METS:amdSec elements act as datastream version group containers just like the METS:fileGrp acts for regular datastreams. Within these elements, the METS "metadata section" elements (i.e., METS:techMD, METS:rightsMD, etc.) are used for the specific version instances of the inline metadata datastreams, just like the METS:file acts for regular datastreams. The datastream IDs work the same way, where the ID attribute on the container element acts as the datastream ID, and the ID on the metadata section element acts as the datastream version ID.
- Do not use the schemaLocation attribute in the root element of inline XML datastreams (within METS:mdWrap element).

Dublin Core Record Datastream

- A Dublin Core (DC) record is optional in the Fedora object submission package. If one is not provided the repository will automatically create a minimal DC record in the object by using the LABEL (on METS root) as the DC title element. It will also use the object PID as the DC identifier element.
- If a DC record is provided in the METS submission package it should be encoded within a METS:dmdSecFedora. The dmdSecFedora element will act as the datastream version group container. It MUST have an ID attribute whose value is "DC" to be recognized by Fedora!
- Within the METS:dmdSecFedora, there must be one METS:descMD element. This element is part of the Fedora extension of METS 1.1 and is used to encode a specific version of the DC datastream within the version group container. The ID attribute on the METS:descMD element MUST have the value "DC1.0" to be recognized by Fedora.
- The actual DC metadata should be encoded using the Open Archives Initiative (OAI) Dublin Core schema.

METS Ingest Example

Sample METS Ingest Encoding

```
<?xml version="1.0" encoding="UTF-8"?>
<METS:mets EXT_VERSION="1.1" LABEL="METS 1.1 Reference Example" OBJID="demo:999"
  PROFILE="TEST_IMAGE" TYPE="FedoraObject"
  xmlns:METS="http://www.loc.gov/METS/"
  xmlns:audit="info:fedora/fedora-system:def/audit#"
  xmlns:foxml="info:fedora/fedora-system:def/foxml#"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.loc.gov/METS/
http://www.fedora.info/definitions/1/0/mets-fedora-ext1-1.xsd">
  <METS:metsHdr RECORDSTATUS="A" />
  <METS:amdSec ID="DC" STATUS="A">
    <METS:techMD ID="DC.0">
      <METS:mdWrap LABEL="Dublin Core Record for this object"
        MDTYPE="OTHER" FORMAT_URI="http://www.openarchives.org/OAI/2.0/oai_dc/"
        MIMETYPE="text/xml" OTHERMDTYPE="UNSPECIFIED">
        <METS:xmlData>
          <oai_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/">
            <dc:title>METS 1.1 Reference Object</dc:title>
            <dc:creator>Sandy Payette</dc:creator>
            <dc:subject>Fedora documentation</dc:subject>
            <dc:description>METS 1.1 showing how a digital object is encoded in METS Fedora
Extension 1.1</dc:description>
            <dc:publisher>Cornell CIS</dc:publisher>
            <dc:identifier>demo:999</dc:identifier>
          </oai_dc:dc>
        </METS:xmlData>
      </METS:mdWrap>
    </METS:techMD>
  </METS:amdSec>
</METS:mets>
```

```

</METS:amdSec>
<METS:amdSec ID="RELS-EXT" STATUS="A">
  <METS:techMD ID="RELS-EXT.0">
    <METS:mdWrap LABEL="RDF Statements about this object" MDTYPE="OTHER"
      FORMAT_URI="info:fedora/fedora-system:FedoraRELSExt-1.0"
      MIMETYPE="application/rdf+xml" OTHERMDTYPE="UNSPECIFIED">
      <METS:xmlData>
        <rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:fedora="info:fedora/fedora-system:def/relations-external#"
          xmlns:myns="http://www.nsdsl.org/ontologies/relationships#"
          xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
          xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
          <rdf:Description rdf:about="info:fedora/demo:999">
            <fedora:isMemberOfCollection rdf:resource="info:fedora/test:collection1"/>
            <fedora:isMemberOfCollection rdf:resource="info:fedora/test:collection2"/>
            <myns:isPartOf rdf:resource="info:fedora/mystuff:100"/>
          </rdf:Description>
        </rdf:RDF>
      </METS:xmlData>
    </METS:mdWrap>
  </METS:techMD>
</METS:amdSec>
<METS:amdSec ID="UVATECH" STATUS="A">
  <METS:techMD ID="UVATECH.0">
    <METS:mdWrap LABEL="UVA Technical Metadata Record"
      MDTYPE="OTHER" MIMETYPE="text/xml" OTHERMDTYPE="UNSPECIFIED">
    <METS:xmlData>
      <uvalibadmin:admin xmlns:uvalibadmin="http://virginia.lib.edu/uvalibadmin:tech">
        <uvalibadmin:technical>
          <uvalibadmin:format>image/jpeg</uvalibadmin:format>
          <uvalibadmin:compression>LZW</uvalibadmin:compression>
          <uvalibadmin:bitDepth BITS="" />
          <uvalibadmin:colorSpace/>
          <uvalibadmin:colorProfile CPFILE="" CPLOCAT="" />
          <uvalibadmin:resolution>600</uvalibadmin:resolution>
        </uvalibadmin:technical>
      </uvalibadmin:admin>
    </METS:xmlData>
  </METS:mdWrap>
</METS:techMD>
</METS:amdSec>
<METS:fileSec>
  <METS:fileGrp ID="DATASTREAMS">
    <METS:fileGrp ID="IMAGE" STATUS="A">
      <METS:file ID="IMAGE.0" MIMETYPE="image/x-mrsid-image" OWNERID="E">
        <METS:FLocat LOCTYPE="URL"
          xlink:href="http://iris.lib.virginia.edu/mrsid/mrsid_images/iva/archerp01.sid"
          xlink:title="Image of Pavilion III, University of Virginia"/>
      </METS:file>
    </METS:fileGrp>
    <METS:fileGrp ID="DRAWING-ICON" STATUS="A">
      <METS:file ID="DRAWING-ICON.0" MIMETYPE="image/jpeg" OWNERID="M">
        <METS:FLocat LOCTYPE="URL"
          xlink:href="http://icarus.lib.virginia.edu/images/iva/archerd05small.jpg"
          xlink:title="Architectural Drawing Pavilion III"/>
      </METS:file>
    </METS:fileGrp>
    <METS:fileGrp ID="DRAWING-BETTER" STATUS="A">
      <METS:file ID="DRAWING-BETTER.0" MIMETYPE="image/jpeg" OWNERID="M">
        <METS:FLocat LOCTYPE="URL"
          xlink:href="http://icarus.lib.virginia.edu/images/iva/archerd05medium1.jpg"
          xlink:title="Architectural Drawing Pavilion III (med res)"/>
      </METS:file>
    </METS:fileGrp>
    <METS:fileGrp ID="DRAWING_BEST" STATUS="A">
      <METS:file ID="DRAWING_BEST.0" MIMETYPE="image/jpeg" OWNERID="R">
        <METS:FLocat LOCTYPE="URL"
          xlink:href="http://icarus.lib.virginia.edu/images/iva/archerd05high.jpg"
          xlink:title="Architectural Drawing Pavilion III (veryhigh res)"/>
      </METS:file>
    </METS:fileGrp>
  </METS:fileGrp>

```

```
</METS:file>  
</METS:fileGrp>
```

```
</METS:fileGrp>
</METS:fileSec>
</METS:mets>
```

Ingest and Export

Table of Contents

- [Table of Contents](#)
- [A Brief Note on Ingest/Export XML Formats](#)
- [Ingest and Export via the Fedora Web Administrator](#)
- [Ingest and Export via the Fedora Administrator](#)
- [Ingest an Object via Command Line](#)
- [Export an Object via Command Line](#)
- [Ingest/Export via Your Own SOAP Client](#)
- [Related Information](#)

A Brief Note on Ingest/Export XML Formats

Fedora digital objects can be encoded in several XML formats for ingest and export. Those formats are FOXML 1.1, FOXML 1.0, METS 1.1, METS 1.0, ATOM 1.1, and ATOM ZIP 1.1.

For encoding ingest files in FOXML 1.1, please refer to the [FOXML 1.1 XML schema](#) and the [Introduction to FOXML](#) guide in the Fedora System Documentation. Also look at the [FOXML ingest example file](#) for a model of a typical ingest file using FOXML.

FOXML 1.0 is included for backwards compatibility. This facilitates ingesting any existing objects you may have in FOXML 1.0 format or exporting objects in FOXML 1.0 for ingest into a legacy repository. It is recommended that if you are creating new objects using FOXML, that you use the FOXML 1.1.

For encoding ingest files in METS 1.1, please refer to the [METS XML schema](#) (Fedora extension) and also [rules for encoding Fedora objects in METS](#).

For encoding ingest files in ATOM 1.1 or ATOM ZIP 1.1, please refer to [Fedora Atom](#) documentation.

Ingest and Export via the Fedora Web Administrator

The Fedora Web Administrator provides options to ingest and export single objects from the repository using a web-based user interface. The Fedora Web Administrator uses the Fedora REST API to perform all of its functions. Please refer to the [Fedora Web Administrator](#) documentation for more information.

Ingest and Export via the Fedora Administrator

The Fedora Administrator client provides a graphical user interface for ingesting and exporting from the repository. Behind the scenes, the Administrator uses Fedora API-M, and the appropriate SOAP calls are made to the repository to accomplish the ingest. Objects are ingested as XML files. The Fedora Administrator allows for ingesting single files and file sets, both from the local file system and from other repositories. Please refer to the [Fedora Administrator](#) documentation for more information.

Ingest an Object via Command Line

The Fedora Administrator client provides a command line utility for ingesting objects into a Fedora repository. Behind the scenes, Administrator uses Fedora API-M, and the appropriate SOAP calls are made to the repository to accomplish the ingest. Objects are ingested as XML files. Please see the the [fedora-ingest](#) utility in the Fedora Command Line Utility Guide.

Export an Object via Command Line

The Fedora Administrator client provides a command line utility for exporting objects from a Fedora repository. Behind the scenes, Administrator uses Fedora API-M, and the appropriate SOAP calls are made the the repository to accomplish the export. Objects are export as XML files. Please see the the [fedora-export](#) utility in the Fedora Command Line Utility Guide.

Ingest/Export via Your Own SOAP Client

Of course, you can write your own client to perform ingest and export operations on a Fedora repository. To do this, you must familiarize yourself with the operation syntax as expressed in the WSDL for [API-M](#).

Related Information

- [FOXML Ingest Example](#)

Ingest with the file URI scheme

Ingest using the file URI scheme

Since Fedora 3.3 it is possible to reference managed and externally-managed content (type "M" and "E") with a `file:` URI within the digital object for the ingest. In order to enable this functionality the following changes are necessary:

First edit the predefined XACML policy, uncomment the relevant rule in the preinstalled policy file `deny-unallowed-file-resolution.xml` and adapt the regex to your needs (or write your own policy). Optionally bind the rule to a specific user. Make sure the RuleId is unique.

```
...
<Rule RuleId="1" Effect="Permit">
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:regexp-string-match">
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">{^}file:/allowed/. *$</AttributeValue>
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
<ResourceAttributeDesignator AttributeId="urn:fedora:names:fedora:2.1:resource:datastream:fileUri"
DataType="http://www.w3.org/2001/XMLSchema#string"/>
</Apply>
</Condition>
</Rule>
...
```

Then create your digital object FOXML and use the `file:` URI where necessary.

```
...
<foxml:datastream CONTROL_GROUP="E" ID="MEDIUM_SIZE" STATE="A" VERSIONABLE="true">
<foxml:datastreamVersion CREATED="2008-07-02T05:09:42.937Z"
ID="MediumSize.jpg.0" LABEL="Medium-size image" MIMETYPE="image/jpeg">
<foxml:contentLocation
REF="file:///path/to/files/image.jpeg" TYPE="URL"/>
</foxml:datastreamVersion>
</foxml:datastream>
...
```



file URI format

The provided `file:` URIs must not have an authority component or the authority component must be empty. This means that URIs with the `file:` scheme must either have one slash (no authority component) after the scheme:

`file:/data/image.jpeg`

or have three or more slashes (an empty authority component) after the scheme:

`file:///data/image.jpeg`

If you use this form the regex in the policy that matches the "one slash form" will nonetheless match because internally Fedora translates all `file:` URIs into the one slash form.

As a result `{^}file:/data/. *$` will match both forms above.

The use of two slashes after the scheme is not allowed and will result in an error because it defines an authority component.



File URIs and externally-managed datastreams

The default policy `deny-unallowed-file-resolution` only allows authenticated users to retrieve files from the allowed file paths. For managed content (type "M"), this restriction only applies at datastream creation; once the datastream is created, it is available via the API-A methods to any user allowed by your policies (by default, API-A methods are unrestricted). However, externally-managed content is protected by the `deny-unallowed-file-resolution` policy at the moment the datastream is created, and thereafter every time the datastream is accessed via either API-A and API-M methods; please be aware that this has the effect of making externally-managed content with file URIs available only to authenticated users, such as `fedoraAdmin`.

**Note:**

Please bear in mind that the activation of `file:` URIs for managed content exposes your filesystem to the ingest process and as such could be abused by inserting URIs to files that are not intended for ingestion. While Fedora sanitizes the given URI and denies URIs such as `file:///data/..etc/passwd`, Make sure that you:

- only expose directories without symlinks
- only expose directories that don't contain any sensitive information, like access to configuration files, password files, user home directories, etc.
- deny file URIs as soon as the ingest is finished

Portable Fedora Objects

Fedora objects that are intended for public distribution or migration must be careful to assure portability so that they can be ingested by a foreign repository and expected to work. Currently, the primary concern is with links or other references to resources assumed to exist locally within the fedora server.

Motivation

Links to resources within the fedora web application typically take the form `http://HOST:PORT/APP/PATH`, where `HOST` is the host on which Fedora runs, `PORT` is the port on which the server runs, and `APP` is the name of the a web application, and `PATH` is the path, and `PATH` is the rest of the path.

As an example, consider a url invoking a dissemination. Using default hosts, ports, and context names, such url might look like:

```
http://localhost:8080/fedora/get/demo:SmileyStuff/demo:Collection/list
```

In a different repository with custom host, port, and context names, the same dissemination on the same objects might look like:

```
http://example.org:9090/prod_fedora_32/get/demo:SmileyStuff/demo:Collection/list
```

Considering both URLs, we notice that the `PATH` is the only part of the url that hasn't changed.

Likewise, there is often a need to link to applications that are not in fedora, but *are* in the same servlet container. An obvious example of this is the saxon servlet, which is distributed with Fedora:

```
http://localhost:8080/saxon/SaxonServlet?...  
http://example.org:9090/saxon/SaxonServlet?...
```

In this situation, both `APP` and `PATH` have remained the same.

In fedora objects, there are sometimes needs to link to a dissemination of another object, or invoke a servlet installed alongside Fedora such as the 'saxon' servlet. Here are some examples:

In a datastream (based on `demo:SmileyStuff` demo object)

```
<foxml:datastream CONTROL_GROUP="E" ID="LIST" STATE="A" VERSIONABLE="true">  
  <foxml:datastreamVersion ID="LIST.0"  
    LABEL="Result of list dissemination" MIMETYPE="text/xml">  
    <foxml:contentLocation  
      REF="http://example.org:9090/prod_fedora_32/get/demo:SmileyStuff/demo:Collection/list"  
    TYPE="URL"/>  
  </foxml:datastreamVersion>  
</foxml:datastream>
```

In the WSDL of a Service Deployment (SDep) object (based on `demo:CollectionImpl`):

```

<wsdl:service name="ImageCollection">
  <wsdl:port binding="this:ImageCollection_http" name="ImageCollection_port">
    <http:address location="http://example.org/9090/" />
  </wsdl:port>
</wsdl:service>
<wsdl:binding name="ImageCollection_http" type="this:ImageCollectionPortType">
  <http:binding verb="GET" />
  <wsdl:operation name="view">
    <http:operation
location="/saxon/SaxonServlet?source=(LIST)&style=(XSLT)&clear-stylesheet-cache=(CLEAR_CACHE)" />
  ...
  <wsdl:operation name="list">
    <http:operation
location="/prod_fedora_32/riresearch?type=(TYPE)&lang=(LANG)&format=(FORMAT)&query=(QUERY)" />
  ...

```

If fedora objects are truly written this way and ingested into a repository with a different hostname, port, or fedora app context, they will fail since the links will be broken in that environment

Solution

As a solution, Fedora uses a form of text substitution that allows local links to be represented in a portable way. Two variables are defined:

```

http://local.fedora.server/ = http://HOST:PORT/
http://local.fedora.server/fedora/ = http://HOST:PORT/FEDORA_APP/

```

This substitution applies only to URLs for external reference datastreams (control groups R and E), and to the content of WSDL datastreams in SDep objects. At present, it does NOT occur in the content of datastreams (WSDL being the exception).

Let us re-write the FOXML examples above into a portable form.
In a datastream (based on demo:SmileyStuff demo object)

```

<foxml:datastream CONTROL_GROUP="E" ID="LIST" STATE="A" VERSIONABLE="true">
  <foxml:datastreamVersion ID="LIST.0"
    LABEL="Result of list dissemination" MIMETYPE="text/xml">
    <foxml:contentLocation
      REF="http://local.fedora.server/fedora/get/demo:SmileyStuff/demo:Collection/list" TYPE="URL" />
    </foxml:datastreamVersion>
  </foxml:datastream>

```

At runtime, this will be interpreted as real url

http://example.org:9090/prod_fedora_32/get/demo:SmileyStuff/demo:Collection/list or
<http://localhost:8080/fedora/get/demo:SmileyStuff/demo:Collection/list>, etc depending on the local Fedora configuration.

As far as the WSDL from (based on demo:CollectionImpl), we re-write it as follows:

```

<wsdl:service name="ImageCollection">
  <wsdl:port binding="this:ImageCollection_http" name="ImageCollection_port">
    <http:address location="LOCAL" />
  </wsdl:port>
</wsdl:service>
<wsdl:binding name="ImageCollection_http" type="this:ImageCollectionPortType">
  <http:binding verb="GET" />
  <wsdl:operation name="view">
    <http:operation
location="http://local.fedora.server/saxon/SaxonServlet?source=(LIST)&style=(XSLT)&clear-stylesheet-cache=(CLEAR_CACHE)" />
  ...
  <wsdl:operation name="list">
    <http:operation
location="http://local.fedora.server/fedora/riresearch?type=(TYPE)&lang=(LANG)&format=(FORMAT)&query=(QUERY)" />
  ...

```

Notice, we made a minor structural change in order to allow the entire URL to be written in a contiguous fashion.

Use and Practice

- By default, Fedora will store objects internally in their portable form. It is only on runtime access that the URLs are translated to concrete values. Thus, changing the port or app server context of an already-installed Fedora instance with loaded objects should work fine
- Objects can be exported in portable form from an existing repository only when exported in the "Migrate" context. See [ingest & export](#) documentation for more details

Fedora Repository

[Checksums](#)

[Command-Line Utilities](#)

[Messaging](#)

[Replication and Mirroring](#)

[Resource Index](#)

[Security](#)

[Service Framework](#)

[Versioning](#)

[Web Service Interfaces](#)

Checksums

Introduction

Fedora provides the capability of computing and storing checksums for Datastreams in a digital object, and later using that checksum to verify that the contents of that object has not been changed. This Datastream checksumming was added to Fedora to aid those in preservation and content security. Using this capability, Fedora repositories can compute a checksum for each Datastream of a digital object, and can later use this checksum to conclusively determine whether the contents of the Datastream has been changed, either through a minor hardware failure (such as a bad disk sector) changing the data stored in the low-level store, or through a user changing the contents of a file in the low-level store either accidentally or maliciously. This is achieved by the Fedora repository computing a checksum for the content whenever the content is added or modified via Fedora API-M functions, and allowing that stored checksum to be compared to one computed for the currently stored version of that content. Note that since valid changes made to Datastreams via the Fedora API-M functions cause the stored checksum to be recomputed, the checksumming feature is only designed to detect changes to the Datastreams of a digital object *outside* of Fedora; changes made to Datastreams within Fedora (via API-M functions) can be tracked through content versioning and through the audit trail records.

Enabling Automatic Checksumming

Since computing checksums for every Datastream when a Datastream is initially added and whenever a Datastream is modified via a Fedora API-M call will cause all such operations to run more slowly, automatic checksumming of Datastreams is *disabled* by default. To enable automatic checksumming, the Fedora Administrator must edit the `fedora.fcfg` file by finding the line in that file specifying: `<param name="autoChecksum" value="false">` and changing the value to "true". The configuration file entry immediately following the "autoChecksum" entry: `<param name="checksumAlgorithm" value="MD5">` specifies which checksumming algorithm is to be used for the automatically generated checksums. The default algorithm is "MD5" and the other valid values for this entry are: "SHA-1" "SHA-256" "SHA-384" and "SHA-512". Entering any other value for the "checksumAlgorithm" parameter will effectively disable automatic checksumming and will generate annoying warning messages.

How Automatic Checksumming Works

When automatic checksumming is enabled, whenever a object is ingested into Fedora, as each Datastream is processed, all of the bytes comprising the content of the Datastream are passed to the appropriate checksumming algorithm. This algorithm will compute and return a digital signature for the content of the Datastream. These checksumming algorithms are designed such that any minor change to the content will produce a wildly different result for the computed checksum. These computed Datastream checksums will then be stored in the XML

representation of the digital object. Additionally, whenever a new Datastream is added to an existing object (via `addDatastream`), and whenever an existing Datastream is modified (via `modifyDatastreamByValue` or `modifyDatastreamByReference`) a new checksum will be computed and stored in the object.

Subsequently, someone interested in verifying that the content of a Datastream has been neither damaged nor tampered with will be able to invoke the new API-M function `compareDatastreamChecksum`. This new function will take the checksum string stored for the specified Datastream and compare it with a newly-computed checksum using the same checksum algorithm as was originally used. If the checksums match, the new API-M function will return a string containing the checksum value. However if the checksums do not match, the function will return a string indicating the error.

Overriding Automatic Checksumming

In some circumstances a user of Fedora will encounter a situation where the checksumming configuration specified for the repository as a whole is not suitable for one or more Datastreams of a given object. Perhaps automatic checksumming is disabled, but there is some Datastream for which you want to have a checksum computed, or, conversely, perhaps automatic checksumming is enabled and some object has a Datastream for which checksumming doesn't make sense (i.e. either a Datastream with dynamic content that changes over time or a truly enormous Datastream for which the checksumming operation would be too time consuming).

It is possible to override the checksumming operation that will be performed and stored for a given Datastream via new parameters that have been added to the API-M functions `addDatastream`, `modifyDatastreamByValue` and `modifyDatastreamByReference`. These functions each have two new parameters, `checksumType` and `checksum`. If a value is specified for the parameter `checksumType` for any of these three functions that is the algorithm that will be used for computing the checksum for that particular Datastream, rather than the global default checksum algorithm specified in the `fedora.fcfg` file.

The valid values for the `checksumType` parameter for these three functions are: "MD5" "SHA-1" "SHA-256" "SHA-384" "SHA-512" as above, but also "DISABLED" which will turn off checksumming for that particular Datastream. Additionally for the two `modifyDatastream` functions the value null specifies that the checksum algorithm in force before the modify operation should continue to be used, whereas the value "DEFAULT" specifies that the checksumming algorithm for that particular Datastream should be changed back to whatever default checksum algorithm has been specified in the `fedora.fcfg` file.

Another way to override the default checksumming mechanism is via new attributes that have been added to the FOXML and METS specs for ingesting digital objects. The FOXML and METS specifications now allow the checksum algorithm to be used for each Datastream to be specified as attributes on one of the elements defining that Datastream. When a checksum algorithm is specified in the XML for a Datastream of a digital object, this value will be used to compute the checksum for that Datastream rather than the default algorithm specified in the `fedora.fcfg` file. Note that the syntax for thusly specifying a checksum algorithm for a Datastream in a digital object is different for FOXML and for METS, the specific syntax to use can be found in the schemas for those XML formats.

Verifying Datastream Content

Once checksums are computed and stored for a given Datastream it is possible to verify that the contents of that Datastream has not been changed in any way since the checksum was initially computed. To perform this verification a user simply invokes the new Fedora API function `compareDatastreamChecksum`, passing in the object id and Datastream id of the Datastream to be verified (plus an optional date string if a version of the Datastream other than the most recent one is to be verified). The API function will read in the digital content of the Datastream, and compute a checksum using the same checksum algorithm stored with the Datastream, and compare the resulting value with the one it previously computed and stored in the digital object. If the two checksums are identical, the function will return the value of the checksum (which could then be stored externally, if desired, as an added measure of security). If the newly computed checksum doesn't match the stored one, the API function will return a message indicating this checksumming failure. The action to take when this situation occurs is left to the user.

Additional Datastream Verification

In some circumstances a user may want further assurances that the content of a Datastream has not been unintentionally changed, perhaps through a faulty network connection or through a "man-in-the-middle" data modification. To provide this capability, there is another new parameter that can be passed in for the API-M functions `addDatastream`, `modifyDatastreamByValue` and `modifyDatastreamByReference`, named `checksum`. If a value is specified for this parameter, rather than leaving it null it is interpreted as a request to compute the checksum using the provided `checksumType`, and then compare it with the checksum that was passed in. If the checksums do not match, Fedora will assume that the data that it read for the content of the new or modified Datastream somehow was changed or damaged in transmission and the API-M function will fail and the repository will be rolled back to the state it was in prior to the call. N.B. for inline XML Datastreams, the content is normalized internally during the checksum computation process, which will make devising and passing-in the correct checksum to ensure the integrity of the passed-in content will be somewhat difficult.

Fedora Administrator

The Datastream display panels in the Fedora Administrator display the checksum algorithm and computed checksum. For the current version of a Datastream, the algorithm to be used can be changed via a drop-down list. For previous versions of a Datastream these values are displayed, but are not editable.

Command-Line Utilities

Introduction

The Fedora server distribution comes with several useful command-line utilities. A description and usage instructions for each follows.

The scripts are located in `FEDORA_HOME/server/bin/`. In Windows, these commands resolve to batch files (.bat); in Unix, they resolve to shell scripts (.sh).

Note: There are also client [command-line utilities](#) which perform object ingest and export as well as several other functions.

This guide assumes you have correctly installed the Fedora server distribution as per the install guide, including having set up your `PATH` and `FEDORA_HOME` appropriately.



Information

Currently, if you are running Fedora with a servlet container other than Tomcat, these scripts will need to be manually modified for your environment to pick up the paths to the Fedora classes and required libraries from a location other than `CATALINA_HOME`.

fedora-rebuild

fedora-rebuild

Reconstitutes Fedora's indexes (the SQL database and/or Resource Index) from the FOXML and datastream files on disk.

This is an interactive utility that should be run only when the server is offline. Depending on the size of your repository, this may take minutes (thousands of objects) or hours (millions of objects) to complete.

It is useful in a variety of situations:

- *Upgrading* from a previous version of Fedora when the SQL database or Resource Index changed significantly between releases.
- *Migrating* from one SQL database product to another in an existing Fedora installation. This can be done at any time by
 1. Modifying your `fedora.fcfg` file to point to a properly-configured `<datastore..>` (see `fedora.fcfg` for examples)
 2. Copying the appropriate JDBC jar file into the Fedora webapp's `WEB-INF/lib` directory.
 3. Running a rebuild of the SQL database
- *Recovering* from inconsistencies and/or corruption of the indexes.

When you run this utility, a text menu will appear, allowing you to specify whether you need to rebuild the SQL database or the Resource Index.

To Run a Rebuild:

1. Stop the Fedora server (if using Tomcat, this can be done with the `shutdown.bat` or `shutdown.sh` command)
2. Run `fedora-rebuild.bat` or `fedora-rebuild.sh`
3. Select which index you want to rebuild and confirm your choice when prompted.
4. Repeat steps 2-3 to rebuild the other index, if needed.
5. Restart the Fedora server (if using Tomcat, this can be done with the `startup.bat` or `startup.sh` command)



Information

When running a SQL rebuild using MySQL with Java 1.5, it may fail with a `java.lang.UnsupportedClassVersionError`. This can occur if the MySQL JDBC driver you're using is a newer version. To resolve, simply run the builder with Java 1.6 (ensuring your `JAVA_HOME` environment variable is set correctly), or use an older MySQL JDBC driver.

fedora-reload-policies

fedora-reload-policies [http|https] [username] [password]

Where:

- **http|https** - Indicates which protocol to use to send the "reload policies" signal to the running Fedora server.
- **username** - An administrative Fedora user with permission to reload policies.
- **password** - Password for the administrative user.

Causes any new or changed repository-wide policies to take effect immediately on the running Fedora server.

As described in the document, [Fedora Authorization with XACML Policy Enforcement](#), Fedora can be configured to enforce a variety of access policies. Many of these XACML policies are applied for all actions and access attempts performed on the repository as a whole. These "repository-wide" XACML policies are automatically loaded at the time the Fedora server is started. If the Fedora server administrator needs to change one or more of these repository-wide policies, this command can be used to tell the running Fedora server to reload the policies. The alternative to using this command is to stop the Fedora server and restart it.

validate-policy

`validate-policy [policyFilename]`

Where:

- **policyFilename** - Name of XACML file containing the new or modified policy

Schema-validates a XACML policy file.

If the Fedora server administrator creates or modifies an existing repository-wide XACML policy, the new policy **should** be run through this program to ensure that it is well-formed before attempting to install it in the Fedora server. Validating a policy in this way will ensure that it is well-formed XML and can follow the XACML XML schema.

fedora-modify-control-group



Ensure DC, RELS-EXT and RELS-INT are versionable if using Managed Content

Due to an outstanding bug [FCREPO-849](#), if you use Managed Content for DC, RELS-EXT or RELS-INT then please make sure these datastreams are versionable (the default setting for versionable is "true", so if you haven't specified this datastream property then you are safe). Particularly take care if you are migrating an existing datastream whose VERSIONABLE property is set to "false", as this will cause problems. You will need to ensure the VERSIONABLE property is "true" before migrating.

The `fedora-modify-control-group` command line utility enables you to modify the control group of existing datastreams. Currently only modifying inline XML datastreams ("X") to managed content ("M") is supported. The utility can be used to modify DC, RELS-EXT and RELS-INT datastreams to managed content with the introduction of support for managed content for these datastreams in Fedora 3.4.

`fedora-modify-control-group` [protocol] [user] [password] [pid] [dsid] [controlGroup] [addXMLHeader] [reformat] [setMIMETypeCharset]

Where:

- **protocol** - the protocol to communicate with Fedora server, either http or https.
- **user** - the Fedora administrator username (e.g., `fedoraAdmin`).
- **password** - the password for the Fedora administrator user.
- **pid** - either
 - a single pid (eg `demo:123`)
 - a comma-separated list of pids (eg `demo:123,demo:124`)
 - the name of a file containing a list of pids (eg `file:///path/to/pidfile.xml`). The file may either be a simple text file containing a list of pids, or an XML file specifying pids as `<pid>demo:123</pid>` elements. The XML output of Fedora's basic search (the `findObjects` REST API method) can be used.
- **dsid** - either a single datastream identifier or a comma-separated list of datastream identifiers (eg DC or DC,RELS-EXT)
- **controlGroup** - the control group to set on the datastream. Only "M" is currently supported
- **addXMLHeader** - optional. If true, an XML declaration specifying the XML version and a character encoding of UTF-8 will be added at the start of the datastream.
- **reformat** - optional. If true, the XML will be reformatted with line breaks and indents.
- **setMIMETypeCharset** - optional. If true, a charset declaration of UTF-8 will be added to the datastream's MIMEType property if one is present

If a single pid and datastream are specified, an error will be generated if the datastream is not found. If a list of datastreams is specified, the datastreams will be upgraded only if found in the object, and no error will be given if the datastream is not found in a particular object. The output will give a full list of objects, datastreams and datastream versions migrated to the new control group.

Messaging

Table of Contents

1. [Introduction](#)
2. [Messaging in Fedora](#)
3. [Configuring Messaging](#)
4. [Messaging Client](#)
5. [Messages](#)
6. [Configuring Messaging with ActiveMQ for Higher Availability](#)

Introduction

Messaging is a communication mechanism used to send and receive information in a manner which allows the senders and receivers to be unaware of the activities or status of the other parties involved in the exchange. This loose coupling is achieved through the use of an intermediary queue or topic managed by a messaging provider. The messaging provider is responsible for delivering messages sent by message producers to message consumers. Messaging in Fedora is implemented using the [Java Messaging Service \(JMS\)](#) which is a specification used by many messaging providers that allows messages to be sent or received from a queue or topic in a generic fashion.

Messaging in Fedora

The goal of messaging in Fedora is to provide updates about the activities of the repository as they occur. This allows external applications to monitor and perform actions based on those activities. In order to provide the capability for multiple independent clients to receive identical update messages simultaneously, an asynchronous publish-and-subscribe model was chosen as the default for Fedora's messaging capability. The messages sent using this model indicate when functions of API-M have been exercised, thus providing information about every update made to digital objects within the repository.

Fedora uses [Apache ActiveMQ](#) as its default messaging provider. While the use of JMS suggests that any messaging provider supporting JMS can replace ActiveMQ, no other providers have been tested. If you do use Fedora with another JMS-compliant messaging provider, please [let us know](#) your results.

Configuring Messaging

Messaging in Fedora is configured primarily through the messaging module within the `fedora.fcfg` file. The following parameters, specified as part of the messaging module, are required in order to establish a JMS Connection:

- **enabled**
 - Default: false
 - Determines whether the messaging module should be initialized in order to send messages. If you want messages to be sent, this must be set to true.
- **java.naming.factory.initial**
 - Default: `org.apache.activemq.jndi.ActiveMQInitialContextFactory`
 - Specifies the JNDI initial context with which the connection factory and destination administered objects will be looked up. As indicated by the default value, Fedora uses [ActiveMQ](#) as its default JNDI provider as well as its default messaging provider. The messaging provider and JNDI provider do not need to be the same.
- **java.naming.provider.url**
 - Default: `vm:(broker:(tcp://localhost:61616))`
 - Specifies the address at which a connection can be made to the messaging provider. Depending on the provider, this address may indicate any of several protocols and may include additional parameters. The default URL uses ActiveMQ specific syntax to allow for both internal JVM transport and transport over TCP connections via port 61616. More information on ActiveMQ broker URIs can be found on [their website](#).
 - Fedora will attempt to connect to the messaging provider at this address on startup, so make sure that your provider is running and available.
- **connection.factory.name**
 - Default: `ConnectionFactory`
 - Specifies the JNDI name of the `ConnectionFactory` object needed to create a connection to the JMS provider. ActiveMQ creates a connection factory on startup and stores it under the name `ConnectionFactory` in its included JNDI provider. If you are using a different JMS or JNDI provider, you will need to create a connection factory in JNDI and specify the name under which it is stored as the value of this parameter.

Once a connection is established to the JMS provider, Fedora needs to know where to publish messages. Two topics are currently available for this purpose:

- **fedora.apim.update** - API-M methods which cause an update to occur within the repository. This includes all ingest, add, modify, set, and purge activities.
- **fedora.apim.access** - API-M methods which access the repository but do not cause an update to occur. This includes all methods not considered updates.

The names of these topics may be changed by specifying new values for the `name` parameter of the `apimUpdateMessages` and `apimAccessMessages` datastores within the `fedora.fcfg` file. Changing the names will not alter the messages being sent but will send those messages to different destinations.

If a point-to-point messaging model is preferred, the type parameters of the datastores mentioned above can be changed to "queue", which will result in the messages being placed in a queue of the name specified in the datastore. Queues allow only a single entity to retrieve and process each message, but they do remove timing dependencies inherent with the publish-and-subscribe model (subscribers must register their interest prior to a message being published in order to receive that message.)

Messaging Client

In order to receive the messages that are being sent by Fedora, you will need to create a message consumer to listen for Fedora's notification messages. To aid in this effort, the Messaging Client was created. To build the Messaging Client, run the `messaging-client` Ant target from the source distribution. After the build completes, look in the `dist` directory for `fedora-messaging-client.zip`, which includes all of the jars necessary to use the Fedora Messaging Client. If you are using a messaging provider other than ActiveMQ, you will need to replace the `activemq-all` jar with the appropriate jars from your messaging provider of choice.

The Messaging Client was designed to provide a simple Java interface for receiving messages from Fedora. Some configuration parameters are necessary in order for the client to create a connection and listen to the appropriate topic or queue. The parameter names here are the same as those listed above for messaging, and the values should be the same as those in your `fedora.fcfg` file. The topic or queue name(s) on which to listen are also included as parameters and the value(s) should match those in the `fedora.fcfg` datastores. If you are using ActiveMQ as your JMS and JNDI providers each topic or queue will be created for you, otherwise you will need to create destination object(s) in JNDI to match the property values you specify here.

- **java.naming.factory.initial** - this String value can be found in javax.naming.Context.INITIAL_CONTEXT_FACTORY
- **java.naming.provider.url** - this String value can be found in javax.naming.Context.PROVIDER_URL
- **connection.factory.name** - this String value can be found in fedora.server.messaging.JMSManager.CONNECTION_FACTORY_NAME
- **topic.{name} OR queue.{name}** - topics are specified using the prefix "topic." followed by a topic name. Queues are specified using the prefix "queue." followed by a queue name.

The code below is a simple example of how to use the Messaging Client. The `JmsMessagingClient` constructor includes three required parameters and two optional parameters. The required parameters include the client ID, the `MessagingListener` instance, and the connection properties mentioned above. The optional parameters include a message selector and flag which determines whether durable subscribers should be used to listen over the topics listed in the properties. More information about each of the available parameters can be found in the `JmsMessagingClient` javadocs.

```
public class Example implements MessagingListener {
    MessagingClient messagingClient;
    public void start() throws MessagingException {
        Properties properties = new Properties();
        properties.setProperty(Context.INITIAL_CONTEXT_FACTORY,
            "org.apache.activemq.jndi.ActiveMQInitialContextFactory");
        properties.setProperty(Context.PROVIDER_URL, "tcp://localhost:61616");
        properties.setProperty(JMSManager.CONNECTION_FACTORY_NAME, "ConnectionFactory");
        properties.setProperty("topic.fedora", "fedora.apim.*");
        messagingClient = new JmsMessagingClient("example1", this, properties, false);
        messagingClient.start();
    }
    public void stop() throws MessagingException {
        messagingClient.stop(false);
    }
    public void onMessage(String clientId, Message message) {
        String messageText = "";
        try {
            messageText = ((TextMessage)message).getText();
        } catch (JMSEException e) {
            System.err.println("Error retrieving message text " + e.getMessage());
        }
        System.out.println("Message received: " + messageText + " from client " + clientId);
    }
}
```

A new feature in Fedora version 3.1 is the option to start the `JmsMessagingClient` asynchronously. If you are starting a messaging client only to listen for messages on a topic, there is likely no need to wait for that connection to be made before continuing with other processing. Once the connection is made, messages will be passed to your `onMessage()` method as usual. To take advantage of this, simply call `jmsMessagingClient.start(false)` rather than `messagingClient.start()` in the example above. Note that if you intend to publish messages, you will still need to wait for the connection to complete (i.e. use `messagingClient.start()`) prior to adding a message to a topic or queue.

Messages

The content of messages sent by Fedora takes the form of feed entries based on the [Atom Syndication Format](#). These messages correspond to API-M method calls, indicating the name of the method, its parameters, return value, and other information about the method. Each message will be similar to the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<entry xmlns="http://www.w3.org/2005/Atom"
    xmlns:fedora-types="http://www.fedora.info/definitions/1/0/types/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <id>urn:uuid:3773e144-1b63-4dde-8786-464243af9186</id>
  <updated>2008-04-14T22:35:13.953Z</updated>
  <author>
    <name>fedoraAdmin</name>
    <uri>http://localhost:8080/fedora</uri>
  </author>
  <title type="text">purgeObject</title>
  <category term="demo:5" scheme="fedora-types:pid" label="xsd:string"></category>
  <category term="purge message" scheme="fedora-types:logMessage" label="xsd:string"></category>
  <category term="false" scheme="fedora-types:force" label="xsd:boolean"></category>
  <summary type="text">demo:5</summary>
  <content type="text">2008-04-14T22:35:13.953Z</content>
</entry>
```

The Atom tags in each message will have the following values:

- `<id>` uniquely identifies each entry
- `<updated>` indicates the date and time at which the call occurred
- `<author>` identifies the initiation point of the API-M method call
 - `<name>` specifies the name of the user making the call
 - `<uri>` corresponds to the baseURL of the Fedora repository from which the call originated
- `<title>` specifies the method name
- Each `<category>` corresponds to a method's argument:
 - The term indicates the argument value. However, null values are indicated as "null", and non-null `xsd:base64Binary` values are indicated as "[OMITTED]".
 - The scheme indicates the argument name
 - The label indicates the argument datatype
- `<summary>` corresponds to the PID of the object operated on by the method, if applicable.
- `<content>` corresponds to the textual representation of the method's return value, noting the following:
 - Null values are represented as "null".
 - `fedora-types:ArrayOfString` values are represented as a comma-separated list, e.g. "value1, value2, value3".
 - Non-null `xsd:base64Binary` values are not returned, and only indicated as "[OMITTED]".
 - Non-null `fedora-types:DataStream` values are not returned, and only indicated as "[OMITTED]".
 - `fedora-types:RelationshipTuple` values are represented in Notation3 (N3).

Two properties are included as part of each JMS message produced by Fedora, primarily for use by message selectors. These two properties are: **methodName** and **pid**. A message selector, which can be specified using the Messaging Client, is used to limit the messages that will be delivered based on selection criteria. An example of a message selector is: "methodName LIKE 'purge%'". This selector would limit the messages received by the client to only those messages for which the method is a purge (purgeObject and purgeDataStream.)

Configuring Fedora with an external ActiveMQ broker for higher availability

By default, Fedora will shut down if it cannot contact your configured ActiveMQ broker upon startup. On the other hand, if your external broker shuts down after Fedora has successfully established a connection to it, no recovery or shutdown is performed, meaning that you could lose messages. The following steps show how to configure a simple store-and-forward ActiveMQ message broker bridge between Fedora (producer broker) and a remote ESB (message consumer broker) set up as a failover. This means that Fedora will:

1. upon startup, load the ActiveMQ embedded broker that ships with Fedora;
2. configure the embedded broker to:
 - a. forward Fedora messages to a remote broker, if the remote broker is available;
 - b. if the remote broker is not available, store the messages until the remote broker becomes available, at which point it will forward the stored messages to it
3. use the ActiveMQ "failover" transport to attempt periodic reconnections to the remote broker when it is unavailable



Best Practice

The following configuration is recommended as a **best practice** in production environments where Fedora messages are a critical part of the repository workflow.

Step 1: Install a remote broker

Install a remote broker, and make sure it can receive messages via TCP. Start it up, make sure that it runs, binds to the correct address and port. Maybe even run a few test, to verify that messages arrive and are processed. In the example configurations below, the remote broker is bound to 0.0.0.0 (all interfaces), port 61617.

Step 2: Configure Fedora

1. You will need to make a few jars available to Fedora so it can load and process the ActiveMQ configuration file. Drop the jars `xbean-spring-3.4.3.jar` and `spring-context-2.5.6.jar` files into the Fedora webapp `WEB-INF/lib` directory. Most spring applications provide these jars; if you are using [Apache Servicemix](#) as the container for your remote broker, they can be found in the `SERVICEMIX_HOME/lib` directory.
2. Create the Fedora `activemq.xml` file, put it in `FEDORA_HOME/server/config`.

```

<beans xmlns:amq="http://activemq.apache.org/schema/core">

  <!-- ActiveMQ JMS Broker configuration -->
  <amq:broker id="broker" useShutdownHook="false">

    <amq:managementContext>
      <amq:managementContext connectorPort="1093" createConnector="false"/>
    </amq:managementContext>

    <!-- Your remote broker, configured with failover -->
    <amq:networkConnectors>
      <amq:networkConnector name="fedorabridge" dynamicOnly="true"
uri="static:(failover:(tcp://0.0.0.0:61617))"/>
    </amq:networkConnectors>

    <!-- The directory where Fedora will store the ActiveMQ data -->
    <amq:persistenceAdapter>
      <amq:amqpPersistenceAdapter directory="file:./data/amq"/>
    </amq:persistenceAdapter>
  </amq:broker>

  <!-- Set this to prevent objects from being serialized when
       passed along to your embedded broker; saves some overhead processing -->
  <bean id="jmsConnectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory">
    <property name="objectMessageSerializationDefered" value="false"/>
  </bean>

</beans>

```

This piece of the config file creates the bridge to the remote broker, configured in failover mode. The *dynamicOnly* property ensures that messages are forwarded only when there is a consumer available to receive them.

```

<amq:networkConnectors>
  <amq:networkConnector name="fedorabridge" dynamicOnly="true"
uri="static:(failover:(tcp://0.0.0.0:61617))"/>
</amq:networkConnectors>

```

3. Configure Fedora to read the `activemq.xml` file upon startup, create the embedded broker. Make sure messaging is enabled in `fedora.fcfg`:

```

<module role="org.fcrepo.server.messaging.Messaging"
class="org.fcrepo.server.messaging.MessagingModule">
  <comment>Fedora's Java Messaging Service (JMS) Module</comment>
  <param name="enabled" value="true"/>
[...]
```

Also change the `java.naming.provider.url` parameter:

```

<param name="java.naming.provider.url"
value="vm://localhost?brokerConfig=xbean:file:/path/to/fedora_home/server/config/activemq.xml"/>

```

4. Restart Fedora, and start up your remote broker (or the other way around: it no longer matters). The ActiveMQ failover transport documentation contains a list of useful parameters that you can use to configure connection management to the remote broker (max reconnect attempts, period between reconnects, etc.)

Useful Documentation

Configuring an ActiveMQ store-and-forward network of brokers: <http://activemq.apache.org/how-do-distributed-queues-work.html>

- With failover: <http://bsnyderblog.blogspot.com/2010/01/how-to-use-automatic-failover-in.html>
- Failover documentation: <http://activemq.apache.org/failover-transport-reference.html>

Using the `xbean:file` URI to load the ActiveMQ broker configuration from a Spring bean config file:
<http://activemq.apache.org/broker-xbean-uri.html>

Replication and Mirroring

Introduction

Using the multicast journal transport, it is possible to achieve replication or read-only mirroring of Fedora servers via journaling. Journal entries are created on a read/write Fedora server (the leader), and sent to other Fedora servers (followers) operating in journal recover mode.

Components

To implement replication between Fedora servers, at present three components are necessary:

1. A leader configured to log journals to all follower repositories
2. Any number of following Fedora servers to read and execute the journal entries
3. An RMI journal receiver deployed for each follower that receives journal messages, and writes journal files for the followers to play back.

Leader (Master) Fedora Repository

The leader operates in full read/write mode, and creates journals for any write operations requested of it. This leader must be configured to use the [multicast journal writer](#), and all follower RMI receivers must be listed in the journal configuration in `fedora.fcfcfg`. All follower receivers must be running when the leader is started. Adding new receivers requires adding them to the server `fedora.fcfcfg`, and re-starting the leader Fedora instance. The leader configuration determines the size and age of all journal files. All transports, remote or local, produce identical journal files in response to journal events.

The multicast journal writer is able to designate any transport as 'crucial' (i.e. Fedora will no longer service write requests if it fails), and at least one transport must be designated as crucial. Common practice is to have a local file transport writing to the leader's file system that is designated as crucial, with all remote RMI transports as not crucial.

RMI Journal Receivers

RMI receivers listen for journal events from the leader, and write journal files into a directory containing journal entries. The size, duration, and name of each file is determined entirely by the leader server, since the server is responsible for sending 'open file' and 'close file' events. In the end, the journal files produced by a receiver are identical to those created locally by the leader server or any other receiver. The receiver application is completely separate from Fedora and merely writes files to a directory that follower Fedora instances read at their convenience. Just as is the case for a Fedora instance logging journals to the local filesystem, journals that are open and still being written to will be prefixed with an `_` underscore.

The RMI receiver application is distributed as a separate [download](#) in the Fedora distribution, or may be built from source by the `rmi-journal-receiver` Maven build. The receiver itself is a single executable jar file `RmiJournalReceiver.jar`. When invoked, it will expect a command-line argument which names the directory in which it deposits files, as in

```
java -jar RmiJournalReceiver.jar /path/to/journals
```

The RMI receiver will run continuously, and will print any logging error messages to the standard output console. In production, it is recommended to run this as a daemon or service, and direct the output to a log file.

Journal events are pushed from the server to the follower. As such, the machines on which the RMI receiver is deployed must have two ports available: 1099 and 1100. Firewalls on follower machines must allow connections to these ports.

Follower (Slave) Fedora Repositories

Follower repositories are simply Fedora instances that are configured to run in recover mode, and use a following journal reader. In their `fedora.fcfcfg`, the journal reader must be specified as either `fedora.server.journal.readerwriter.multifile.MultiFileJournalReader` or `fedora.server.journal.readerwriter.multifile.LockingFollowingJournalReader`. These readers continuously poll a directory for the presence of journal files, and execute journal playback whenever a non-closed journal file becomes available. The journal directory should be the same directory that the RMI journal receiver is configured to write to.

While the RMI receiver must run continuously, the follower Fedora server can be stopped and started at will. The consequence of shutting down a follower Fedora is merely that journal files will pile up in the specified directory. When started, the follower Fedora will quickly play back journals and empty the directory until caught up.

The follower Fedora server is always in read-only mode. Query and read-only API operations are available at all times. Because journal files are read only when they are closed, any data written to the journal file will not appear in the follower until its file is closed, which depends on the leader configuration. Small journal file size limits and shorter lifetimes will result in the followers being closer to the leader state at any point in time.

Also note that the follower Fedora, when playing back journals, behaves exactly as the leader server when it initially received each request. This has implications for JMS. For example, if a follower has messaging enabled, each journal operation `_will_` result in a JMS message. If the follower has a Gsearch index, it will update from the follower exactly as one for the leader instance would. Thus, it is possible to replicate entire stacks of services around each follower.

Scenarios

The replication scheme described here is useful in a number of situations:

Backups

Because data in Fedora is distributed between the filesystem and multiple databases (object registry database, resource index, etc), it is nearly impossible to obtain a completely consistent snapshot/dump of the all Fedora content from a live system. Thus, it is helpful to deploy a follower that can be shut down as necessary to allow for database and filesystem dumps to be assembled and archived - allowing the leader to run uninterrupted.

Upgrades and Maintenance

Followers may be manually turned into leaders by swapping `fedora.fcfg` configuration and re-starting. In this fashion, a leader may be taken down and replaced by one of its followers with minimal downtime (about as long as it takes for Fedora to start), freeing up the machine for upgrades or other scheduled maintenance. Once finished and caught up on journals as a follower, it may be switched back to a leader. Three servers allow for maintenance without loss of redundancy.

Load Balancing

For high-volume mostly read-only sites, followers may be used to provide data as part of a load-balancing scheme. The caveat here is that the followers will be slightly behind the leader as the journal files are written, closed, and processed. Small journal sizes and short lifetimes will mitigate this issue slightly.

Journaling

Table of Contents

1. Introduction
 - a. Goals
 - b. Repository Equivalence
 - c. Forensic Information
 - d. Status
2. Design
 - a. A Decorator on the Management Module
 - b. Modes of Operation
 - c. Extended Context Interface
 - d. The Repository Hash
 - e. Repeatability
 - f. Extensibility
3. Options
 - a. General Options
 - b. Options for Single-file Operation
 - c. Options for Multi-file Operation
 - d. Multicast Journaling
 - e. Options for Recovery Log
4. Implementation
 - a. Configuring Fedora
 - b. Journal Content Example
 - c. Recovery Log Examples
 - d. Sequence Diagrams

Introduction

The Fedora configuration now includes an optional Journaling module. This module creates Journal files which capture all calls against the Fedora Management API, or rather, all calls that have changed the state of the repository. The Journal files can be replayed against another Fedora instance with the same initial state, resulting in a repository which is a mirror of the original.

Goals

The Journaling module can increase Fedora availability in one of two ways: server recovery to the level of the most recent action, or the creation of a standby server for fast failover.

Without Journaling, a Fedora environment which fails for any reason can only be recovered to the most recent backup. Actions which have

occurred since the backup are lost. With Journaling, the environment can be reconstructed to the point of the failure. All successful actions will be recovered, and the server can then resume normal operation.

It is also possible to have two Fedora servers operating in a "leader/follower" configuration. The leading server handles requests from users, and creates Journal files to reflect those requests. The following server processes each Journal file as it is completed. If the leading server should fail, the following server will very quickly be ready to take over the handling of user requests.

Repository Equivalence

When recovering from a failure, the recovered repository must match the original in all significant ways. PIDs must match those that were initially generated. Timestamps in "create date" or "last modified" fields must reflect the date of the original API call, not the date of the recovery. Generation of PIDs must resume with the same sequence as the original server would have used.

The need for deterministic Journaling places limits on Fedora's ability to conduct simultaneous multi-threaded operations. When the Journal module is installed, it synchronizes the calls to the Management module in order to insure that those calls can be replayed in the correct sequence.

Forensic Information

The Journal files contain the information that is needed to reconstruct the repository, and some extra information as well. These additional fields are included to permit analysis of the Journal files. Tools that scan the Journal files can easily determine the user and IP address that initiated each API call, as well as the time of the call. Other information may be added to the Journal files to assist in analyzing the server operation.

Status

The Journaling module currently writes Journal files to a disk directory, and recovers from that same directory. If a "leader/follower" configuration exists with Fedora instances on different servers, those servers must somehow share the directory where the Journal files are written.

The Journaling module is designed to permit additional transport mechanisms for Journal data, and current plans call for a transport that uses JMS (Java Message Service) instead of file-based transport. No date has been set for this enhancement.

Design

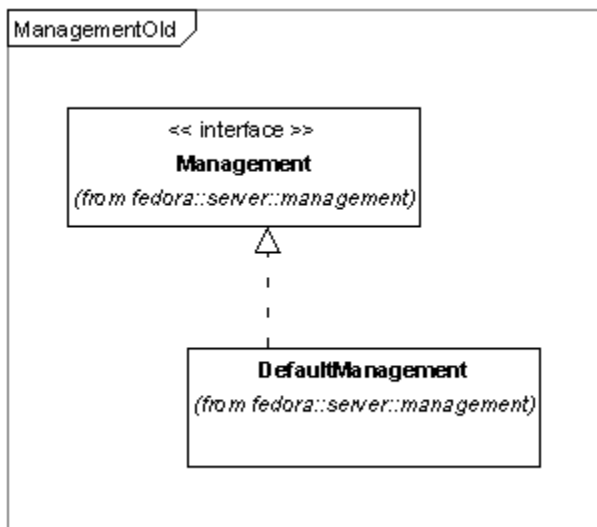
A Decorator on the Management Module

The Journaling module uses the Decorator design pattern. The Journaler class implements the Management interface, and adds functionality to each of the methods of that interface before delegating the calls to another Management instance.

The design pattern is slightly modified, since the convention in Fedora is that each module will implement its own interface. A new interface was created, called ManagementDelegate, and the Journaler requires a module that implements that interface to use as its delegate.

Here is a class diagram for the Management interface, and its implementing class, followed by an excerpt from `fedora.fcfg`, showing how the module is configured (without Journaling).

Class diagram for Management module:



Configuring the Management module within `fedora.fcfg`:

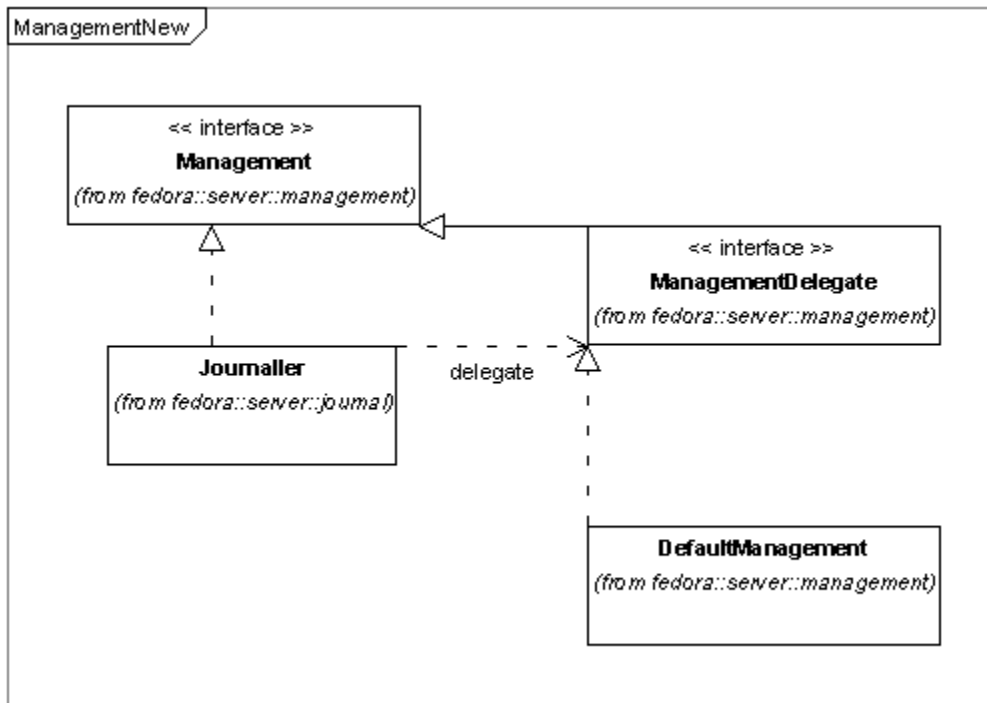
```

<module role="fedora.server.management.Management"
  class="fedora.server.management.ManagementModule">
  <!-- ManagementModule configuration params go here -->
</module>

```

The next class diagram shows the addition of the Journaler module, followed by another excerpt from `fedora.fcfg` with both Journaler and Management modules. Notice how the Journaler class implements the Management module interface, while the ManagementModule class has moved to implement ManagementDelegate.

Class diagram for the Journaler and Management modules:



Configuring the Journaler and Management modules within `fedora.fcfg`:

```

<module role="fedora.server.management.Management"
  class="fedora.server.journal.Journaler">
</module>
<module role="fedora.server.management.ManagementDelegate"
  class="fedora.server.management.ManagementModule">
  <!-- ManagementModule configuration params go here -->
</module>

```

Modes of Operation

The Journal operates in one of two modes: "normal", or "recover".

In normal mode, each call to the Management API is intercepted and passed to the ManagementDelegate. The arguments and the return value from that call are recorded in the Journal.

In recover mode, a worker thread processes the Journal files and makes appropriate calls to the ManagementDelegate. Any call from a user is blocked, returning an Exception message.

One might expect a third mode of operation, to be used when following another server. In fact, this is implemented using recover mode with a different JournalReader implementation class. This is illustrated in the section "[Configuring Fedora](#)"

Extended Context Interface

The behavior of the Management module changes slightly when it is acting as a delegate for the Journaler module. This is accomplished by adding a "recovery" namespace to the Context object that is passed to the methods of the Management module.

In normal mode, the Journaler stores information in the recovery namespace of the Context. This information is needed in addition to the arguments and return value of the API method called, in order to fully capture the state of the repository for recovery.

When recovering, the Management module will look for this additional information, and use it when executing the API method.

For example, if the Ingest method is called and the ingested FOXML does not supply a PID for the object, Fedora will generate one. However, to insure that the same PID is assigned during recovery, the Journaler records the PID, and on recovery the Management module will assign the recorded PID to the new object, instead of generating a new PID which might be different.

The Repository Hash

Each Journal file is tagged with a "repository hash" value, which indicates the internal state of the repository when the Journal file was created. In recover mode, the hash value in the file is compared to the hash value of the new repository. If the values do not match, the recovery is aborted.

The repository hash value insures that Journal files are not applied in the wrong order, and that no files are skipped during the recovery. Either of these problems will be detected by comparing the hash values.

Repeatability

A basic concept of Journaling is the repeatability of actions. If a series of actions is executed on two Fedora servers, the result must be two equivalent Fedora repositories.

The need for repeatability limits Fedora's ability to use multi-threading in API calls. Multi-threaded operation is non-deterministic by nature, but a Journal file must record a series of events must occur in a particular order, and that order must reflect the actual execution of the events. Otherwise, it would be possible to execute two actions successfully, and to have those same two actions fail on another server, because they were executed in a different order.

By careful implementation, this single-threaded sequencing can be reduced to the smallest possible interval, but it cannot be eliminated without breaking the repeatability of the Journal.

Extensibility

Some components of the Journaler module are dynamically loaded at server startup time, based on parameter values. The storage and transport mechanisms for the Journals are controlled by these components. It is fairly simple to implement an alternative file structure for the Journal files, or to use a message-based transport instead of simple flat files.

The recovery log that is generated by the Journaler is controlled by a similar set of parameters, and can be extended or replaced in the same manner.

Options

These options are recognized by the Journaler module, or by its configurable components. See the section "[Configuring Fedora](#)" for examples of how the options are used in `fedora.fcfig`.

General Options

These options are recognized by the Journaler module, and can be used with any configurable components.

- **journalMode**
Sets the operating mode of the Journaler module. Optional. Default is "normal".
 - normal
API calls are performed normally, and a Journal is written.
 - recover
The Journal is read and executed, while API calls from outside are rejected.
- **continueOnHashError**
Permits the recovery to continue, even if the repository hash does not match the value recorded in the Journal file. Optional. Applies only to recover mode. The default is "false".
 - true
Errors in the repository hash value will be recorded in the recovery log, but will not cause the recovery to terminate.
 - false
Any error in the repository hash value will cause the recovery to terminate immediately.
- **journalWriterClassname**
Specifies the configurable component that writes the Journal during normal mode operation. This class must be an extension of `fedora.server.journal.JournalWriter`. Required for normal mode. There is no default. Note that this component may require additional parameters.
 - `fedora.server.journal.readerwriter.multifile.MultiFileJournalWriter`
Writes a collection of Journal files to a specified directory on disk. Best class for most operating scenarios.
 - `fedora.server.journal.readerwriter.singlefile.SingleFileJournalWriter`
Writes a single Journal file. This writer is likely to be useful only for testing, since the Journal file will quickly become too large to handle during normal operation.
- **journalReaderClassname**
Specifies the configurable component that reads the Journal during recover mode operation. This class must be an extension of

fedora.server.journal.JournalReader. Required for recover mode. There is no default. Note that this component may require additional parameters.

- fedora.server.journal.readerwriter.multifile.MultiFileJournalReader
Reads a collection of Journal files from a specified directory on disk. As each file is processed, it is moved to an "archive" directory. When all files have been processed, recovery is complete. Good class for a recovery scenario.
- fedora.server.journal.readerwriter.multifile.LockingFollowingJournalReader
Monitors a specified directory on disk for the existence of new Journal files. As each file is found, it is processed and moved to the "archive" directory. When all files have been processed, the server will continue to poll the directory for new files. This reader enables a server to "follow" another. It also accepts "lock requests" to pause processing and allow an orderly shutdown (see "lockRequestedFilename" and "lockAcceptedFilename" options below). Best class for a following scenario.
- fedora.server.journal.readerwriter.multifile.MultiFileFollowingJournalReader
Same as LockingFollowingJournalReader, but does not recognize the "lockRequestedFilename" and "lockAcceptedFilename" options.
- fedora.server.journal.readerwriter.singlefile.SingleFileJournalReader
Reads a single Journal file. When the file has been processed, recovery is complete. This reader is likely to be useful only for testing, since the Journal file will quickly become too large to handle during normal operation.
- journalRecoveryLogClassname
Specifies the configurable component that writes the recovery log during recover mode operation. This class must be an extension of fedora.server.journal.recoverylog.JournalRecoveryLog. Required for recover mode. There is no default. Note that this component may require additional parameters.
 - fedora.server.journal.recoverylog.RenamingJournalRecoveryLog
Writes log messages to a disk file as soon as they are initiated. The path specified in the recoveryLogFilename option is used as the basis for the filename. A timestamp will be appended to the filename in order to avoid a problem with later logs overwriting earlier ones. While recovery is in progress, the filename is preceded by an underscore character. When recovery is complete, the underscore character is removed. This class is best for most operating scenarios.
 - fedora.server.journal.recoverylog.UnbufferedJournalRecoveryLog
Writes log messages to a disk file as soon as they are initiated. Writes to a file specified in the recoveryLogFilename option.
 - fedora.server.journal.recoverylog.BufferedJournalRecoveryLog
Accumulates log messages in a StringBuffer and writes them to disk only when recovery is complete. This component is useful for testing, since it gives a clear external indication of completion. It will almost certainly be too memory-intensive for normal operation.

Options for Single-file Operation

These options are recognized by the reader and writer components in the fedora.server.journal.readerwriter.singlefile package.

- journalFilename
The full path to the journal file. Required. There is no default.

Options for Multi-file Operation

These options are recognized by the reader and writer components in the fedora.server.journal.readerwriter.multifile package:

- journalDirectory
The full path to the directory where the journal files are written. Required. There is no default.
- archiveDirectory
The full path to the directory where the journal files are archived after being processed in recover mode. Required for recover mode. There is no default.
- journalFilenamePrefix
The name of each Journal file will consist of this prefix followed by a timestamp to insure uniqueness. Optional. The default is "fedoraJournal".
- journalFileSizeLimit
When a Journal file exceeds this size, it is closed. Subsequent API calls will be recorded in a new Journal file. The value must be an integer number of bytes, optionally followed by "K", "M", or "G" to indicate Kilobytes, Megabytes, or Gigabytes, respectively. Optional. The default is "5M".
- journalFileAgeLimit
When the age of a Journal file exceeds this value, it is closed. Subsequent API calls will be recorded in a new Journal file. The value must be an integer number of seconds, optionally followed by "M", "H", or "D" to indicate Minutes, Hours, or Days, respectively. Optional. The default is "1D". Options for following readers only:
- followPollingInterval
The length of time to wait between checking for new journal files. If the journal directory is empty, the JournalReader will wait this number of seconds before checking again. The value must be an integer number of seconds, optionally followed by "M", "H", or "D" to indicate Minutes, Hours, or Days, respectively. Optional. The default is "3". Options for LockingFollowingJournalReader only:
- lockRequestedFilename
The full path of a file which indicates a lock request. After each journal file, the following JournalReader will check to see whether the lock request file is present. If so, the JournalReader will create the lock accepted file (see next option), and enter a polling wait state. This process allows a safe shutdown of a server in following mode.
- lockAcceptedFilename
The full path of a file which indicates that a lock request has been accepted (see previous option). When the JournalReader creates this file, it is in an idle state, and shutting down the server will not have a negative impact on journal processing.

Multicast Journaling

The MulticastJournalWriter is configured with one or more Transport objects, each of which writes an identical set of Journal files to its

destination. So far, these Transport classes are available:

- `fedora.server.journal.readerwriter.multicast.LocalDirectoryTransport`
Writes a set of Journal files to a directory on the local file system. This functions very much like a `MultiFileJournalWriter`, adapted for multicasting.
- `fedora.server.journal.readerwriter.multicast.RmiTransport`
Writes a set of Journal files to an instance of `RmiJournalReceiver` running on the same or another server.

The `MulticastJournalWriter` can write to any combination of these Transports.

If a Transport encounters an error while writing a Journal entry, the action taken depends on whether the Transport has been configured as "crucial" or not. If the Transport is not crucial, the exception is written to the log, and operation continues normally. If the Transport is crucial, the exception is passed on, and the user sees an error status on their request, The server is placed into read-only mode, and will accept no further alterations to the repository until the error is corrected. At least one Transport must be configured as "crucial". The following excerpt from `fedora.fcfg` shows a server that writes Journal files to a local file system, and to two follower systems via RMI.

```
<module role="fedora.server.management.Management" class="fedora.server.journal.Journaler">
  <param name="journalFileSizeLimit" value="100M"/>
  <param name="journalFileAgeLimit" value="1H"/>
  <param name="journalWriterClassname"
    value="fedora.server.journal.readerwriter.multicast.MulticastJournalWriter"/>
  <param name="transport.local.classname"
    value="fedora.server.journal.readerwriter.multicast.LocalDirectoryTransport"/>
  <param name="transport.local.directoryPath" value="/usr/local/journals/archiveFiles"/>
  <param name="transport.local.crucial" value="true"/>
  <param name="transport.follow1.classname"
    value="fedora.server.journal.readerwriter.multicast.rmi.RmiTransport"/>
  <param name="transport.follow1.crucial" value="false"/>
  <param name="transport.follow1.hostName" value="follow1.nsd1.org"/>
  <param name="transport.follow1.service" value="RmiJournalReceiver"/>
  <param name="transport.follow2.classname"
    value="fedora.server.journal.readerwriter.multicast.rmi.RmiTransport"/>
  <param name="transport.follow2.crucial" value="false"/>
  <param name="transport.follow2.hostName" value="follow2.nsd1.org"/>
  <param name="transport.follow2.service" value="RmiJournalReceiver"/>
</module>
```

Options for the Recovery Log

These options are recognized by all of the current `RecoveryLog` components.

- `recoveryLogFilename`
The full path of the disk file that will contain the recovery log. Required for recover mode. There is no default.
- `recoveryLogLevel`
Specifies the amount of information that is written to the recovery log. Optional. The default is "high".
 - `high`
Verbose logging. Writes all Journal information to the log, including method identifiers, arguments, and context.
 - `medium`
Writes partial information to the log, including method identifiers and arguments, but omitting contexts.
 - `low`
Terse logging. Writes only method identifiers to the log, omitting contexts and arguments.

Implementation

This section contains some details of the `Journaler` implementation, including how to configure it, what it produces, and how it operates.

Configuring Fedora

These excerpts from `fedora.fcfg` show three different examples of how to configure the `Journaler` module.

Configuring for normal mode:

```

<module role="fedora.server.management.Management"
  class="fedora.server.journal.Journaler">
  <param name="journalDirectory"
    value="/usr/local/ndr-content/journals/journalFiles"/>
  <param name="journalWriterClassname"
    value="fedora.server.journal.readerwriter.multifile.MultiFileJournalWriter"/>
  <param name="journalFileSizeLimit" value="100M" />
  <param name="journalFileAgeLimit" value="1H" />
</module>
<module role="fedora.server.management.ManagementDelegate"
  class="fedora.server.management.ManagementModule">
  <!-- ManagementModule configuration params go here -->
</module>

```

Configuring for recover mode:

```

<module role="fedora.server.management.Management"
  class="fedora.server.journal.Journaler">
  <param name="journalMode" value="recover"/>
  <param name="journalDirectory"
    value="/usr/local/ndr-content/journals/journalFiles"/>
  <param name="archiveDirectory"
    value="/usr/local/ndr-content/journals/archiveFiles"/>
  <param name="journalReaderClassname"
    value="fedora.server.journal.readerwriter.multifile.MultiFileJournalReader"/>
  <param name="journalRecoveryLogClassname"
    value="fedora.server.journal.recoverylog.RenamingJournalRecoveryLog"/>
  <param name="recoveryLogFilename"
    value="/usr/local/ndr-content/journals/journal.recovery.log"/>
</module>
<module role="fedora.server.management.ManagementDelegate"
  class="fedora.server.management.ManagementModule">
  <!-- ManagementModule configuration params go here -->
</module>

```

Configuring for recover mode in a "following" server:

```

<module role="fedora.server.management.Management"
  class="fedora.server.journal.Journaler">
  <param name="journalMode" value="recover" />
  <param name="journalDirectory"
    value="/usr/local/ndr-content/journals/journalFiles"/>
  <param name="archiveDirectory"
    value="/usr/local/ndr-content/journals/archiveFiles"/>
  <param name="journalReaderClassname"
    value="fedora.server.journal.readerwriter.multifile.LockingFollowingJournalReader"/>
  <param name="lockRequestedFilename"
    value="/usr/local/ndr-content/journals/StopAcceptingJournals.lock"/>
  <param name="lockAcceptedFilename"
    value="/usr/local/ndr-content/journals/AcceptedJournalStop.lock"/>
  <param name="journalRecoveryLogClassname"
    value="fedora.server.journal.recoverylog.RenamingJournalRecoveryLog"/>
  <param name="recoveryLogFilename"
    value="/usr/local/ndr-content/journals/journal.recovery.log"/>
  <param name="followPollingInterval" value="10"/>
</module>
<module role="fedora.server.management.ManagementDelegate"
  class="fedora.server.management.ManagementModule">
  <!-- ManagementModule configuration params go here -->
</module>

```

Journal Content Example

This shows the possible content of an entry in the Journal file. The Context structure was simplified for this example. In normal operation a Context will probably contain more keys and values.

```

<?xml version="1.0" encoding="UTF-8"?>
<FedoraJournal repositoryHash="3|2,1,0" timestamp="2006-06-15T09:45:35.683-0400">
  <JournalEntry method="purgeObject" timestamp="2006-06-15T09:45:35.683-0400"
    clientIpAddress="127.0.0.1" loginId="fedoraAdmin">
    <context>
      <password>fedoraAdmin</password>
      <noOp>false</noOp>
      <now>2006-06-15T09:45:35.683-0400</now>
      <multimap name="environment">
        <multimapkey name="urn:fedora:names:fedora:2.1:environment:httpRequest:authType">
          <multimapvalue>BASIC</multimapvalue>
        </multimapkey>
        <multimapkey name="urn:fedora:names:fedora:2.1:environment:httpRequest:serverIpAddress">
          <multimapvalue>127.0.0.1</multimapvalue>
        </multimapkey>
        <multimapkey name="urn:fedora:names:fedora:2.1:environment:httpRequest:serverPort">
          <multimapvalue>8080</multimapvalue>
        </multimapkey>
        <multimapkey name="urn:fedora:names:fedora:2.1:environment:httpRequest:method">
          <multimapvalue>POST</multimapvalue>
        </multimapkey>
        <multimapkey name="urn:fedora:names:fedora:2.1:environment:httpRequest:protocol">
          <multimapvalue>HTTP/1.1</multimapvalue>
        </multimapkey>
      </multimap>
      <multimap name="subject">
        <multimapkey name="fedoraRole">
          <multimapvalue>administrator</multimapvalue>
        </multimapkey>
        <multimapkey name="urn:fedora:names:fedora:2.1:subject:loginId">
          <multimapvalue>fedoraAdmin</multimapvalue>
        </multimapkey>
      </multimap>
      <multimap name="action"></multimap>
      <multimap name="resource"></multimap>
      <multimap name="recovery"></multimap>
    </context>
    <argument name="pid" type="string">demo:99</argument>
    <argument name="message" type="string">That dog's gonna die.</argument>
    <argument name="force" type="boolean">>false</argument>
  </JournalEntry>
</FedoraJournal>

```

Recovery Log Examples

These examples show excerpts from recovery logs that were created with different "recoveryLogLevel" settings.

Excerpt of recovery log, recoveryLogLevel=low:

```

2006-06-15T09:45:36.605-0400: Event: method='getNextPid',
      file='C:\fedoraJournal20060615.134531.152Z',
      entry='2006-06-15T09:45:30.167-0400'
2006-06-15T09:45:36.714-0400: Call complete:getNextPid

```

Excerpt of recovery log, recoveryLogLevel=medium:

```

2006-06-15T09:45:36.605-0400: Event: method='getNextPid',
      file='C:\fedoraJournal20060615.134531.152Z',
      entry='2006-06-15T09:45:30.167-0400'
  arguments
    numPids='4'
    namespace='newPIDs'
2006-06-15T09:45:36.714-0400: Call complete:getNextPid

```

Excerpt of recovery log, recoveryLogLevel=high:

```

2006-06-15T09:45:36.605-0400: Event: method='getNextPid',
    file='C:\fedoraJournal20060615.134531.152Z',
    entry='2006-06-15T09:45:30.167-0400'
context=fedora.server.journal.entry.JournalEntryContext
environmentAttributes
    urn:fedora:names:fedora:2.1:environment:httpRequest:authType
        BASIC
    urn:fedora:names:fedora:2.1:environment:httpRequest:security
        urn:fedora:names:fedora:2.1:environment:httpRequest:security-insecure
    urn:fedora:names:fedora:2.1:environment:currentDate
        2006-06-15Z
    urn:fedora:names:fedora:2.1:environment:currentTime
        13:45:30.027Z
    urn:fedora:names:fedora:2.1:environment:httpRequest:sessionStatus
        invalid
    urn:fedora:names:fedora:2.1:environment:httpRequest:scheme
        http
    urn:fedora:names:fedora:2.1:environment:httpRequest:clientIpAddress
        127.0.0.1
    urn:fedora:names:fedora:2.1:environment:httpRequest:contentLength
        576
    urn:fedora:names:fedora:2.1:environment:httpRequest:clientFqdn
        localhost
    urn:fedora:names:fedora:2.1:environment:httpRequest:serverIpAddress
        127.0.0.1
    urn:fedora:names:fedora:2.1:environment:httpRequest:serverPort
        8080
    urn:fedora:names:fedora:2.1:environment:currentDateTime
        2006-06-15T13:45:30.027Z
    urn:fedora:names:fedora:2.1:environment:httpRequest:messageProtocol
        urn:fedora:names:fedora:2.1:environment:httpRequest:messageProtocol-soap
    urn:fedora:names:fedora:2.1:environment:httpRequest:method
        POST
    urn:fedora:names:fedora:2.1:environment:httpRequest:contentType
        text/xml; charset=utf-8
    urn:fedora:names:fedora:2.1:environment:httpRequest:serverFqdn
        localhost
    urn:fedora:names:fedora:2.1:environment:httpRequest:protocol
        HTTP/1.1
subjectAttributes
    fedoraRole
        administrator
    urn:fedora:names:fedora:2.1:subject:loginId
        fedoraAdmin
actionAttributes
resourceAttributes
    urn:fedora:names:fedora:2.1:resource:object:nPids
        4
recoveryAttributes
    info:fedora/fedora-system:def/recovery#pidList
        newPIDs:1
        newPIDs:2
        newPIDs:3
        newPIDs:4
password='*****'
noOp=false
now=false
arguments
    numPids='4'
    namespace='newPIDs'
2006-06-15T09:45:36.714-0400: Call complete:getNextPid

```

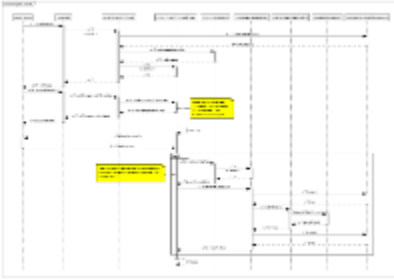
Sequence Diagrams

These UML diagrams give an overview of the control sequences used in creating a Journal (normal mode) or consuming a Journal (recover mode).

Creating the journal (click on image to enlarge).



Consuming the journal (click on image to enlarge).



Resource Index

Introduction

The Resource Index is the Fedora module that provides the infrastructure for indexing relationships among objects and their components. Examples of relationships between digital objects include well-known management relationships such as the part-whole links between individual chapters and a book, and semantic relationships useful in digital library organization such as those expressed within the [Functional Requirements for Bibliographic Records \(FRBR\)](#).

Fedora expresses relationships by defining a base relationship ontology [Fedora Relationships](#) using RDFS and provides a slot in the digital object abstraction for RDF expression of relationships based on this ontology. Assertions from other ontologies may also be included along with the base Fedora relationships. All relationships are represented as a graph that can be queried using an RDF query language. The query interface to the Resource Index is exposed as a Web service [riearch](#).

Implementation

The Fedora object model can be abstractly viewed as a directed graph, consisting of *internal* arcs that relate digital object nodes to their dissemination nodes and *external* arcs between digital objects. The Resource Index is a Fedora service that allows storage and query of this graph. The Resource Index is automatically updated whenever an object is added or modified.

The Resource Index builds on the RDF primitives build within the semantic web community. Fedora supplies a base relationship ontology [Fedora Relationships](#) (defining a core set of internal and external relationships) that can co-exist with domain-specific relationship ontologies from other namespaces. Each digital object's external relationships to other digital objects are expressing in RDF/XML within a reserved Datastream in the respective object. A relationship graph over the digital objects in the repository can then be derived by merging the internal relationships implied by the Fedora object model with the external relationships explicitly stated in their relationship Datastreams.

System Relationships & Properties

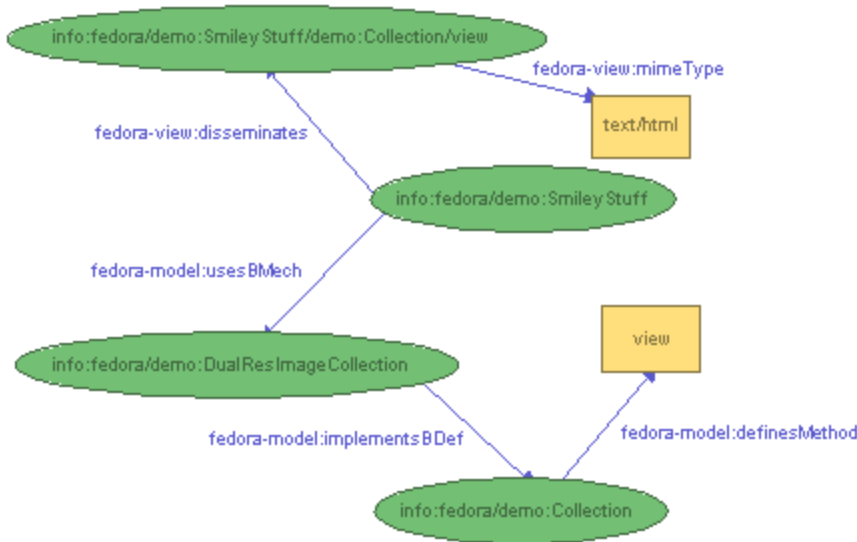
The Fedora base ontology describes such relations and properties as the behavior definition implemented, behavior mechanism used, creation date, state, and mime-type.

In the figure below, the graph (abbreviated for clarity) represents three objects in the repository. The object "demo:SmileyStuff" uses the Service Deployment "demo:dualResImageCollection", which in turn implements the Service Definition "demo:Collection".



Note

The following picture has not yet been updated to conform with the Fedora 3 ontology. However, it is useful for demonstrating the general concepts described in this section.

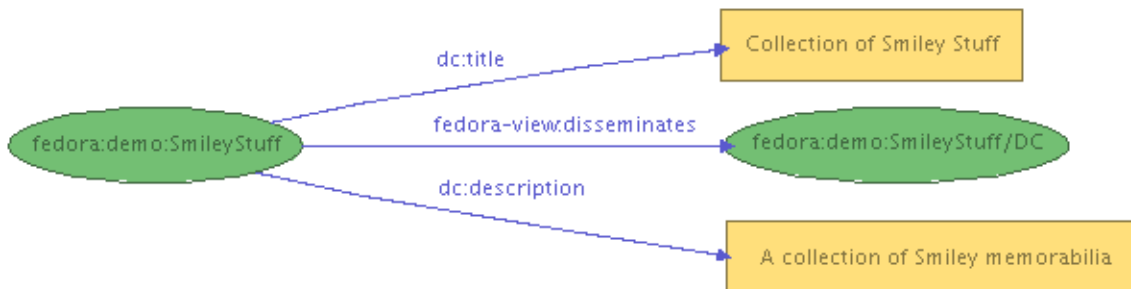


Dublin Core statements, as shown in the figure below, are automatically extracted from an object's DC Datastream and inserted into the Resource Index Datastream, as shown in the figure below:



Note

The following picture has not yet been updated to conform with the Fedora 3 ontology. However, it is useful for demonstrating the general concepts described in this section.



User-defined Relationships

The Resource Index will automatically index object-to-object relationships defined in the RELS-EXT Datastream and datastream relationships defined in the RELS-INT datastream. Please consult [Digital Object Relationships](#) for more information.

Configuration

Please note that many configuration changes require a full rebuild of the Resource Index to ensure consistency. For example, turning the Resource Index service off and on again will result in an inconsistent state, as the Resource Index will know nothing about the digital objects created or modified while the service was not operating. Similarly, enabling full-text indexing [Full-Text](#) after the repository has already been populated will only add new objects to the full-text model. In general, the only safe configuration changes to make on a running repository are limited to the performance-related pool, buffer and flush parameters. In all cases, configuration changes require a restart of the Fedora server before taking effect.

The Resource Index is configured within two sections of `fedora.fcfig`, `module` and `datastore`.

Module Configuration

The Resource Index module is configured with `fedora.fcfig`. Here's an example of a Resource Index module configuration that uses Mulgara with delayed updates:

```
<module role="fedora.server.resourceIndex.ResourceIndex"
  class="fedora.server.resourceIndex.ResourceIndexModule">
  <param name="level" value="1"/>
  <param name="datastore" value="localMulgaraTriplestore"/>
  <param name="syncUpdates" value="false"/>
</module>
```

Here's another example, this time using MPTStore with immediate updates:

```
<module role="fedora.server.resourceIndex.ResourceIndex"
  class="fedora.server.resourceIndex.ResourceIndexModule">
  <param name="level" value="1"/>
  <param name="datastore" value="localPostgresMPTTriplestore"/>
  <param name="syncUpdates" value="true"/>
</module>
```

An explanation of the parameters and their possible values:

- **level** – Sets the operating level of the Resource Index.
 - 0 – Off: the Resource Index will not load at server startup.
 - 1 – On: the Resource Index will index system properties, inter & intra-object relationships, and user-defined relationships.
- **datastore** – The id of the datastore to use with the Resource Index. The referenced datastore must assert a connectorClassName parameter with a valid Trippi Connector class.
- **syncUpdates** – Whether to flush the triple buffer before returning from object modification operations. This defaults to false. Specifying this as true will ensure that RI queries immediately reflect the latest triples. Specifying false will not provide this guarantee, but can reduce roundtrip time for API-M operations (especially when using Mulgara).

Mulgara Datastore Configuration

The example datastore configuration below (with the path parameter modified for the installation environment) would provide a local Mulgara triplestore that buffers up to 20,000 triples in memory at a time or waits for 5 seconds of buffer inactivity before flushing them to disk. Because writing triples to disk is a relatively expensive operation, the buffer takes advantage of Mulgara's bulk update handler to ingest a mass of triples at a time. The performance gain is significant during a bulk ingest of objects. The size or inactivity interval of the buffer may be adjusted according to performance needs and physical memory capacity.

```
<datastore id="localMulgaraTriplestore">
  <param name="connectorClassName" value="org.trippi.impl.mulgara.MulgaraConnector"/>
  <param name="remote" value="false"/>
  <param name="path" value="/opt/fedora/store/resourceIndex"/>
  <param name="serverName" value="fedora"/>
  <param name="modelName" value="ri"/>
  <param name="poolInitialSize" value="3"/>
  <param name="poolMaxGrowth" value="-1"/>
  <param name="readOnly" value="false"/>
  <param name="autoCreate" value="true"/>
  <param name="autoTextIndex" value="false"/>
  <param name="memoryBuffer" value="true"/>
  <param name="autoFlushDormantSeconds" value="5"/>
  <param name="autoFlushBufferSize" value="20000"/>
  <param name="bufferSafeCapacity" value="40000"/>
  <param name="bufferFlushBatchSize" value="20000"/>
</datastore>
```

An explanation of the parameters and their possible values follows. Certain parameters require other parameters, as indicated in the hierarchy below. Optional parameters are also indicated below. As noted previously, many of these parameters, with the exception of the pool, buffer, and flush parameters, cannot be changed on a running repository without a full rebuild of the Resource Index.

- **connectorClassName** – The name of the Trippi Connector class used to communicate with the triplestore. When using Mulgara, the value should be as follows:
 - org.trippi.impl.mulgara.MulgaraConnector
- **remote** – Tells the connector to communicate with Mulgara in remote or local mode.
 - true – If remote is true, the host parameter must be specified.
 - host – The hostname where Mulgara is running. The hostname can be specified as an argument when starting Mulgara, e.g. java -jar mulgara-2.0.0.jar -k localhost
 - port – The rmi port Mulgara is running on (default is 1099).
 - false – If remote is false, path parameter must be specified.
 - path – The local path to the main triplestore directory.

- **serverName** – The server name for rmi binding. When configuring a remote instance of Mulgara, the server name defaults to "server1". To change this default, use the "-s" argument, e.g. `java -jar mulgara-2.0.0.jar -s fedora`
- **modelName** – The name of the model to use.
- **poolInitialSize** – The initial size of the session pool used for queries. Note: A value of 0 will cause the Resource Index to operate in synchronized mode: concurrent read/write requests are put in a queue and handled in FIFO order; this will severely impair performance and is only intended for debugging.
- **poolMaxGrowth** – Maximum number of additional sessions the pool may add. If specified as -1, no limit will be placed on pool growth.
- **poolSpareSessions** – The number of spare sessions to proactively make available. If unspecified, this defaults to 0, which means that additional sessions will only be created as needed. Note that if poolMaxGrowth is 0, the value of this parameter is inconsequential.
- **readOnly** – Whether the triplestore should be read-only. Most Fedora repositories will set this to false.
 - true - No additional parameters need to be set if readOnly is true.
 - false - The following parameters must also be set if readOnly is false.
 - **autoCreate** – Whether to create the model if it doesn't already exist. At startup, the model will be automatically created. In addition, an XML schema datatype model named "xsd" will also be automatically created.
 - true
 - false
 - **autoTextIndex** – Whether to propagate adds/deletes to a full-text [Full-Text](#) model automatically. While a very useful feature, enabling full-text indexing adds significantly to object ingest times. If true, the text model will be named `modelName-fullText`. Note that if this is true and autoCreate is true, the text model will also be created if it doesn't already exist.
 - true
 - false
 - **memoryBuffer** – Whether to use a memory buffer or a database buffer for write operations. The buffer is where triples are stored before they're actually written. Normally, a memory buffer will be sufficient. However, in certain server environments, a database buffer is the better option, offering higher capacity and persistence.
 - true
 - false
 - **dbDriver** – The JDBC driver class name. This must be in the classpath.
 - `com.mysql.jdbc.Driver`
 - `com.mckoi.JDBCdriver`
 - `oracle.jdbc.driver.OracleDriver`
 - **dbURL** – The JDBC URL for the database. Examples: For MySQL, `jdbc:mysql://localhost/mydb` would use the local database named mydb. For McKoi, `jdbc:mckoi://localhost:9157/` would use the local database at port 9157. For oracle, `jdbc:oracle:thin:@localhost:1521:mydb` would use the thin driver to connect to the local database named mydb at port 1521.
 - **dbUsername** – The database username.
 - **dbPassword** – The password for the db user.
 - **dbTableName** – The table for buffered triple updates. This table must already exist with the following five columns: num (a large numeric type), action (char(1)), subject, predicate, and object (all large varchar or text types).
- **autoFlushDormantSeconds** – Seconds of buffer inactivity that will trigger an auto-flush. If this threshold is reached, flushing will occur in the background, during which time the buffer is still available for writing.
- **autoFlushBufferSize** – The size at which the buffer should be auto-flushed. If this threshold is reached, flushing will occur in the background, during which time the buffer is still available for writing.
- **bufferSafeCapacity** – The maximum size the buffer can reach before being forcibly flushed. If this threshold is reached, flushing will occur in the foreground and the buffer will be locked for writing until it is finished. This should be larger than `autoFlushBufferSize`.
- **bufferFlushBatchSize** – The number of updates to send to the triplestore at a time. This should be the same size as, or smaller than `autoFlushBufferSize`.

Remote Mulgara Datastore Configuration

Instead of using the Mulgara instance embedded in Fedora a separate instance of Mulgara can be used, running in its own JVM (which can be on a different server). This may give performance advantages as the resources for the Mulgara JVM can then be configured separately from Fedora itself, and the data files can be located on separate storage.

Mulgara can be downloaded from the [Mulgara download page](#).

Use the same version of Mulgara that Fedora uses - this can be determined from the libraries included in the Fedora web application, which will include a jar file `mulgara-core.x.y.z.jar` - use version "x.y.z" of Mulgara (it is possible to use other versions but this will involve updating the Mulgara and Trippi libraries that Fedora uses - and any dependencies). Fedora 3.3 uses Mulgara version 2.1.4.

An example datastore configuration for a remote Mulgara instance follows. The Resource Index module must be configured to use this datastore.

```

<datastore id="remoteMulgaraTriplestore">
  <param name="connectorClassName" value="org.trippi.impl.mulgara.MulgaraConnector"/>
  <param name="remote" value="true"/>
  <param name="host" value="localhost"/>
  <param name="port" value="1099"/>
  <param name="serverName" value="fedora"/>
  <param name="modelName" value="ri"/>
  <param name="poolInitialSize" value="3"/>
  <param name="poolMaxGrowth" value="-1"/>
  <param name="readOnly" value="false"/>
  <param name="autoCreate" value="true"/>
  <param name="autoTextIndex" value="false"/>
  <param name="memoryBuffer" value="true"/>
  <param name="autoFlushDormantSeconds" value="5"/>
  <param name="autoFlushBufferSize" value="20000"/>
  <param name="bufferSafeCapacity" value="40000"/>
  <param name="bufferFlushBatchSize" value="20000"/>
</datastore>

```

The parameters for this are largely the same as for a local Mulgara datastore; except **remote** is specified as "true", and **host** and **port** have been specified for the RMI connection to the remote Mulgara instance. The **serverName** parameter must match the server name that Mulgara is configured to use (by default this is "server1", the examples below configure Mulgara instead to use "fedora" as the server name).

Starting and Stopping Mulgara

The Mulgara website has documentation on [starting and stopping Mulgara](#) and on [command line parameters](#).

An example command for starting Mulgara is

```
java -jar mulgara-x.y.z.jar -p 8088 -u 8089 -s fedora
```

This configures Mulgara to start using ports 8088 and 8089 for the [web user interface](#) as Mulgara's default ports for the Web UI will conflict with Tomcat. Alternatively start Mulgara using `--nohttp` to disable the Web UI. The server name is specified as "fedora" to match the Fedora datastore configuration.

By default Mulgara stores its data in a folder relative to where it was started, instead this can be specified with a `--path` parameter.

Mulgara can be stopped using a command line interrupt, by pressing **Control-C**. Alternatively it can be run using `nohup`, and the following command can then be used to stop Mulgara:

```
java -jar mulgara-x.y.z.jar -x
```

MPTStore Datastore Configuration

The example datastore configuration below would provide a local MPTStore triplestore backed by Postgresql.

```

<datastore id="localPostgresMPTTriplestore">
  <comment>
    Example local MPTStore backed by Postgres.
    To use this triplestore for the Resource Index:
    1) In fedora.fcfg, change the "datastore" parameter of the
       ResourceIndex module to localPostgresMPTTriplestore.
    2) Login to your Postgres server as an administrative user and
       run the following commands:
       CREATE ROLE "fedoraAdmin" LOGIN PASSWORD 'fedoraAdmin'
         NOINHERIT CREATEDB
         VALID UNTIL 'infinity';
       CREATE DATABASE "riTriples"
         WITH ENCODING='SQL_ASCII'
         OWNER="fedoraAdmin";
    3) Make sure you can login to your Postgres server as fedoraAdmin.
    4) Download the appropriate Postgres JDBC 3 driver from
       http://jdbc.postgresql.org/download.html
       and make sure it's accessible to your servlet container.
       If you're running Tomcat, putting it in common/lib/ will work.
  </comment>
  <param name="connectorClassName" value="org.trippi.impl.mpt.MPTConnector"/>
  <param name="ddlGenerator" value="org.nsdl.mptstore.impl.postgres.PostgresDDLGenerator"/>
  <param name="jdbcDriver" value="org.postgresql.Driver"/>
  <param name="jdbcURL" value="jdbc:postgresql://localhost/riTriples"/>
  <param name="username" value="fedoraAdmin"/>
  <param name="password" value="fedoraAdmin"/>
  <param name="poolInitialSize" value="3"/>
  <param name="poolMaxSize" value="10"/>
  <param name="backslashIsEscape" value="true"/>
  <param name="fetchSize" value="1000"/>
  <param name="autoFlushDormantSeconds" value="5"/>
  <param name="autoFlushBufferSize" value="1000"/>
  <param name="bufferFlushBatchSize" value="1000"/>
  <param name="bufferSafeCapacity" value="2000"/>
</datastore>

```

Search Interface

The Resource Index Search interface is exposed in a REST architectural style, providing a stateless query interface that accepts queries by value or by reference [Fedora Relationships](#).

The query interface to the Resource Index currently supports two RDF query languages, iTQL (Mulgara-only), and SPO (Mulgara and MPTStore). Support for SPARQL is planned for a future release.

Please consult the [Resource Index Search](#) documentation for more information.

Resource Index Demo

Demonstration objects that utilize the Resource Index are included in the Fedora distribution. Please see the [Demonstrations](#) documentation for more information.

Fedora Guide To Triples (and estimator for size of Resource Index)

To understand what triples are indexed in the Fedora Resource Index, and to estimate the size of your Mulgara triplestore consult document named [Triples in the Resource Index](#).

Related Materials

- [Fedora: An Architecture for Complex Objects and their Relationships](#), Carl Lagoze, Sandy Payette, Edwin Shin, and Chris Wilper, 2004.
- [Mulgara Full-Text Models](#)
- [Fedora Relationship Ontology](#)
- [Resource Index Search](#)

Triples in the Resource Index

Triples in the Resource Index

The resource index stores system and user-controlled metadata about each object in the repository in the form of RDF triples. The number and type of triples stored depends on the content of the object. This document describes the triples that may exist in the resource index for a given object. The color of each row indicates how many of each kind of RDF triple can be expected. See the key on the right.

Cardinality Key
Zero or more
Zero or one
Exactly one
One or more

Namespaces Used

For brevity, the URIs used in this document are shown in abbreviated form. To determine the unabbreviated form of any such URI, replace the **Prefix** with the associated **Namespace URI** below.

Prefix	Namespace URI
dc:	http://purl.org/dc/elements/1.1/
fedora-model:	info:fedora/fedora-system:def/model#
fedora-view:	info:fedora/fedora-system:def/view#
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#

Breakdown

Base Triples

The following triples will exist for any object, irrespective of its Datastream content.

Subject	Predicate	Object	Cardinality
info:fedora/\$PID	fedora-model:createdDate	(date created in UTC)	Exactly One
info:fedora/\$PID	fedora-view:lastModifiedDate	(date modified in UTC)	Exactly One
info:fedora/\$PID	fedora-model:state	fedora-model:Active fedora-model:Inactive fedora-model:Deleted	Exactly One
info:fedora/\$PID	fedora-model:owner	(not used)	Exactly One
info:fedora/\$PID	fedora-model:label	(any string)	Exactly One

Dublin Core Triples

Every object in Fedora has a Dublin Core ("DC") Datastream. The following triples are derived from its content, which may consist of any number of each of the 15 unqualified Dublin Core elements. If unprovided at ingest, the DC Datastream will be automatically created with minimal information (a *dc:title* and a *dc:identifier*).

Subject	Predicate	Object	Cardinality
info:fedora/\$PID	dc:title	(any string)	One or More
info:fedora/\$PID	dc:identifier	(any string)	One or More
info:fedora/\$PID	(any other dc predicate)	(any string)	Zero or More

RELS-EXT and RELS-INT Triples

Subject	Predicate	Object	Cardinality
info:fedora/\$PID	(any non-reserved predicate)	(any URI or literal)	Zero or More

Content Model Architecture Triples

Note: If not explicitly provided the Fedora Repository will assume there is a `fedora-model:hasModel` relation asserted from a Data Object to a system-supplied base CModel Object satisfying the cardinality constraint. This relation should be explicitly stated in the REL-EXT Datastream as a recommended practice. Also note that, while CModel, SDef, and SDep objects may be created without asserting their respective relations, they

will not perform their functions without them.

Note: In Fedora 3.0, though permitted, it is not recommended that one SDep object be used to deploy a service for more than one CModel-SDef pair. Future versions of Fedora are likely to provide better support for this configuration.

Subject	Predicate	Object	Cardinality
info:fedora/\$PID	fedora-model:hasModel	info:fedora/\$CMODEL_PID	One or More
info:fedora/\$CMODEL_PID	fedora-model:hasService	info:fedora/\$SDEF_PID	Zero or More
info:fedora/\$SDEP_PID	fedora-model:isDeploymentOf	info:fedora/\$SDEF_PID	Zero or More
info:fedora/\$SDEP_PID	fedora-model:isContractorOf	info:fedora/\$CMODEL_PID	Zero or More

Datastream Triples

Subject	Predicate	Object	Cardinality
info:fedora/\$PID	fedora-view:disseminates	info:fedora/\$PID/\$DSID	Exactly One
info:fedora/\$PID/\$DSID	fedora-view:disseminationType	info:fedora/*/ \$DSID	Exactly One
info:fedora/\$PID/\$DSID	fedora-view:mimeType	(any mime type string)	Exactly One
info:fedora/\$PID/\$DSID	fedora-view:lastModifiedDate	(date modified in UTC)	Exactly One
info:fedora/\$PID/\$DSID	fedora-model:state	fedora-model:Active fedora-model:Inactive fedora-model:Deleted	Exactly One
info:fedora/\$PID/\$DSID	fedora-view:isVolatile	(true if R or E, false if M or X)	Exactly One

Calculating Triples/Object

You can use the following table to estimate the number of triples for each kind of object in your Fedora repository.

Calculation Method	# Triples
Every object automatically gets the following triples: <ul style="list-style-type: none"> • Base: 6 • Dublin Core (dc:title and dc:identifier): 2 • Content Model Architecture (fedora-model:hasContentModel): 1 • Datastream Triples (for DC): 6 	15
If the DC datastream has any elements besides the standard dc:title and dc:identifier, add 1 for each additional element.	
If the object has a RELS-EXT and/or a RELS-INT datastream, add 7 , then add 1 for each statement asserted therein (except fedora-model:hasContentModel which has been counted above).	
For each additional datastream, add 7 .	

Security

Security Options

Securing your repository is achieved through many activities and not the result of a single feature. Attending to the physical security of the servers and good systems administration practices are a necessary first step. It is recommended that you prepare a security policy to determine the requirements, processes, and practices appropriate for your repository. Using your security policy, you choose which of Fedora's many options are right for your needs.

Quick Start Guide to Securing Your Repository

Here is a quick start guide that describes what you will need to do to configure your Fedora repository. It is recommended that you start with the new installer and one of the base security configurations it creates, and become familiar with the new installation and default security features. Then you can go back and experiment with customizing various aspects of your repository configuration and policies.

1. Select a base security configuration by running Fedora installer jar
2. Optionally customize `fedora.fcfg` for your repository

3. Choose basic security, the XACML Policy Engine, or the new Fedora Security Layer (FeSL)
4. Optionally customize XACML policies (repository-wide and object-specific policies)
5. Optionally customize fedora-users.xml for your repository and users
6. Optionally customize web.xml to use servlet filters for authentication and user attributes
7. Start the fedora server

- Remember that beSecurity has been turned off

Introduction to Fedora Servlet security filters

We here assume that you have already installed Fedora, in either quick or custom varieties. This document gives advice on using Fedora's servlet security filters and its surrogate feature. These filters authenticate Fedora users and/or provide user attributes to use in XACML authorization.

Fedora's servlet security filters are configured in the web deployment descriptor file (web.xml), typically in Fedora's webapp directory in whatever servlet container (e.g., Tomcat) you've deployed Fedora in. Section "Specifying Filter Configuration" in [The Essentials of Filters](#) discusses the format of specifying filters and filter-mappings and gives more information on the format of the servlet filter section of web.xml. (That section does not have an anchored location to link to directly.) It may be helpful to use the Fedora web.xml as guide while reading this document.

The Fedora installer will have configured several servlet filters in the correct order. Leave these in the order given, with the filter elements grouped first, and then the grouped filter-mapping elements following as a second group. Within either filter or filter-mapping unit, the filter definitions are ordered: SetupFilter, XmlUserfileFilter, (LdapFilterForAttributes), (LdapFilterForGroups), EnforceAuthnFilter, FinalizeFilter. The filters in parenthesis are optional, and won't be installed by default. Again, retain this order and if you add a filter, use the place indicated.

The filter-mappings of the EnforceAuthnFilter determine which Fedora urls require user authentication. The installer will set up these various mappings either for api-m alone or for both api-m and api-a urls/servlets. You can customize web.xml for this, likely by adding or deleting mappings for this filter, if you need and know.

Parameter settings are specific to a servlet filter, and are given below for the Fedora servlet security filters. Here is the format which the specification takes in web.xml:

```
<filter>
  <filter-name>LdapFilterForAttributes</filter-name>
  <filter-class>fedora.server.security.servletfilters.ldap.FilterLdap</filter-class>
  <init-param>
    <param-name>authenticate</param-name>
    <param-value>>false</param-value>
    . . .
  </init-param>
</filter>
```

Use this format to define in web.xml the parameter settings you need.

General Parameters

The following parameters are useful for XmlUserfileFilter, LdapFilterForAttributes, and LdapFilterForGroups.

parameter	use	default	note
authenticate	whether the current filter should attempt to authenticate the user	true	if a previous filter has already authenticated the user, this filter doesn't try also for the current request. a value of "false" still permits associated-filters from providing user attributes
associated-filters	comma-separated list of previous filters, of any number including none. if any of these listed filters have authenticated the current user, then this filter will provide attributes for the user if it can.	current filter	if this parm is specified, the current filter must be explicitly named, i.e., it's no longer implicitly in the list
lookup-success-timeout-unit	how long to cache a successful lookup (whether for authentication or attribute/group lookup) – the <i>units</i> themselves	minute	
lookup-success-timeout-duration	how long to cache a successful lookup (whether for authentication or attribute/group lookup) – the <i>number</i> of units	10	
authn-failure-timeout-unit	how long to cache user not found (whether for authentication or attribute/group lookup) – the <i>units</i> themselves	second	

authn-failure-timeout-duration	how long to cache user not found (whether for authentication or attribute/group lookup) – the <i>number</i> of units	1	
lookup-exception-timeout-unit	how long to cache a problematic lookup (whether for authentication or attribute/group lookup) – the <i>units</i> themselves	second	
lookup-exception-timeout-duration	how long to cache a problematic lookup (whether for authentication or attribute/group lookup) – the <i>number</i> of units	1	

Parameters for LDAP servlet filter for user attributes

The following parameters are useful for either `LdapFilterForAttributes` or `LdapFilterForGroups`. The example values are chosen for `LdapFilterForAttributes`. If you are setting up this filter, use "LdapFilterForAttributes" as filter-name, "fedora.server.security.servletfilters.Ldap.FilterLdap" as filter-class, and choose values from the parameters below which fit your Ldap directory configuration for reading user *attributes*. You can also use parameters for either/both Ldap authentication/binding and/or the surrogate feature, as explained elsewhere in this document. You may need to talk to your directory administrator to find out these settings.

parameter	use	example
url	internet address of directory server	ldap://ldap.virginia.edu:389/
search-base	ldap-style specification where in directory to base user search	o=University of Virginia,c=US
search-filter	ldap-style specification how to conduct user search	(uid={0})
id-attribute	directory attribute which is user id	uid
attributes	comma-separated list of directory attributes to use as user's xacml subject attributes	mailAlternateAddress,eduPersonAffiliation

Parameters to use LDAP servlet filter for user group memberships

The following parameters are useful for `LdapFilterForGroups` and have example values chosen for `LdapFilterForGroups`. If you are setting up this filter, use "LdapFilterForGroups" as filter-name, "fedora.server.security.servletfilters.Ldap.FilterLdap" as filter-class, and choose values for the parameters below which fit your Ldap directory configuration for reading *group* memberships. This will be more specific to your directory than for reading user attributes. You can also use parameters for either/both Ldap authentication/binding and/or the surrogate feature, as explained elsewhere in this document. You may need to talk to your directory administrator to find out these settings. Some directories will store no group memberships, or store them in a way for which this servlet filter isn't configurable.

parameter	use	example
url	internet address of directory server	ldap://pitchfork.itc.virginia.edu:389/
search-base	ldap-style specification where in directory to base user search	ou=Groups,o=University of Virginia,c=US
search-filter	ldap-style specification how to conduct user search	(memberUid={0})
id-attribute	directory attribute which is user id	uid
attributes	comma-separated list of directory attributes to use as user's xacml subject attributes	cn
attributes-common-name	return all attribute values under this name; this override prevents using the awkward "cn" as an XACML subject attribute	groups

Parameters for Authentication and Binding with LDAP

The following parameters are useful for either `LdapFilterForAttributes` or `LdapFilterForGroups`, and are used with other values given elsewhere in this document. You must choose values from the parameters below which fit your Ldap directory configuration for binding to the directory. You may need to talk to your directory administrator to find out these settings.

parameter	use	example	note
-----------	-----	---------	------

security-authentication	specification of how to bind to directory server	simple	if specified, a directory bind will occur. so neither an anonymous connect nor a field-compare authentication will occur. if security-principal and security-credentials are specified, they are used to bind the connection. if they are not not specified, a bind is attempted with the user's credentials, and success determines user authentication, if authenticate is also specified
security-principal	privileged (non-user) id with which to bind to directory server	site-specific; get from your directory administrator	
security-credentials	privileged password with which to bind to directory server	site-specific; get from your directory administrator	
password-attribute	directory attribute which is user password. if given, marks that user password will be compared to the directory to authenticate.	uid	

Obviously, some combinations of these values are incompatible, and yet others necessary to achieve certain aims.

Parameters for Surrogate Feature

The surrogate feature supports end-user authentication by a Fedora client or web server front-end. The surrogate user is represented in the request directly (in the usual header) and is authenticated by Fedora as usual. A From: header holds the identity of the represented virtual user.

parameter	use	example	note
surrogate-attribute	name of attribute which a user authenticated by this or an earlier filter must have to become a surrogate user. Any value of the attribute is acceptable.	SURROGATE	
surrogate-associated-filters	comma-separated list of previous filters, of any number including none. if any of these listed filters have authenticated a surrogate user and so there is a virtual user, then this filter will provide attributes for the virtual user if it can.		if this parm is specified, the current filter must be explicitly named, i.e., it's not implicitly in the list.

Authorization via XACML

Fedora 2.0 hardcoded minimal authorization constraints, beyond those provided by specifications in Tomcat's web.xml file. Fedora now exposes these to customization by encoding them in the XACML standard. A complete description can be found in the documentation for the [Fedora Authorization with XACML Policy Enforcement](#).

Default Repository Policies

Fedora ships with a set of [default repository-wide XACML policies](#) that approximate the minimal security level provided by Fedora. This set of repository-wide policies includes the following policies:

Custom Policies

Note that the default repository policies enforce a minimal level security (e.g., API-A is totally unrestricted). If you need a more customized level of access control what is provided by the default, you will need to add additional repository-wide policies or individual object-specific policies to customize your access environment. Refer to the [Fedora XACML Policy Writing Guide](#) document for more information about how to construct policies for your repository.

Authentication and User Attributes

Introduction

This document collects technical notes on how to supply user attributes for evaluation of Fedora's XACML authorization policies.

For a practical introduction to using Fedora's servlet filters for authentication and user attributes, see [Securing Your Fedora Repository](#). The

current document is a technical appendix to that section.

Injecting Arbitrary User Attributes

Fedora 2.1.1 provided a mechanism to include user attributes from arbitrary source(s), and merge them with attributes provided by Tomcat realms or login modules. Fedora 2.2 continues this with its servlet filter approach, independent of specific servlet container. The Fedora code looks for a request attribute whose name equals the string constant `fedora.server.Context.FEDORA_AUX_SUBJECT_ATTRIBUTES`. A request attribute found under that name is taken as a Map, mapping names to values, and so giving additional user subject attributes. Currently, name must be a String and this is unlikely to change. Value must also be a String, and later this might be relaxed to include `String[]`, to allow attributes with multiple values. Other types of value are not serviced. The effect within Fedora of having a key => value pair "a" => "b" in the Map is the same as the current user having an attribute named "a" with value "b", e.g., as defined in `fedora-users.xml`.

An arbitrary site-supplied servlet filter must create the map and populate it, and put it into the http servlet request as attribute named `fedora.server.Context.FEDORA_AUX_SUBJECT_ATTRIBUTES`. Fedora will then look for it, and use the attributes in xacml authorization. This services any subject attribute source, and remains source-neutral, i.e., you could use it to implement Shibboleth, but it doesn't favor Shibboleth.

If doing this, it would be best to isolate your interface code to be independent both of being in a servlet filter and also of interacting with the Map. This is so you could later refactor the code more in line with Fedora 2.2 servlet filters.

This means of introducing arbitrary attributes has been tested by another developer successfully before Fedora 2.2, but not yet with Fedora 2.2. Eventually, the Fedora servlet filter approach will be documented as a best practice to providing user attributes.

Fedora Security Layer (FeSL)

Introduction

The Fedora Security Layer (FeSL) is the future replacement of Fedora's legacy authentication and authorization system. Fedora 3.3 introduces FeSL as an experimental option, a preview of things to come.

FeSL offers a new authentication layer that reduces complexity and allows for integration with more authentication systems. FeSL authentication is built on the [Java Authentication and Authorization Service \(JAAS\)](#). JAAS offers a well-established standard for authentication implementations. Among the benefits that JAAS provides are: cascading authentication, re-use of existing JAAS authentication modules and comprehensive documentation for those who wish to write their own authentication modules.

FeSL extends and improves upon Fedora's legacy XACML-based authorization. Notably, FeSL provides hierarchical enforcement of access control policies. Access controls can be set at the collection level, object level or datastream level. As a result collection level policies can be applied to all collection members.

FeSL was produced under a community funding model, made possible by generous contributions from a number of institutions, including University of Prince Edward Island, Stanford University, University of Hull, University of Virginia and MediaShelf.

FeSL Installation

FeSL Authentication

FeSL Authorization

Open Issues

[\[FCREPO-986\] Increased flexibility to derive the configuration of a repository](#)

[\[FCREPO-600\] RISearch queries with FeSL ignore format parameter](#)

[\[FCREPO-957\] Ability to access more subject attributes in FeSL policies](#)

[\[FCREPO-953\] Create an AttributeFinderModule based on XML content of datastream](#)

[\[FCREPO-750\] Ability to clear the FeSL response cache](#)

[\[FCREPO-943\] Review and update FeSL bootstrap policies](#)

[\[FCREPO-795\] Integrate FeSL with module/spring architecture](#)

[Add a new issue](#)

FeSL Installation

Prerequisites

FeSL Authentication (AuthN) and Authorisation (AuthZ) are separate components, and can be selected as separate installation options (see below).

FeSL Authorization uses Oracle Berkeley DB XML (DBXML) for its policy data store. In order to enable FeSL Authorization in during Fedora installation, DBXML must first be installed.

If you only wish to enable FeSL Authentication, you do not need to install DBXML. FeSL AuthN is the default Fedora authentication mechanism.

Oracle Berkeley DB XML

FeSL has been tested with DBXML 2.5.13, which is available from <http://www.oracle.com/technology/software/products/berkeley-db/xml/index.html>. 32-bit Windows users can use the .msi installer, but other platforms will require a build from source, e.g.:

```
./buildall.sh --prefix=/usr/local/dbxml-2.5.13 --enable-java
```

After installing DBXML, the DBXML_HOME, LD_LIBRARY_PATH and DYLD_LIBRARY_PATH environment variables must be set, e.g.:

```
export DBXML_HOME=/usr/local/dbxml-2.5.13
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:${DBXML_HOME}/lib
export DYLD_LIBRARY_PATH=${DBXML_HOME}/lib:$DYLD_LIBRARY_PATH
```

(DYLD_LIBRARY_PATH might only be necessary for OS X)

On Windows, the PATH and CLASSPATH environment variables need to be updated to reference DBXML - dbxmlvars.bat in the DBXML home directory should be run to do this.

Installation

FeSL is a custom option in the Fedora Installer. See the [Installation and Configuration Guide](#) for a detailed description of general Fedora installation options. Set "Enable FeSL AuthN" to "true" to enable FeSL Authentication in your Fedora installation, and set "Enable FeSL AuthZ" to true to enable FeSL Authorization (ensuring you have already installed DBXML).

By default, Fedora will load any policies located in FEDORA_HOME/pdp/policies on startup.

FeSL Authentication

Configuration

The default location for the Fedora JAAS configuration file is: \$FEDORA_HOME/server/config/jaas.conf. This default location can be overridden by specifying an alternative in the Fedora Web Application's web.xml file.

A detailed tutorial on the JAAS configuration file can be found at: <http://java.sun.com/j2se/1.5.0/docs/guide/security/jaas/tutorials/LoginConfigFile.html>.

To provide a basic overview, the structure of the JAAS configuration file is

```
application-name {
  module-class01 mode
  option=value
  option=value
  ...
  option=value;

  module-class02
  mode option=value
  option=value
  ...
  option=value;
};
```

The application-name can be any name and is referenced by the application performing the authentication. The configuration file can contain multiple application-names, each with different configurations.

Each application section can contain one or more login modules. Each login module has a flag which specifies how it will behave and can also have an unlimited number of options. Login modules are executed in sequence as listed in the application section. There are four possible flags that can be used which affect the behaviour of the login modules. Each login module configuration is terminated by a semi-colon ';':

The flags for the LoginModule are as follows:

FLAG	DESCRIPTION
------	-------------

Required	The LoginModule is required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.
Requisite	The LoginModule is required to succeed. If it succeeds, authentication continues down the LoginModule list. If it fails, control immediately returns to the application (authentication does not proceed down the LoginModule list).
Sufficient	The LoginModule is not required to succeed. If it does succeed, control immediately returns to the application (authentication does not proceed down the LoginModule list). If it fails, authentication
Optional	The LoginModule is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

A. XmlUsersFile Configuration

This is a basic configuration using the XmlUsersFile authentication module. It authenticates users against the fedora-users.xml file that ships with Fedora as the default authentication mechanism.

```
fedora-auth {
    fedora.server.jaas.auth.module.XmlUsersFileModule required;
};
```

B. LDAP Configuration

When using LDAP authentication there are typically three basic configurations that cover most LDAP deployments.

1. DIRECT BIND

This configuration provides direct binding to an LDAP server for authentication.

```
fedora-auth {
    fedora.server.jaas.auth.module.LdapModule required
    host.url="ldap://directory.example.org"
    auth.type="simple"
    bind.mode="bind"
    bind.filter="uid={0},ou=people,dc=example,dc=org";
};
```

2. BIND-SEARCH-COMPARE

Some LDAP configurations have a 'binduser' that has access to people objects and their one-way encrypted passwords. This configuration allows the 'binduser' to connect to an LDAP server, search for the user object based on the users entered credentials, extract the users password from the user object and identify the encryption scheme used. Using this encryption scheme, the provided user password is then also encrypted and the results compared. A match results in successful authentication.

continues down the LoginModule list.

Optional

The LoginModule is not required to succeed. If it succeeds or fails, authentication still continues to proceed down the LoginModule list.

```
fedora-auth {
    fedora.server.jaas.auth.module.LdapModule required
    host.url="ldap://directory.example.org"
    auth.type="simple"
    bind.mode="bind-search-compare"
    bind.user="uid=binduser,ou=people,dc=example,dc=org"
    bind.pass="somepassword"
    search.base="ou=people,dc=example,dc=org"
    search.filter="(uid={0})"
    attrs.fetch="cn,sn,mail,displayName,carLicense";
};
```

3. BIND-SEARCH-BIND

This configuration is almost identical to the bind-search-compare strategy except that instead of finding and comparing the passwords, once a user object is found a bind is executed using that user and the provided password. This configuration is particularly useful for authenticating against Active Directory where user passwords are available from the initial bind and search.

```

fedora-auth {
  fedora.server.jaas.auth.module.LdapModule required
  host.url="ldap://directory.example.org"
  auth.type="simple"
  bind.mode="bind-search-bind"
  bind.user="uid=binduser,ou=people,dc=example,dc=org"
  bind.pass="somepassword"
  search.base="ou=people,dc=example,dc=org"
  search.filter="(uid={0})"
  attrs.fetch="cn,sn,mail,displayName,carLicense";
};

```

C. Cascading Multiple Authentication Mechanisms

Occasionally, it might be useful to authenticate from multiple sources. You might want to authenticate off an LDAP directory, but have a couple of extra users with admin privileges in the fedora-users.xml file. This is in fact recommended as you can then still get access to your repository in the event of a network failure to your LDAP directory. You also might want to authenticate users from multiple LDAP directories, or any other combination. Achieving this with JAAS is trivial.

The example below demonstrates authenticating first off an LDAP server using a direct bind. If this fails, control is passed on to the XmlUsersFile module to attempt to authenticate with the users credentials there. Note the flags for these modules are set to 'sufficient'. This means that only one of these modules needs to successfully authenticate a user for authentication to be successful.

```

fedora-auth {
  fedora.server.jaas.auth.module.LdapModule sufficient
  host.url="ldap://directory.example.org"
  auth.type="simple"
  bind.mode="bind"
  bind.filter="uid={0},ou=people,dc=example,dc=org";

  fedora.server.jaas.auth.module.XmlUsersFileModule sufficient;
};

```

UserServlet

A servlet is provided that produces an XML representation of the authenticated user, along with their attributes. The XML format produced is similar to that of the fedora-users.xml file and is structured as follows:

```

<user id="userid">
  <attribute name="attributename1">
    <value>value1</value>
    <value>value2</value>
  </attribute>
  <attribute name="attributename2">
    <value>value1</value>
  </attribute>
</user>

```

This servlet is configured by default to be accessible via <http://server:port/fedora/user> and should always be protected by the JAAS authentication filter.

The purpose of this servlet is to provide applications the ability to access user attributes and authenticate users without the need to implement security infrastructure on the application end. For example, attributes such as email and display names can be obtained by applications without them having to configure and use an LDAP based data/authentication store. This also means that the authentication is handled at a single point rather than at the Fedora end and the application end. This will allow easier development of front end applications and remove the duplication of security infrastructure. It also means that the authentication mechanisms are client system agnostic.

FeSL Authorization

FeSL Authorization

FeSL Authorization is based on XACML version 2. XACML policies are stored in the Fedora repository as FESLPOLICY datastreams on Fedora objects. These datastreams can be either inline XML ("X") or managed content ("M").

A set of bootstrap system policy objects are created when Fedora first starts, from the policies in the `$FEDORA_HOME/pdp/policies` directory. If you need to amend any of these bootstrap policies you will need to edit the Fedora objects created. These objects have a `fedora-policy PID` namespace.

Configuration

Policy evaluation results caching.

Results of previous policy evaluations are cached. This can mean that if you update an existing policy, an entry already stored in the cache from a previous request may be returned rather than results based on the evaluation of the updated policy.

If you wish to disable caching, set the environment variable:

```
PEP_NOCACHE=true
```

Otherwise it is currently necessary to restart Fedora to clear the policy evaluation cache.

Configuration files

Configuration files are located in `FEDORA_HOME/pdp/conf`

- `config-pdp.xml`
This is the dynamic configuration file for the Sun XACML PDP Evaluation Engine. This file is used to register PolicyFinder modules, AttributeFinder modules and ResourceFinder modules
- `config-policy-manager.xml`
This file configures the Policy Manager. This component specifies which PolicyDataIndex (component that indexes/searches/retrieves policies) to use. The default one that is used with the FeSL PDP is the DbXmlPolicyDataIndex. This uses Oracle DBXML as a back-end policy management system. In addition, this configuration file is used to specify which policy combination algorithm to use when multiple policies are retrieved.
- `config-dbxml.xml`
These settings are for the Oracle DBXML database. They specify where the database is to be located and what it is to be called. It also specifies whether to validate policies against a schema when they are being added or not. In addition you can specify a custom index map. This lets you select which attributes in an XACML Policy Target you are going to be indexing your policies on.
- `config-pdm-fedora.xml`
This file contains settings for how Fedora policy objects are created through a PolicyDataManager - however the methods of this are not externally exposed, and so are only relevant to testing.
- `config-attribute-finder.xml`
~~This config file is for the Fedora RISEarch Attribute finder. When policies need additional information from Fedora that was not provided in the XACML Request, they can be retrieved by custom AttributeFinders. This module uses the Fedora RISEarch interface to fetch additional attributes for policies. You can specify which attributes to look for in this file as well as what URL, username and password to use for the RISEarch.~~

XACML Policy Enforcement

1. Introduction

A major feature of the Fedora security architecture is the [eXtensible Access Control Markup Language](#) (XACML) and an XACML-based policy enforcement module. Developed by the OASIS Consortium, XACML is an XML-based markup language to encode access control policies. The policy language is flexible and enables the specification of fine-grained, machine-readable policies that can be used to control access to Fedora web services, Fedora digital objects, datastreams, disseminations, and more. Since a policy is only worth its salt if it can be enforced, Fedora 2.1 introduces a new Authorization module implemented as part of the core Fedora repository service. The Authorization module is built upon the [Sun XACML engine](#). Each XACML policy defines: (1) a "target" describes what the policy applies to (by referring to attributes of users, operations, objects, datastreams, dates, and more), and (2) one or more "rules" to permit or deny access. There is a [Fedora-specific policy vocabulary](#) for referring Fedora operations and Fedora-specific entities within XACML policies. Regarding policies, Fedora supports both repository-wide policies (that specify broad access controls that apply to the entire repository), and object-specific policies (that specify rules for a single object, and can even be stored within the object in a special datastream). Fedora 2.1 comes out-of-the-box with a set of [default repository-wide policies](#) that establish baseline access controls that are equivalent to what was provided in Fedora 2.0. These default policies restrict access to the Fedora management web service (API-M) to only the Fedora administrator, permit open access to the Fedora access web service (API-A), and establish some other basic controls (e.g., allow access to API-M only from localhost; restrict policy management to administrator). In addition to the default policies (which can be modified), any number of custom XACML policies can be written and loaded into Fedora. For assistance in creating new policies for your repository and for your objects, see the [Fedora XACML Policy Writing Guide](#).

This guide is intended to give a general overview of XACML-based Policy Enforcement Module to support authorization in Fedora repositories. This guide provides Instructions on how to configure XACML-based policy authorization in Fedora as well as a discussion of how the Fedora Policy Enforcement module works. ** For more information consult the following sources:

Fedora Security Architecture

[Securing Your Fedora Repository](#): documentation on security options and configuring user authentication sources for Fedora repositories

[Fedora XACML Policy Writing Guide](#): documentation for details on writing XACML policies for Fedora repositories

OASIS (for policy writers)

[OASIS XACML Specification](#): this is the official specification and a good reference document

[A Brief Introduction to XACML](#): this is nice introduction to the XACML concepts

[XACML Technical Committee](#): this home page of the technical committee provides access to other documents on XACML

SUN (for developers)

[Sun XACML Home Page](#): general information on the Sun Java reference implementation of XACML

[Sun XACML Programmer Guide](#): technical details of the reference implementation

[Sun XACML Javadocs](#): interface definitions of the reference implementation

2. Configuring the XACML Authorization Module

Note that to do identity-based policies (user login id or user attributes), you must have authentication configured. Run `fedora-setup` to choose which Fedora service interfaces will be configured for authentication. If a policy is written that makes reference to required (`MustBePresent="true"`, either explicitly or by default) user identity or attributes (`Subject attributes in Policy target`) and authentication has not been configured, the policy will be evaluated as "indeterminate" and the service request will fail with an authorization exception. Be sure to enable authentication for API-A if you intend to write policies for accessing objects based on user identity/attributes.

By default, Fedora XACML-based authorization is enabled. Configuration of the Fedora XACML-based Policy Enforcement Module is done in the Fedora server configuration file (`fedora.fcfg`). Depicted below is the section of the configuration file for the Authorization module that controls XACML-based policy enforcement.

```
<module role="fedora.server.security.Authorization"
  class="fedora.server.security.DefaultAuthorization">
  <comment>Builds and manages Fedora's authorization structure.</comment>
  <param name="REPOSITORY-POLICIES-DIRECTORY"
    value="/fedora-xacml-policies/repository-policies"/>
  <param name="XACML-COMBINING-ALGORITHM"
    value="com.sun.xacml.combine.OrderedDenyOverridesPolicyAlg"/>
  <param name="ENFORCE-MODE" value="enforce-policies"/>
  <param name="POLICY-SCHEMA-PATH" value="xsd/cs-xacml-schema-policy-01.xsd"/>
  <param name="VALIDATE-REPOSITORY-POLICIES" value="true"/>
  <param name="VALIDATE-OBJECT-POLICIES-FROM-DATASTREAM" value="false"/>
  <param name="OWNER-ID-SEPARATOR" value=","/>
</module>
```

2.1 Enabling/Disabling XACML Policy Enforcement

To enable/disable XACML policy enforcement in Fedora, use the Fedora configuration file (`fedora.fcfg`). Whether Fedora uses XACML for authorization decisions is controlled by the `ENFORCE-MODE` parameter in the Authorization module:

```
<param name="ENFORCE-MODE" value="enforce-policies"/>
```

The `ENFORCE-MODE` parameter can contain one of three values, with the following meanings:

1. `enforce-policies` – enable XACML enforcement to determine whether a request is permitted or denied
2. `permit-all-requests` – disable XACML enforcement; PERMIT every request by default
3. `deny-all-requests` – disable XACML enforcement; DENY every request by default

The `enforce-policies` setting is used to enable the enforcement of XACML policies, and is the default setting for a Fedora repository. The `permit-all-requests` setting can facilitate testing code independent of security. The `deny-all-requests` setting can be used to quickly shut down access to the server, but requires a server restart to affect this.

Tomcat container security is, of course, still a first barrier to authentication/authorization (i.e., Fedora's Tomcat web.xml specifies access protection earlier than XACML. Tomcat container security is always in place regardless of the setting for parameter `ENFORCE-MODE`.

2.2 Configuring the Storage Locations of Policies

Active policies are those policies that are "in play" for the repository. There are two places that active policies can be stored: (a) in a file system

location configured for the repository, or (b) in digital objects within the special "POLICY" datastream. It should be noted that Fedora comes out-of-the-box with a set of default repository-wide policies. In addition to these default policies, it is possible to create any number of custom XACML policies for a repository and for specific digital objects. Consult the [Fedora XACML Policy Writing Guide](#) for instructions on authoring new policies and a set of sample policies for Fedora.

2.2.1 Storing Repository-Wide Policies in a Configured Location

Repository-wide policies are broad policies that are intended to be in play for the entire Fedora repository. By saying that these policies are *broad* does not mean they must be course-grained. Repository-wide policies can be fine-grained and they can be written to control access to any Fedora API operation, to groups of digital objects, or even to sets of specifically identified digital objects. Repository-wide policies are distinguished from object-specific policies (described below) in that **they are stored in memory when the server is started and they are evaluated for their applicability on every Fedora service request.**

In the Authorization module section of the Fedora server configuration file (`fedora.fcfg`), there are parameters to set the storage location for active repository-wide policies. **The Fedora repository will only know about policies that are stored within the configured policy directory location.** Policies stored in any other location will be invisible to the repository and will not be loaded into the repository upon server startup! The storage location for active repository-wide policies can be set by the repository administrator using the follow parameter in Authorization module (in `fedora.fcfg`):

```
<param name="REPOSITORY-POLICIES-DIRECTORY" value="/fedora-xacml-policies/repository-policies" />
```

The configuration parameters in the Authorization module provide the repository administrator with a choice as to where XACML repository-wide policy files are stored. If you keep the default path value, then the repository-wide policy directory will be created at: `FEDORA_HOME/data/fedora-xacml-policies/repository-policies`. You can override this default with either a fully-qualified directory path or a relative directory path of your choice. The policy storage location work just like the digital object and datastream storage locations that are also configured for the server (in `fedora.fcfg`). It is expected that repository administrators appropriately protect these directories so that they cannot be tampered with.

Default Repository-Wide Policies:

It should be noted that Fedora comes out-of-the-box with a set of default repository-wide policies. The default policies are automatically copied into the active policies storage location the first time the Fedora repository server is started (i.e., copied into a subdirectory named `/default`). It is recommended that changes to the default repository-wide policies are made with extreme care. These policies establish the baseline authorization rules for Fedora. Refer to the documentation below on [default repository policies](#) for a description of what each policy does.

NOTE! Any changes to the default policies or your own custom policies should be made in the `/default` subdirectory of the configured location for repository-wide policies. The original versions of the default policies are also maintained in the internal fedora staging directory named `"FEDORA_HOME/server/fedora-internal-use/"` from which the default policies are copied into the configured location. **Do not edit the policies that are stored in this internal staging directory!** They are your record of how the default policies shipped with Fedora.

Custom Repository-Wide Policies:

You can create your own custom repository-wide policies and put them in the repository-wide policy directory. You can also create your own subdirectories under the repository-wide directory to organize your policies. It is recommended that you do NOT put your custom policies in the `/default` subdirectory that was created by the Fedora server. This location is reserved for the policies that are distributed with Fedora. Consult the [Fedora XACML Policy Writing Guide](#) for instructions on authoring new custom policies and to view a set of sample policies for Fedora.

Custom repository-wide policies can be written to deny/permit access to Fedora API operations, to control access to groups of objects, or to control access to objects with certain attributes under certain conditions. It should be noted that custom repository-wide policies can be also written to address a specific digital objects (identified by PID). This is in contrast to storing or referencing an object-specific policy inside the digital object's POLICY datastream (described below). The disadvantage of putting an object-specific policy in the repository-policies storage location is that the policy is unnecessarily evaluated for requests that do not pertain to the specific digital object it is about. This is because repository-wide policies are stored in memory and are evaluated for every Fedora API request. As described below, there are better places to store a policy that pertains to a single digital object (or a small set of named digital objects).

2.2.2 Storing Object-Specific Policies in a POLICY Datastream

An object-specific policy can be stored inside a digital object within the special reserved datastream whose ID is "POLICY". There are several benefits to storing object-specific policies in the POLICY datastream. First, these policies are only evaluated for those Fedora API requests that pertain to the digital object in which the policy "resides." Policies that are stored in the POLICY datastream are not loaded into memory upon startup of the Fedora server (as are the repository-wide policies); therefore, the POLICY datastream approach may be a way to enhance performance in cases where a large number of object-specific policies will exist. Another benefit is that the POLICY datastream storage strategy may provide for easier distribution of policy management responsibilities. For example, authors or owners of particular digital objects can be granted the rights to view and modify the POLICY datastream of their objects, without having to obtain repository administrator privileges to modify policies in a configured policy storage locations external to a repository.

Unlike when an object-specific policy is placed in the repository-wide policies directory, it is not necessary to specify the an object PID in the target of the XACML policy target when the policy resides in a POLICY datastream. The Fedora system will automatically make this association during policy enforcement.

Policies as stored as Inline-XML (X) or Managed-Content (M) Datastreams: One benefit of putting an object-specific policy inside a digital object as a type X or M datastream is that the policy is stored within the either in the object's XML wrapper file (i.e., FOXML), or it is stored in the managed content datastream area of the repository. In either case, the POLICY file itself is under the direct custodianship of the Fedora

repository. As such, these policies are managed like any other X or M datastreams, and the policies become **portable** with their parent digital objects (e.g., the policies go with an object, as datastream content, when the object is exported from the repository).

Policies as Externally Referenced (E) or Redirected (R) Datastreams: One benefit of putting object-specific policies in the POLICY datastream in a by-reference fashion is that multiple digital objects can point to the same policy. In this case, you probably do not want to have the target of the XACML policy make reference particular digital object PIDs. In this scenario, you can store the policy external to the digital objects to which it pertains, and store the URL of the external policy location in the E or R type datastream. Fedora resolves such URLs at runtime, so when the policy enforcement module needs to evaluate the XACML policy, Fedora will automatically retrieve the policy file from its external location and pass it on to the policy enforcement module for evaluation.

2.3 Activating and Loading Repository-Wide Policies

Repository-wide policies are not considered **active** unless they are placed in the configured storage location specified in the Authorization module configuration (see `fedora.fcfg`). Once policies are placed either directly in this directory, or within subdirectories under it, they are able to be **loaded** by the Fedora repository server. To put new custom policies into play, simply add them to the configured repository-wide directory, or subdirectories within it. To inactivate a policy, remove it from the directory. To modify an existing policy, edit the policy in the configured policy directories (or preferably, edit in another location and replace the existing policy in the configured policy directory).

To activate and load repository-wide policies take the following steps:

1. **Validate** your policies using the `validate-policy` command line utility (*optional, but recommended!*)
2. **Activate** your policies by copying them into the configured storage location for repository-wide policies (configured in the Authorization module in `fedora.fcfg`)
 - **REPOSITORY-POLICIES-DIRECTORY** – put your custom repository-wide policies in any sub-directory under this configured directory
3. **Load** your policies by starting the Fedora server. If the server is already started, run the `fedora-reload-policies` command line utility.

All repository-wide policies are loaded into the Fedora server's memory every time the server is started. For obvious performance reasons, object-specific policies stored in the POLICY datastreams of digital objects are not pre-loaded; instead they are loaded on an as-needed basis (i.e., when a request for that object is made).

WARNING: It should be noted that the **_first time_** the Fedora server is started, the **_default_** repository-wide policies are copied into a subdirectory named `/default` under the repository-wide policy storage location. To make sure this happens correctly, you should initially start the server with the default value for the server hostname (see `fedoraServerHost` in `fedora.fcfg`). The default is `127.0.0.1` (or `localhost`). This initial starting of the server as `localhost` is necessary to make sure that the default XACML policies are properly configured and placed in the repository-wide policy directory. Once the default policies are in place, you can shutdown the server and change the hostname at any time. If you subsequently change the hostname to something other than `localhost` you must manually update some of the default policies to reflect the fixed IP address of your server. In the following default policies, you will see the loopback IP address of `127.0.0.1` (and `::1` for IPv6) in the policy condition:

1. `deny-apim-if-not-localhost.xml`
2. `deny-reloadPolicies-if-not-localhost.xml`
3. `deny-serverShutdown-if-not-localhost.xml`

Simply add the new IP address to the policy rule as follows:

```
<Rule RuleId="1" Effect="Deny">
  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:or">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
        <EnvironmentAttributeDesignator
          AttributeId="urn:fedora:names:fedora:2.1:environment:httpRequest:clientIpAddress"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
          <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">127.0.0.1</AttributeValue>
          <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">::1</AttributeValue>
          <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">111.22.333.4</AttributeValue>
        </Apply>
      </Apply>
    </Apply>
  </Condition>
</Rule>
```

2.4 Enabling/Disabling Policy Validation

There are several parameters in the Authorization module configuration that control whether Fedora will attempt to validate policies against the XML schema for XACML.


```
<param name="POLICY-SCHEMA-PATH" value="xsd/cs-xacml-schema-policy-01.xsd"
<param name="VALIDATE-REPOSITORY-POLICIES" value="true"/>
<param name="VALIDATE-OBJECT-POLICIES-FROM-DATASTREAM" value="false"/>
<param name="VALIDATE-SURROGATE-POLICIES" value="false"/>
```

By default all repository-wide policies will be validated by the repository when they are loaded. In general it is recommended that all new policies are validated before they are put into the active policy location. Use the `validate-policy` command line utility to validate an XACML policy file. A policy failing validation would result in the following situation, according to the stated case.

- repository policy: authorization module failure at server startup time, server not properly started

Object policies are not set to validate by default because they are loaded dynamically at each use, and neither testing to prove adequate performance nor caching to ensure it has been done. This policy validation check can be enabled by the appropriate `fedora.fcgi` setting.

2.5 Configuring Multiple Owner IDs

The `OWNER-ID-SEPARATOR` value specifies the delimiter used between multiple values within the `ownerId` property of the objects stored in the repository. It is specified as a regular expression, and the default value is `,` (comma). This delimiter allows the use of policies that act on objects with multiple possible `ownerId` values. For more information, see the sample policies in the [Fedora XACML Policy Writing Guide](#).

2.6 Surrogate Feature No Longer Requires Policies.

This is an **experimental feature** continued with Fedora 2.2 that enables a higher level application or service (e.g., a Web front end, middleware, or other web service) to authenticate to Fedora with its own identity, but pass along the identity of an end-user that authenticated with the service. The higher level application or service is thus acting as a "surrogate" for the ultimate end user. The surrogate user can pass through to Fedora the identity of the end-user it is representing in the HTTP request header (using the `From:` attribute). If Fedora authenticates the identity of the service-as-surrogate, then the identity of the ultimate end user can be used in XACML policies.

Refer to the [Authentication and User Attributes](#) section of the [Securing Your Fedora Repository](#) for information on configuring servlet filters in support of the surrogate feature.

3. Implementation of the Fedora Policy Enforcement Module

3.1 Policy Determination Point (PDP) and Policy Enforcement Point (PEP)

According to the OASIS XACML specification, an application functions in the role of the Policy Enforcement Point (PEP) if it guards access to a set of resources and asks the Policy Determination Point (PDP) for an authorization decision. The PEP MUST abide by the authorization decision in the following way: A PEP SHALL allow access to the resource only if a valid XACML response of Permit is returned by the PDP. The PEP SHALL deny access to the resource in all other cases. An XACML response of Permit SHALL be considered valid only if the PEP understands all of the obligations contained in the response.

The Fedora Policy Enforcement Module fulfills the responsibilities of both the PEP, except that for 2.1, obligations are ignored. (But neither are any coded in the default policies.) The Fedora module wraps the Sun XACML implementation of the PDP. Fedora module also implements custom attribute finders and a custom PEP.

The PDP determines the set of policies that are applicable to any given Fedora service request. Remember the PDP determines whether a policy is applicable by comparing the Subject/Resource/Action attribute designations in a Policy Target to the context of an incoming Fedora service request (i.e., attributes that describe the Fedora service request, the user/subject, the desired object/datastream/dissemination, and the runtime environment). Repository-wide policies are always in play and will be evaluated by the PDP to determine whether they are applicable to the particular incoming Fedora service request. Object-specific policies if the incoming request refers to an object by its PID and there exists an object-specific policy mentioning that PID. ***PDP makes the decision of deny/permit/indeterminate, and then the PEP makes sure to enforce this decision for the incoming Fedora API request.***

NOTE: To quiet the Sun XACML engine's INFO messages, a `logging.properties` file is included with the Fedora distribution, edited to print only SEVERE messages to the console. Fedora then starts Java using that `logging.properties` file. The original file, as distributed with JDK 1.4.2.8, prints INFO (and more severe) messages, making the console a bit chatty. To revert to the original logging behavior, edit `fedora-start.bat` (Windows) or `fedora.sh` (Unix), and remove the `logging.properties` from the `java vm` arguments (see `-D` flags).

3.2 Understanding the XACML Policy Combining Algorithm

Policy writers must understand the interaction effect of multiple XACML policies that are in scope for any particular action. The Fedora configuration file (`fedora.fcgi`) sets the policy combining algorithm that will be used by the Policy Enforcement Module in evaluating sets of policies. In the the Fedora server configuration file (`fedora.fcgi`) note the following parameter in the section for the Authorization module::

```
<param name="XACML-COMBINING-ALGORITHM" value="com.sun.xacml.combine.OrderedDenyOverridesPolicyAlg"/>
```

This parameter sets the XACML policy combining algorithm that controls how the Fedora Policy Enforcement Module will deal with multiple policies that may be applicable to a Fedora service request. The default value in Fedora is the Ordered Deny Overrides policy combining

algorithm. It allows a single evaluation of **deny to take precedence** over any number of permit, not applicable or indeterminate results. Note that this uses the regular Deny Overrides implementation since it is also ordered. Consult the OASIS and Sun XACML documentation for a description of alternative combining algorithms. Note that the default policies assume and require this algorithm. Generally policies are developed with an algorithm in mind.

In the PDP, policies are matched based on their applicability to an incoming service request. All policies that are applicable are combined programmatically and dynamically per request into a PolicySet. To estimate the number of policies that may be in the PolicySet for a given service request, consider N to be the number of policies configured in Fedora repository-wide policy storage location. Then we have the following possible number of policies in a PolicySet for the PDP to consider:

N :	# of policies if a service request does not refer to a particular digital object
N :	# of policies if a service request refers to an object, but there is no object-specific policy for that object
N+1:	# of policies if a service request refers to an object that has an object-specific policy datastream, but has no object-specific policy in the object policies directory
N+1:	# of policies if a service request refers to an object that has an object-specific policy in the object policies directory, but has no object-specific policy datastream
N+2:	# of policies if a service request refers to an object that has an object-specific policy datastream and a policy in the object-policies directory

3.3 A Simplified Understanding of the Authorization Decision

For an incoming service request to succeed, there must be an explicit permit and the absence of a deny; the absence of a deny is not enough to permit an action. By default, if any of the applicable policies in a Policy Set yield a deny, the requesting subject will be denied access, even if some other policy permitted the action. In other words, deny will prevail over permit. Also, if there is a policy in the set that is evaluated as "Indeterminate," then the result of that policy evaluation will be considered a deny. A policy can be evaluated as Indeterminate if there was an error during policy evaluation. Also, a policy can be evaluated as Indeterminate if there is a required attribute specified in the policy that did not exist in the context of the incoming requests. See the section of [Required vs. Optional Attributes](#) for more details.

For the purposes of a simple understanding of the Fedora Policy Enforcement module, things work like this:

NOTE:	=0 means NO policy in a policy set evaluated to that result
	=1 means one or more policies in a policy set evaluated to that result
===== POLICY SET RESULTS =====	=== FINAL DECISION ===
DENY=0 INDETERMINATE=0 PERMIT=0	result is DENY
DENY=1 INDETERMINATE=0 PERMIT=0	result is DENY
DENY=1 INDETERMINATE=0 PERMIT=1	result is DENY (denial trumps permit)
DENY=0 INDETERMINATE=0 PERMIT=1	result is PERMIT
DENY=0 INDETERMINATE=1 PERMIT=1	result is DENY (indeterminate is treated as denial which trumps permit)
DENY=0 INDETERMINATE=1 PERMIT=0	result is DENY

Put another way... assuming the default policy combining algorithm for Fedora is "Deny Overrides", an action is Permitted or Denied depending on the evaluation of the various policies, as follows:

Permit requires ALL of the following conditions to be TRUE:

- at least one policy was evaluated to Permit the action
- NO policy must evaluate to explicitly Deny the action
- NO policy must evaluate as Indeterminate for the action
- NO error or unknown result is returned by the Sun XACML engine

Deny only requires ONE of the following conditions to be TRUE:

- at least one policy was evaluated to explicitly Deny the action
- at least one policy was evaluated to be Indeterminate for the action
- the Sun XACML engine returned a unknown result (an error or a return value that is not in the XACML specification)

3.4 PDP Implementation Details

We can understand the results of the PDP's evaluation of a policy set from three perspectives: (1) the Sun XACML engine, (2) the Fedora

wrapper of Sun XACML, and (3) the bottom line outcome.

(1) PDP (Sun XACML engine perspective):

The Sun XACML engine, which underlies the Fedora Policy Enforcement Module, will evaluate a Policy Set and return a decision. Refer to the [OASIS XACML Specification](#) and the [Sun XACML documentation](#) for details.

In making its authorization decision, the Sun XACML engine will return a single result from its evaluation of a Policy. The result will be one of the following:

- **Permit** – returned if a policy rule was applicable and thus it returned its permit effect.
- **Deny** – returned if a policy rule was applicable and thus returned its denial effect.
- **Indeterminate** – returned if an attribute value that was needed to evaluate a rule could not be found, or another error prevented processing.
- **NotApplicable** – returned if no rule applied and so no effect could be returned.

Given the default policy combining algorithm of "Ordered Deny Overrides," the PDP will make its final decision for a policy set such that * DENY* will prevail over PERMIT. If one or more policies evaluate to Deny, Sun XACML gives a verdict of DENY. If one or more policies evaluate to Indeterminate, and no policies evaluate to Deny, the verdict is INDETERMINATE. If one or more policies evaluate to Permit, and no policies evaluate to either Deny or Indeterminate, the verdict is PERMIT. If one or more policies evaluate to NotApplicable, and no policies evaluate to Permit or Deny or Indeterminate, the verdict is NOTAPPLICABLE.

(2) PEP (Fedora wrapper perspective):

The Fedora wrapper respects the verdict of the Sun XACML PDP, which should usually be PERMIT or DENY, given our default set of policies. But, for safety, the Fedora wrapper code imposes a DENY result in any of several extraordinary cases: (1) if somehow Sun XACML returned an unexpected final result of Indeterminate or NotApplicable, perhaps due to a policy written incorrectly, (2) if Sun XACML returned no result at all, (3) if Sun XACML returned a result which is not defined by OASIS XACML or Sun XACML standards (the sunxacml Java interface uses int to code the results, so sunxacml could return bad results), or (4) if authorization processing results in an exception being thrown.

3) Combined (Bottom line perspective):

As a rule, a policy set evaluates to PERMIT when at least one policy in the set evaluates to Permit and no policies evaluating to Deny or Indeterminate. Otherwise, the policy set evaluates to DENY.

Specifically:

- a) DENY occurs when there is at least one policy that evaluates to Deny or an Indeterminate.
- b) DENY occurs if no policy evaluates to Permit. A DENY also occurs in several exceptional situations:
 - a) DENY occurs when no policies were found to be applicable (all evaluate to NotApplicable) or the related case of there being no policies at all configured with Fedora
 - b) DENY occurs when errors occurred during authorization processing, including no or bad results obtained.

3.5 Fedora PEP Implementation Details

Fedora's Policy Enforcement Point (PEP) builds a minimal request for the Sun XACML engine to evaluate. One job of the Fedora PEP is to gather up all of the Subject/Resource/Action/Environment attributes that are relevant for an incoming service requests. The values of this attributes are the key to determining what policies are in scope for an incoming service request. To gather up all relevant attributes, the Fedora PEP has two custom "attribute finder" modules that interact with the Sun XACML engine.

- The **ContextAttributeFinderModule** has the job of obtaining attributes that are stored in the enhanced Fedora Context object that is associated with an incoming Fedora service request. Attributes that originate with an incoming service requests can be Subject attributes (i.e., attributes of the requesting user/agent), Action attributes (i.e., the identity of the Fedora API operation that is the basis of the request), Resource attributes (i.e., attributes that identify specific objects/datastreams/disseminations that are being requested, and Environment attributes (i.e., attributes that describe the runtime environment of the incoming request, like the current date/time or the server IP address). It will honor a callback, even for an attribute which hasn't been explicitly coded, so can provide arbitrary attributes (e.g., from LDAP lookup in a JAAS login module, from Shibboleth via a servlet filter). [There are a few attributes which it explicitly doesn't serve, to prevent stack overflow on improper recursion, or because the attributes are known to be provided in the xacml request itself.]
- The **ResourceAttributeFinderModule** has the job of obtaining all relevant attributes about Fedora resources (i.e., digital objects, datastreams, and disseminations) for resources that are in scope for an incoming request. While a few attributes of a resource are picked up from the incoming request itself (typically, the identity of a resource like a PID or a datastream Id), the ResourceAttributeFinderModule gets all other attributes about such resources by introspecting on actual digital objects in the repository.
- The ContextAttributeFinderModule supplies environment values, named into the Fedora XACML URN namespace. An **EnvironmentAttributeFinderModule** may be added for full compliance with XACML requirement that environment attributes be serviced which are named into its namespace XACML's namespace itself.

4. How to Bind User Attributes into the Fedora Policy Enforcement Module

The availability of user identity and attributes depends on the authentication configuration option selected for the repository. Refer to the

[Authentication and User Attributes](#) guide for information on authentication configuration options. In terms of understanding what user attributes can be referenced for Subjects in an XACML policy, you must first know what the sources of authentication information are for the repository. The Fedora XACML-based Policy Enforcement module will automatically be able to obtain attributes from one or more of following authentication sources when they are configured as described in [Authentication and User Attributes](#).

4.1 fedora-users attributes

In Fedora, XACML policies can refer to user identity and attribute information that is specified within the `fedora-users.xml` file. This file is one of the Fedora configuration files and is created initially on running the installer. It contains username and password information about "authenticated" users. Attributes can also be assigned to user entries. These attributes are then available as attributes in related XACML policies.

By default, the `fedora-users` file contains the following users, passwords, and attributes:

```
<users>
  <user name="fedoraAdmin" password="fedoraAdmin">
    <attribute name="fedoraRole">
      <value>administrator</value>
    </attribute>
  </user>
  <user name="fedoraIntCallUser" password="changeme">
    <attribute name="fedoraRole">
      <value>fedoraInternalCall-1</value>
      <value>fedoraInternalCall-2</value>
    </attribute>
  </user>
</users>
```

Notice that each attribute has a name and can have multiple values. The cases of one or two values are shown; not shown here is that an attribute can have any number of values, including none, depending on your need. The second of these entries is for Fedora internal use, and should be left intact as installed. You will probably leave the first entry in place, also as-is. But if you change the default policies, you may require additional values of the *fedoraRole* attribute, or additional named attributes with their own values. Additional users (with unique names) can be added with attributes as needed. This is a convenient place to define repository managers, a small number of users, or a surrogate user. An example of an additional user follows:

```
<users>
  <user name="testuser1" password="testuser1">
    <attribute name="someAttribute">
      <value>xyz</value>
    </attribute>
    <attribute name="fedoraRole">
      <value>researcher</value>
    </attribute>
  </user>
</users>
```

The above example indicates that the user has two attributes (an attribute named "someAttribute" whose value is `xyz`, and an attribute named "fedoraRole" whose value is `researcher`). These attribute names can be used in SubjectAttributeDesignator specifications in XACML policies and the values can be used in AttributeValue specifications in policies. Below is a snippet of an XACML policy that refers to attributes from `fedora-users` (refer to the [Fedora XACML Policy Guide](#) for more info on policy syntax):

```
<Subjects>
  <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        administrator
      </AttributeValue>
      <SubjectAttributeDesignator AttributeId="*fedoraRole*" MustBePresent="false"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </SubjectMatch>
  </Subject>
</Subjects>
```

4.2 LDAP attributes

In Fedora, XACML policies can refer to user attribute names and values that are registered in an LDAP that is configured with Fedora's Ldap servlet filter. Refer to the [Authentication and User Attributes](#) section of the [Securing Your Repository](#) guide for information on servlet filter configuration options. Given that an LDAP is properly configured with Fedora, the Fedora XACML-based Policy Enforcement module will be able

to access LDAP user attributes, which means that you can refer to LDAP attributes in reference to a Subject in a policy, as shown in the following XACML snippet (refer to the [Fedora XACML Policy Guide](#) for more info on policy syntax):

```
<Rule RuleId="1" Effect="Deny">
  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
    <SubjectAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
      AttributeId="*ou*" />
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        Lb-Info Technology
      </AttributeValue>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        Lb-Univ Librarian-General
      </AttributeValue>
    </Apply>
  </Condition>
</Rule>
```

In the above example, the SubjectAttributeDesignator refers to an LDAP attribute name ("ou"), which refers to the university organizational department that a user belongs to. The policy rule applies a function to set up the condition that the value of the "ou" attribute must be one of the listed values ("Lb-Info Technology" or Lb-Univ Librarian-General").

4.3 Shibboleth attributes via an HTTP Servlet Filter

There are cases when an application may obtain authenticated user attributes in an application or service layer outside the context of the Fedora repository service. Fedora provides a simple means of getting these attributes into the Fedora repository service so they can be used by the Fedora XACML policy enforcement module. To support the ability for upstream applications or services to send these user attributes into a repository, Fedora will recognize a special HTTP servlet request attribute named after the value of static final String constant `fedora.server.Context.FEDORA_AUX_SUBJECT_ATTRIBUTES`.

The Fedora repository service will automatically look for an HTTP servlet request attribute named after this constant. The Fedora code now takes a request attribute found under that name, as a Map giving name and values of subject attributes. Currently, name must be a String and this is unlikely to change. Value must be a String, and later this might be relaxed to include String[], to allow attributes with multiple values. Other types of value are not serviced. The effect of having a key => value pair "a" => "b" in the Map is the same as having a Tomcat role "a=b", with the exception that the effect of having the same attribute key redefined both places is right now undefined. So your servlet filter needs only create the map and populate it, and put it into an HTTP servlet request as attribute named after `fedora.server.Context.FEDORA_AUX_SUBJECT_ATTRIBUTES`. Fedora will then look for it, and use the attributes in XACML-based authorization.

This approach was initially developed to support the OhioLink [Shibboleth servlet filter](#) that will be available to the Fedora community from the OhioLink implementation. Although full Shibboleth integration with Fedora will be in future releases of Fedora, the ability to send attributes into Fedora via a servlet filter is a way to get started with using Shibboleth-acquired attributes in Fedora XACML policy enforcement. It should be noted that this means of getting attributes into Fedora can be used with any subject attribute source (i.e., it remains source-neutral and doesn't favor Shibboleth or any other particular scheme).

5 Default Repository Policies for Fedora

5.1 Default Access Control Policies

Out-of-the-box, the Fedora repository will have a default set of access control policies that provide for a highly restricted management service (API-M), an open access service (API-A), and an open OAI provider service. The default access control policies establish the same level of out-of-the-box security on the repository that was previously configured for Fedora 2.0 release; however, as of Fedora 2.1 these basic access controls are now specified as XACML policies. The default access control policies can be found within the Fedora software distribution in the following directory:

```
FEDORA_HOME/data/fedora-xacml-policies/repository-policies/default
```

The first time the Fedora repository server is started, these policies will be automatically copied into the official [repository-wide policy storage location](#) that was specified in the Fedora configuration file (`fedora.fcfg`). The policies are activated once they are copied into this location.

Rule	Service	XACML Policy File	Policy Description
1	any	permit-anything-to-administrator.xml	This is a "positive policy" that permits the Fedora administrator to have access to any operation on any Fedora repository service (API-M, API-A, OAI, RISearch). By default the Fedora administrator is configured in the default Tomcat user credentials file (<code>tomcat-users.xml</code>).
2	API-M	deny-apim-if-not-localhost.xml	This is a "negative policy" that denies access to API-M operations that are not made from the IP address of the machine on which the Fedora repository is running on. In other words, the policy will not allow API-M requests from hosts other than "localhost."

3	API-A	permit-apia-unrestricted.xml	This is a "positive policy" that permits unrestricted access to API-A. In other words, API-A operations are completely open for use by any user/agent.
4	OAI	permit-oai-unrestricted.xml	This is a "positive policy" that permits unrestricted access to the default OAI provider interface to the Fedora repository. In other words, OAI-PMH operations are completely open for use by any user/agent. (Note, this does not control access to the stand-alone PROAI service that is distributed with Fedora 2.1. PROAI is a stand-alone web application that must be secured separately.

A review of how the policy combining algorithm works, will reveal that access to a service operation cannot occur unless access is **expressly permitted**. The net effect of the default access control policies is that the administrator is expressly permitted to do anything (with the restriction of having to make API-M requests from the same IP address that the server runs on), and all users are expressly permitted access to API-A and OAI service requests.

5.2 Default Utility Policies

Generally, the default repository utility policies **should not be removed**. They enforce core and crucial protections of the repository. Considerate understanding of how they work should proceed any (unlikely) needed editing. For example, consider and edit to permit other IPs than localhost, as opposed simply to deleting the policy.

Rule	Service	XACML Policy File	Policy Description
1	serverAdmin	deny-policy-management-if-not-administrator.xml	
2	any	deny-inactive-or-deleted-disseminations-if-not-administrator.xml	This is a "negative policy" that will deny all access to inactive/deleted disseminations if the user/agent is not the Fedora administrator. Unlike purged disseminations, inactive/deleted disseminations still exist, but they are just marked as inactive/deleted. As such they should not be available to users. The exception is that the Fedora administrator is allowed to access them.
3	any	deny-inactive-or-deleted-objects-or-datastreams-if-not-administrator.xml	This is a "negative policy" that will deny all access to inactive/deleted datastreams if the user/agent is not the Fedora administrator. Unlike purged objects/datastreams, inactive/deleted objects/datastreams still exist, but they are just marked as inactive/deleted. As such they should not be available to users. The exception is that the Fedora administrator is allowed to access them.
4	API-M	deny-purge-datastream-if-active-or-inactive.xml	This is a "negative policy" that will ensure that datastreams cannot be purged (permanently removed) unless they are in the deleted state. Purging of active or inactive datastreams is not allowed.
5	API-M	deny-purge-object-if-active-or-inactive.xml	This is a "negative policy" that will ensure that objects cannot be purged (permanently removed) unless they are in the "deleted" state. Purging of active or inactive objects not allowed.
6	serverAdmin	deny-reloadPolicies-if-not-localhost.xml	This is a "negative policy" that will deny requests to reload policies (i.e., policy reactivation) if this requests is not initiated from the IP address of the machine on which the repository is running (i.e., localhost).
7	serverAdmin	deny-serverShutdown-if-not-localhost.xml	This is a "negative policy" that will deny requests to shutdown the Fedora server if this requests is not initiated from the IP address of the machine on which the repository is running (i.e., localhost).
9	serverAdmin	permit-serverStatus-unrestricted.xml	This is a "positive policy" that permits unrestricted access for obtaining the Fedora server status.

6 Sample Policies for Typical Fedora Use

The Fedora Policy Enforcement Module is intended to provide a flexible means of creating access control for a repository and for digital objects within a repository. Therefore, it is expected that each Fedora repository will have XACML policies appropriate for specific contexts and use cases. All repositories will start off, out-of-the-box, with the set of default repository policies. These policies set up a world where (1) users in the Fedora administrator role are permitted to do anything (see [permit-everything-to-administrator.xml](#)), (2) access to the API-M service is restricted to localhost ([deny-apim-if-not-localhost.xml](#)), and (3) the Fedora API-A service is totally unrestricted (see [permit-apia-unrestricted.xml](#)).

Given this out-of-the-box starting point, the perspective that can be taken to easily understand how to write new custom policies is:

- write new policies to *_tighten up_* access controls for the API-A service (i.e., start to selectively deny access)
- write new policies to*_loosen up_* access controls for the API-M service (i.e., start to selectively permit access)

Please consult the [XACML Policy Writing Guide](#) which describes a reference collection of sample policies that would be useful for Fedora repositories. The collection contains examples of repository-wide and object-specific policies for restricting access to groups of digital objects based on certain attributes of the objects, for restricting access to certain kinds of datastreams and disseminations, for selectively permitting access to different API-M operations, and more. These policies can be found within the Fedora software distribution in the following directory: `FEDORA_HOME/userdocs/server/security/xacml-policies/examples`

Fedora XACML Policy Writing Guide

With a Reference Collection of Sample Policies for Fedora

1 Introduction

XACML provides a very flexible language for expressing access control policies. This document offers guidance on writing a range of useful policies for Fedora such as

- Broad repository-wide policies for controlling access to Fedora API operations
- Specific repository-wide policies for controlling access to groups of digital objects based on various attributes
- Fine-grained object-specific policies for controlling access to individual digital objects.

This guide also provides a collection of sample XACML policies intended as reference material to help in writing custom XACML policies for Fedora. The sample policies demonstrate one possible authoring style for XACML; there are other ways to write XACML policies that have the same effect. Most of the sample repository-wide policies are authored to have a **single effect**, meaning that each policy has a single rule that either permits or denies access. This style of policy writing results in many individual policies, but each policy is atomic and uncomplicated. An alternative is to have fewer policies, each with multiple rules within. This multi-rule approach can result in more complicated policies, but is, nevertheless, appropriate for writing **object-specific policies** in Fedora where a single policy states all the rules for a particular digital object. In either case, it is essential to understand the policy combining algorithm that is configured for your repository's XACML-based Authorization module. By default, the "Deny Overrides" algorithm is configured in Fedora, which means that when multiple policies are applicable to an incoming request, deny will trump permit. As you add new policies to the mix, you must be aware of what kinds of policies are already active in the repository. Also, when writing a policy that contains more than one rule, you must understand the rule combining algorithm (which is specified in the root element of an individual XACML policy). The sample policies use the "first-applicable" rule combining algorithm, which means that the first applicable rule in the policy will prevail.

This document is not intended to be a comprehensive tutorial on writing XACML policies. Anyone intending to author custom XACML policies for Fedora is encouraged to read the following documentation provided by OASIS Technical Committee that defined the XACML standard, and Sun who is the provider of the open source Sun XACML engine that is used in the Fedora implementation. It is very important to understand the basics of XACML to ensure that a suite of policies works as intended. One of the most important concepts in using XACML is understanding how multiple policies can interact with each other (in good ways, or in ways you didn't intend). By following the examples in this guide, you should be able to set up many kinds of access control policies for your repository. With additional help from the following documents, you should be able to do more advanced policies, and change some of the XACML settings for how sets of policies are combined.

More information on writing XACML policies:

[OASIS XACML Specification](#): this is the official specification and a good reference document

[A Brief Introduction to XACML](#): this is nice introduction to the XACML concepts

[OASIS XACML Technical Committee](#): this home page of the technical committee provides access to other documents on XACML

More information on the Fedora security architecture:

[Securing Your Fedora Repository](#) : documentation on security options and configuring configuring user authentication sources for Fedora repositories

[Fedora Authorization with XACML Policy Enforcement](#) : documentation on configuration and implementation of the Fedora XACML-based policy enforcement module

[Binding to user attributes to policies](#) : a discussion of how to use attributes from different sources (e.g., Tomcat, LDAP, Shibboleth) in policies

2 Writing Fedora XACML Policies

2.1 The Fedora Policy Vocabulary

A [Fedora-specific policy vocabulary](#) is defined to enable the creation of XACML policies for Fedora repositories and digital objects. This vocabulary define a set of URNs that can be used to identify specific Fedora API operations, Fedora object attributes, and the Fedora environment within an XACML policy. These URNs are used as attribute designators in XACML policies, specifically within a SubjectAttributeDesignator, ResourceAttributeDesignator, ActionAttributeDesignator, or EnvironmentAttributeDesignator.

The set of identifiers defined for the Fedora policy vocabulary can be found in the Fedora software distribution at:

```
$FEDORA_HOME\server\fedora-internal-use\vocabulary.txt
```

This vocabulary provides a set of identifiers (URNs) that can appear in XACML policies to refer to Fedora API operations (Actions in XACML), any aspects of a Fedora digital object (Resources in XACML), key attributes of the environment in which Fedora runs in (Environment in XACML), and common subject (i.e., user) attributes. (Other user attributes are named according to site-usage and so their names aren't included in the Fedora XACML vocabulary.)

2.2 Policy Basics: Identifier, Description, and Rule Combining Algorithm

Every policy has an identifier, a rule combining algorithm, and a description. In the root element of an XACML policy there is an attribute to provide the policy with a unique identifier. Also, the <Description> element provides a place to put a textual description of the purpose of the policy.

```
<Policy PolicyId="deny-apia"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable"
  xmlns="urn:oasis:names:tc:xacml:1.0:policy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Description>This policy will DENY access to THESIS datastreams.</Description>

  <Target>
    ...
  </Target>

  <Rule>
    ...
  </Rule>

</Policy>
```

The main body of a policy consists of a **Policy Target** and one or more **Rules** which are described in the next sections. Note that in the root element of a policy, the rule combining algorithm (i.e., attribute "RuleCombiningAlgId"), specifies how the Fedora Policy Enforcement Module will deal with multiple Rules in a policy (how those rules are combined and evaluated together). This algorithm is valid for only the specific policy containing it, and is independent of similar algorithms in other policies. It governs how the various effects of the potentially several rules of a policy are combined into the single effect of the policy as a whole. It is also independent of the policy-combining algorithm operative for all policies collectively, which governs how the various results of all policies are combined into a single result.

2.3 Defining the Policy Target

A Policy Target is the part of a policy that specifies matching criteria for figuring out whether a particular policy is applicable to an incoming service request. A Target contains three basic "matching" components: **Subjects**, **Actions**, and **Resources**. All of these components must be matched to the context of an incoming request for the policy to be applicable. These matching specifications can be built upon [XACML Functions](#). (A fourth matching component, Environments, is defined in XACML and will be available in Fedora's XACML policies when it is available in the Sun XACML version as used in Fedora.)


```

<Policy PolicyId="deny-apia"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable"
  xmlns="urn:oasis:names:tc:xacml:1.0:policy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Description>This policy will DENY access to THESIS datastreams.</Description>

  <Target>

    <Subjects>
      ...
    </Subjects>

    <Resources>
      ...
    </Resources>

    <Actions>
      ...
    </Actions>

  </Target>

</Rule/>

</Policy>

```

A Policy Target can be specified for a Policy (or for a PolicySet, which is an advanced way of grouping policies together). A <Target> element is defined at the Policy level (as a child of the root <Policy> element). A Policy Target applies to any contained Rules that are expressed in that policy. However, a Rule may have its own Target, in which case the Rule-level Target overrides — for that Rule only — the Policy level Target. Typically, a Target defined at the Rule level is used to replace and so tighten a broader match specification found at the overall Policy level. (This is described below.)

Resources

All Fedora resources (objects and datastreams) have attribute identifiers defined in the Fedora policy vocabulary (see: ^vocabulary.txt).

The <Resources> element of a Policy Target is used to wrap one or more descriptions of the kinds of Fedora resources (objects, datastreams, disseminations, etc.) that the policy should apply to. At runtime, the Policy Enforcement Module will compare attributes of a requested resource against the criteria in the <Resources> specification within the policy Target to determine if the policy is applicable to the incoming request. For example, to define a policy that is applicable to any Fedora resource, the following is specified:

```

<Resources>
  <AnyResource/>
</Resources>

```

Within a single <Resource> specification, there may be one or more attributes that together determine whether a policy match should occur. Each <ResourceMatch> element is used to specify the name/value of an attribute of a Fedora resource.

If multiple <ResourceMatch> elements are specified, they will be logically **AND**ed together, meaning that for a policy to be applicable to an incoming service request, **all** <ResourceMatch> specifications must match the attributes of the requested Fedora resource. The AttributeID in the <ResourceAttributeDesignator> element is used to identify a particular resource attribute by a URN, as defined in the Fedora policy vocabulary. In the example below, there are two attributes to match on: "urn:...datastream:id" and "urn:...mimeType". The snippet says that a policy match will occur if the incoming request context indicates that the requested resource has the datastream id of THESIS and the MIME type of "application/pdf."

```

<Resources>
  <Resource>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <ResourceAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:resource:*datastream:id*"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        THESIS
      </AttributeValue>
    </ResourceMatch>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <ResourceAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:resource:datastream:mimeType"
        DataType="http://www.w3.org/2001/XMLSchema#string" />
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        application/pdf
      </AttributeValue>
    </ResourceMatch>
  </Resource>
</Resources>

```

To create an **OR condition** for resource matching, multiple <Resource> elements must be specified. If there are multiple <Resource> elements within the <Resources> wrapper component, the <Resource> elements are **logically OR-ed together**. This means that a match on only one of the Resource specifications is necessary for the policy to apply to a service request. For example, the snippet below says that a resource match will occur if the incoming request is for a digital object that has the content model type of either "UVA_STD_IMAGE" or "MRSID."

```

<Resources>
  <Resource>
    <ResourceMatch
      MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        UVA_STD_IMAGE
      </AttributeValue>
    <ResourceAttributeDesignator
      AttributeId="urn:fedora:names:fedora:2.1:resource:object:contentModel"
      DataType="http://www.w3.org/2001/XMLSchema#string" />
    </ResourceMatch>
  </Resource>
  <Resource>
    <ResourceMatch
      MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        MRSID
      </AttributeValue>
    <ResourceAttributeDesignator
      AttributeId="urn:fedora:names:fedora:2.1:resource:object:contentModel"
      DataType="http://www.w3.org/2001/XMLSchema#string" />
    </ResourceMatch>
  </Resource>
</Resources>

```

Actions

All Fedora service operations have an action identifier in defined by the Fedora policy vocabulary (see: ^vocabulary.txt).

The <Actions> element of a Policy Target is used to wrap one or more service operations that this policy should apply to. At runtime, the Policy Enforcement Module will compare the identity of an incoming request against the criteria specific in the <Actions> of a Target in a policy. For example, to define a policy that is applicable to **any** Fedora service operation, the following is specified:

```

<Actions>
  <AnyAction/>
</Actions>

```

From a practical standpoint in Fedora, there are only two attributes that pertain to identifying Fedora API operations:

1. an attribute that indicates what Fedora API is in context
2. an attribute that indicates the specific service operation within that API.

To create a policy that is intended for an *entire* service (e.g., ALL operations of API-A) do the following:

```
<Actions>
  <Action>
    <ActionMatch
      MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        urn:fedora:names:fedora:2.1:action:api-a
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:action:api"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </ActionMatch>
    </Action>
  </Actions>
```

To create a policy that is about a specific operation in a Fedora API do the following:

```
<Actions>
  <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue
        DataType="http://www.w3.org/2001/XMLSchema#string">
        urn:fedora:names:fedora:2.1:action:id-getDatastreamDissemination
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:action:id"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </ActionMatch>
    </Action>
  </Actions>
```

The above can be considered a shortcut for fully qualifying a service operation within its respective service API. An alternative way to specify an Action as a Fedora API operation is to refer to the Fedora service API **AND** the service operation. As with <SubjectMatch> and <ResourceMatch> specifications, multiple <ActionMatch> **elements are *logically AND-ed together***. For example the following snippet says that the policy will match if the incoming request pertains to the Fedora API-A service AND the service request is for the "getDatastreamDissemination" operation.

```
<Actions>
  <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        urn:fedora:names:fedora:2.1:action:api-a
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:action:api"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </ActionMatch>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        urn:fedora:names:fedora:2.1:action:id-getDatastreamDissemination
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:action:id"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
      </ActionMatch>
    </Action>
  </Actions>
```

To create an **OR condition** for action matching, multiple <Action> elements must be specified. If there are multiple <Action> elements within the <Actions> wrapper component, the <Action> elements are **logically OR-ed together**. This means that a match on only one of the Action specifications is necessary for the policy to apply to a service request. For example, the snippet below says that a resource match will occur if the incoming request is **either** the getDatastreamDissemination operation of API-A or the getDissemination operation of API-A.

Note: The "shortcut" method of referring to a Fedora API operation is used in the example (i.e., we have an ActionMatch for the specific Fedora API of "api-a") because Fedora actions identifiers are unique in themselves. Fedora automatically knows that the getDatastreamDissemination operation is part of API-A.

```
<Actions>
  <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        urn:fedora:names:fedora:2.1:action:id-getDatastreamDissemination
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:action:id"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </ActionMatch>
  </Action>
  <Action>
    <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        urn:fedora:names:fedora:2.1:action:id-getDissemination
      </AttributeValue>
      <ActionAttributeDesignator
        AttributeId="urn:fedora:names:fedora:2.1:action:id"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </ActionMatch>
  </Action>
</Actions>
```

Subjects

The Fedora policy vocabulary (see: ^vocabulary.txt) defines general-purpose attributes for use in policies (e.g., login-id). However, attributes for subjects will vary depending on what a repository uses as the source of user information (e.g., tomcat-users.xml, LDAP, Shibboleth). Fedora XACML policies can make reference to the identifiers of any subject attribute that can be passed into Fedora from authenticating sources. See the section below on "Getting User Attributes into the Fedora Policy Enforcement Module" for more information on the sources of subject attributes.

The **<Subjects>** element of a Policy Target is used to wrap one or more descriptions of users or agents that this policy should apply to. At runtime, the Fedora Policy Enforcement Module will compare attributes of the user/agent making a service request against the criteria specific in the **<Subjects>** specification of the policy Target to determine if the policy is applicable to the incoming request. For example, to define a policy that is applicable to any kind of user or agent, the following is specified:

```
<Subjects>
  <AnySubject/>
</Subjects>
```

Within a single **<Subject>** specification, there may be one or more XACML attributes that together determine whether a policy match should occur. Each **<SubjectMatch>** element is used to specify an name/value of an attribute of a user/agent. Multiple **<SubjectMatch>** *elements are used to specify multiple attributes of a subject, and are *logically AND-ed together. This means that for a policy to be applicable to an incoming service request, *all* **<SubjectMatch>** specifications must match the attributes of the requesting user/agent. In the example below, there is only one attribute to match on (i.e., "fedoraRole"). The AttributeID in the **<SubjectAttributeDesignator>** element is used to identify a particular subject attribute by its local or global identifier. The snippet says that a policy match will occur if the incoming request context indicates that the user/agent has a role attribute with the value of "administrator."

```
<Subjects>
  <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        administrator
      </AttributeValue>
      <SubjectAttributeDesignator AttributeId="fedoraRole" MustBePresent="false"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </SubjectMatch>
  </Subject>
</Subjects>
```

To create an **OR condition** for subject matching, multiple **<Subject>** elements must be specified. If there are multiple **<Subject>** elements within the **<Subjects>** wrapper component, the **<Subject>** elements are **logically OR-ed together**. This means that a match on only one of the Subject specifications is necessary for the policy to apply to a service request. For example, the snippet below says that a subject match will occur if the requesting user has the role of either "administrator" or "superuser."

```

<Subjects>
  <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        administrator
      </AttributeValue>
      <SubjectAttributeDesignator AttributeId="fedoraRole" MustBePresent="false"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </SubjectMatch>
  </Subject>
  <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        superuser
      </AttributeValue>
      <SubjectAttributeDesignator AttributeId="fedoraRole" MustBePresent="false"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </SubjectMatch>
  </Subject>
</Subjects>

```

Environments

The Environments component of a Target is intended to specify aspects of the runtime environment that would make the policy match the incoming request. Such attributes include the current date, current time, the IP address of the client, and the protocol being used for the request. The **Environments** element is discussed in the OASIS XACML specification, but it is **not yet implemented by the Sun XACML engine** that underlies the Fedora Authorization module. This prevents the expression of environment matching criteria within Targets.

Therefore, do not create policies that specify Environment matching criteria in the policy Target.

Although the Environments element is not currently supported for use within a Target, this does not mean that you cannot encode matching criteria for environmental attributes within a policy. Within a policy Rule, you can specify a Condition that contains matching criteria for environmental attributes. Refer to the discussion of policy rules below for an example of Environment attribute matching in a Condition.

2.4 Defining Policy Rules

Each policy has at least one, and possibly more, rules. There must be at least one Rule in a policy that matches the incoming request for a policy to be deemed applicable to that request. The way the Sun XACML engine determines whether a rule is applicable to an incoming request is by evaluating the Target and optional Condition (if it exists). These are ANDed together, and the rule's effect achieved if the ANDed value is TRUE. (If there is no Condition, this result is simply the value of the Target.) The rule's Target is so used, and if it has no Target, the policy's Target is used instead.

```

<Policy PolicyId="deny-apia"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable"
  xmlns="urn:oasis:names:tc:xacml:1.0:policy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <Description>This policy will DENY access to Dublin Core datastreams.</Description>

  <Target>
    ...
  </Target>

  <Rule RuleId="1" Effect="Deny">

    <Target>
      ...
    </Target>

    <Condition>
      ...
    </Condition>

  </Rule>

</Policy>

```

Rule

A policy contains one or more Rules. Each rule has a **RuleId** and an **Effect**. An Effect is the intended consequence of a satisfied rule, which can be either "Deny" or "Permit." This means that if the rule is deemed applicable to an incoming service request, and the rule's conditions evaluate to TRUE, then the specified effect should be enforced.

Target

Each Rule in a policy can have its own Rule Target. While a Policy Target generally describes the kinds of requests to which an entire policy applies, a Rule Target describes the kinds of request to which a particular rule applies. If a Rule Target is not present, the Policy Target is used to determine whether the Rule is applicable to an incoming request. When a policy target exists, it is applicable to every rule in the policy which does not have its own Target. In practice, a rule target is often more constrained than the associated policy target, fine tuning to specific Subject/Resource/Action match criteria that are in the context of a particular rule.

Refer to the documentation above for the [Defining a Policy Target](#) for the structure of a Target, since rule and policy Targets are defined using the same elements.

Condition

A Condition is a predicate that must be satisfied for a rule to be assigned its effect. These predicates can be built upon [XACML Functions](#).

While Targets are appealing, frame-like expressions, they have a constrained logic which isn't always expressive enough to narrow down whether a policy is applicable to a service request. Hence, the need for Condition elements. If either the policy Target or the rule Target is not able to adequately express a constraint, a Condition can be added to a Rule. **A Condition can appear only within a Rule.** It cannot appear within a Target, nor directly under Policy or PolicySet. If a Condition is intended to be applicable to the entire Policy, the Condition must be repeated in every Rule in that Policy. Unlike the relationship of rule targets to policy targets, conditions do in fact begin with the associate (rule or policy) target, and proceed to further constrain that target.

2.5 XACML Functions

The XACML specification defines numerous functions that can be used in defining attribute match criteria in Targets and in defining predicates for Conditions. Consult the [XACML Specification](#) for a complete list of functions with their descriptions. For convenience, here is a small sampling of convenient functions with their XACML identifiers:

- **Equality predicates**
 - **String Equality** – urn:oasis:names:tc:xacml:1.0:function:string-equal
 - **Boolean Equality** – urn:oasis:names:tc:xacml:1.0:function:boolean-equal
 - **Date/Time Equality** – urn:oasis:names:tc:xacml:1.0:function:date-time-equal
 - others
- **Logical functions**
 - **OR** – urn:oasis:names:tc:xacml:1.0:function:or
 - **AND** – urn:oasis:names:tc:xacml:1.0:function:and
 - **NOT** – urn:oasis:names:tc:xacml:1.0:function:not
 - others
- **Comparison functions**
 - **Greater Than** – urn:oasis:names:tc:xacml:1.0:function:integer-greater-than
 - **Less Than** – urn:oasis:names:tc:xacml:1.0:function:type-bag
 - **Greater Than or Equal** – urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-equal
 - **Less Than or Equal** – urn:oasis:names:tc:xacml:1.0:function:integer-less-than-or-equal
 - **Date/Time Greater Than** – urn:oasis:names:tc:xacml:1.0:function:date-time-greater-than
 - others
- **Bag and Set functions**
 - **Bag of Strings** – urn:oasis:names:tc:xacml:1.0:function:string-bag
 - **Member of Set** – urn:oasis:names:tc:xacml:1.0:function:type-at-least-one-member-of
 - others

Below is an example Condition that uses several of these functions. This Condition evaluates to TRUE if the client IP address (from the environment of the incoming request) is NOT a member of a set of privileged IP addresses. The Condition element itself contains an outer-most function which is a **negation function**. Within the condition, we see the application of the **set membership function**, which specifies that the environment attribute "clientIpAddress" (from the Fedora vocabulary) should be evaluated. Finally, the inner most **bag function** wraps a set of possible values for the clientIpAddress attribute. Again, if the clientIpAddress on the incoming request is not one of those in the bag of addresses, then the rule's Deny effect should take place.

```

<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-at-least-one-member-of">
    <EnvironmentAttributeDesignator
      AttributeId="urn:fedora:names:fedora:2.1:environment:httpRequest:clientIpAddress"
      DataType="http://www.w3.org/2001/XMLSchema#string"/>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        127.0.0.1
      </AttributeValue>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        128.84.103.11
      </AttributeValue>
    </Apply>
  </Apply>
</Condition>

```

In summary, each policy must have at least one Rule. For a Rule to have an effect, (1) the Rule must match the incoming request by virtue of a Target match (either via a policy Target, or a constraining rule Target), and (2) if a Condition is specified, the condition predicate evaluates to TRUE. An applicable rule will result in a Permit or Deny for an incoming request, based on what is specified in the Rule Effect.

2.6 Required vs. Optional Attributes in a Policy

There are times when an attribute that is referred to by a policy target will not be available on an incoming service request. By default, when the policy matching activity occurs - and an attribute specified in a policy is not found in the incoming request context - an Indeterminate result is returned and an authorization exception is thrown. Policy authors can avoid unwanted Indeterminate results by indicating in the attribute designators of a Target or Condition that a particular attribute can be considered optional in terms of whether it must exist in the incoming request context. This is done by setting `MustBePresent="false"` on a `SubjectAttributeDesignator`, `ResourceAttributeDesignator`, `ActionAttributeDesignator`, or `EnvironmentAttributeDesignator` element. This will tell the Fedora Policy Enforcement module that it's ok if the incoming request does not have the specified attribute available within it. (The implicit/unstated default is `MustBePresent="true"`)

Let's take an example to make this clearer. Consider a policy where the `SubjectMatch` specification talks about an attribute "fedoraRole" and specifies that the value of this attribute must be "administrator" in order for this policy to be considered applicable by the PDP. Now consider an incoming service request that has a user login id (e.g., "wdn5e") in the request context, but this user does not have a "fedoraRole" attribute associated with it. So, when the PEP tries to determine whether this policy is applicable to the incoming service request, it returns `INDETERMINATE` because it can't figure out whether there is a subject match without the presence of a "fedoraRole" attribute. This will cause an authorization exception to be thrown for the request because the PDP expects the "fedoraRole" attribute to be present in the request context. However, we essentially want to somehow indicate that the fedoraRole attribute is considered "optional" on an incoming request (i.e., not every incoming request must have this particular attribute in context). To do this, you must indicate in the policy Target that the attribute does not have to be present (`MustBePresent="false"`) in the incoming request as follows:

```

<Subjects>
  <Subject>
    <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
        administrator
      </AttributeValue>
      <SubjectAttributeDesignator AttributeId="fedoraRole" MustBePresent="false"
        DataType="http://www.w3.org/2001/XMLSchema#string"/>
    </SubjectMatch>
  </Subject>
</Subjects>

```

In this example, it's easy to imagine that another policy could independently permit access. Hence the fit of `MustBePresent="false"`: if the policy above lacks an attribute, it may not be crucial to the ultimate authorization decision. Policies are not authored in isolation, but to work together.

2.7 Recommended Best Practices for Authoring Fedora XACML Policies

1. Set the `PolicyId` attribute in the XACML policy to match the filename of the policy.

```
<Policy PolicyId="deny-objects-to-students" . . . > – corresponds to filename of "/repository-policies/deny-objects-to-students.xml"
```

2. For object-specific policies, especially if kept in an XML file, set the `PolicyID` in the XACML and the policy filename to match the object PID, but with concession to demand of OS filenames (e.g., uses dash instead of colon).

```
<Policy PolicyId="demo-5" . . . > – corresponds to filename of "/object-policies/demo-5.xml"
```

3. Policies should use simplest rule-combining algorithm which gives desired outcome. Avoid a more complicated algorithm which happens to work, but which confuses because it implies more than what's there. A simple choice is the "first-applicable" rule combining algorithm

which give precedence to the first rule in a policy to apply to a situation.

```
<Policy PolicyId="demo-5" RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable" . . . >
```

4. An object-specific policy should be coded so that it applies only to that specific object. So-coded, if misplaced among general repository policies, that wouldn't be hurtful.
5. Try to stick with "single-effect" policies, that is a policy should either permit or deny. Things can get confusing if a single policy has some rules that permit and some that deny. If most policies are single-effect, try to have them all be single-effect. You may wind up writing more individual policies, some that deny and some that permit, but from a policy management standpoint, it is probably easier to have atomic, unambiguous policies.

2.8 XACML Gotchas

1. XACML provides for an AttributeValue in a <Target> evaluation as a single value, but provides for an AttributeValue in a <Condition> evaluation as "bags" (sets), doing so even for either singleton or empty bags. Code policies accordingly.
2. MatchId functions (which are used in Targets) are much restricted in allowed values, compared to the values allowed in the analogous FunctionIds (which are used in Conditions). There are no existing functions which are self-contained boolean combinations, such as not-equal. Since attributes are generally not boolean themselves (and so possibly negated), the not function can't be used as a MatchId, e.g., in a SubjectMatch element. Since SubjectMatch, e.g., expresses a single binary operation, there is no possibility of introducing negative logic into a Target. [An exception would be an explicit value returned by an attribute finder, which would signify the absence of the attribute.]
3. Despite some evidence that <Environments> was added to <Target> generally, it doesn't seem to work currently in sunxacml.
4. sunxacml has a relaxed parsing of policies; e.g., we have encountered schema violation (e.g., Action omitted between Actions and ActionMatch) which resulted only in the policy not being evaluated correctly, as opposed to failing parse. How widespread this is, we don't know. As a precaution, policies should be tested for effect. This is good practice, anyway, since testing is the only check of the policy-writer's understanding of xacml and against the inevitable typo.
5. Though sunxacml parsing is relaxed, <Description> </Description> apparently requires at least one-character content: <Description/> doesn't do it.
6. In SubjectMatch, ResourceMatch, and ActionMatch blocks, place AttributeValue elements before AttributeDesignator. Also, avoid using two AttributeDesignator elements (without any AttributeValues). Though it may seem logical to use other ordering or attribute selection, it doesn't match the standard and won't work.

```
<ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:dateTime-less-than">
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#dateTime">
    2004-12-07T20:22:26.705Z
  </AttributeValue>
  <ResourceAttributeDesignator
    AttributeId="urn:fedora:names:fedora:2.1:resource:object:lastModifiedDate"
    DataType="http://www.w3.org/2001/XMLSchema#dateTime" />
</ResourceMatch>
```

3. Default Repository-Wide Policies

out-of-the-box, the Fedora repository is configured with a default set of access control policies that provide for a highly restricted management service (API-M), an open access service (API-A), and an open OAI provider service. These default access control policies establish the same level of out-of-the-box security on the repository that was previously configured for Fedora 2.0 release; however, as of Fedora 2.1 these basic access controls are now specified as XACML policies. Please consult the [Default Repository Policies](#) documentation for a description of each default policy.

4. Custom Policies - Sample Repository-Wide Policies

The sample policies are written with the assumption that Fedora's [Default Repository-Wide Policies](#) are unedited and activated. These default policies lock down access to the Fedora API-M service so that only the Fedora Administrator is permitted access. The default policies also result in open access to API-A (all users are permitted). Given this starting point, you can think writing custom policies as a way to "loosen up" the API-M defaults and "tighten up" the API-I defaults. In other words, it is likely that you will want to write policies that let more users have access to API-M operations. Also, you will likely want to add restrictions in who can access digital objects, datastreams, and disseminations (i.e., via API-A). The sample policies will demonstrate how to do these things, given various authentication scenarios. Notice that there are some policies which restrict access based on user identity/attributes based on Tomcat's default user directory (tomcat-users.xml). Other policies demonstrate how to restrict access by user attributes/groups that are defined in an LDAP directory.

Note that the sample policies have been written for demonstration purposes. They are not intended to work as a collaborative set of policies, since they often demonstrate different ways of doing the same basic thing (e.g., one policy demonstrating rules based on Tomcat user identity, another showing a similar thing with LDAP groups). If you want to try them out, you can put one or more of the sample policies into play by following the instructions for activating and loading policies into a Fedora repository. However, it is recommended that you test them individually to understand the effect each policy has. This approach of augmenting the default policies, which are left as-is, allows progressive learning, without

endangering your repository. It may be that this approach goes farther in opening up API-M than in tightening up API-I, and that eventually the default policy for API-A will need to be replaced by one or more policies written to your site's needs. So it goes. Ultimately, you can proceed to write a meaningful suite of policies that are intended to work together for your repository.

4.1 Repository Policies to *tighten* the API-A defaults at the *service interface* level

Policy	Service	XACML Policy File	Policy Description
4.1.1	API-A	deny-apia-to-ldap-group.xml	Deny access to all API-A methods to users who are "Librarians" or "Info Technologists" (as indicated by their LDAP attributes).
4.1.2	API-A	deny-apia-if-not-tomcat-role.xml	This policy will DENY access to ALL API-A methods to users who are NOT in the "administrator" or "professor" ROLES.
4.1.3	API-A	deny-apia-to-tomcat-user.xml	This policy will deny access to all API-A methods to a particular user based on login id (as registered in the tomcat-users.xml file).
4.1.4	API-A	deny-apia-except-by-owner.xml	Deny access to all API-A methods to any user unless that user is the owner of the object being accessed. This sample policy primarily exists to show how to create a policy that compares the owner-id of an object to the login-id of the current user. It is important to note that due to how XACML policies are processed, you cannot do this comparison in the <Subject> section of the XACML policy. The comparison must appear in a <Condition> specification in the <Rule> section.

4.2 Repository Policies to *tighten* the API-A defaults based on *object attributes*

Policy	Service	XACML Policy File	Policy Description
4.2.1	API-A	deny-objects-by-pids-to-tomcat-role.xml	Overall, this policy will identify a set of objects by their PIDs and it will DENY ALL APIA access to users of particular ROLES. NOTE: As a repository-wide policy, this policy demonstrates how to control access to specific objects (identified by PID). As an alternative, it is possible to create "object-specific" policies that either resides in the digital object's POLICY datastream, or that is stored in the object-specific policy directory. (See the Fedora system documentation on XACML policies for more information.)
4.2.2	API-A	deny-objects-by-cmodel-to-ldap-group.xml	This policy will DENY all APIA access to digital objects that are EAD Finding AIDS. This is based on the object content model attribute having a value of "UVA_EAD_FINDING_AID." Specifically, the policy will DENY access to users that belong to a particular LDAP-defined GROUP.
4.2.3	API-A	deny-objects-hide-datastreams-if-not-tomcat-role.xml	The overall intent of this policy is datastream hiding, meaning that raw datastreams must not be accessible to anyone except very privileged users, but service-mediated disseminations are accessible by a broader audience. The key point is that students can access disseminations of the object, but not the raw datastreams. This might typically be done in cases where lesser privileged users are given a derivation of the main datastream, or a lesser quality view, or a less complete view of the raw datastream content. Given that an object is of a certain content model (in this case UVA_STD_IMAGE), this policy will DENY datastream access to users who do NOT have the ROLE of "administrator" or "professor". It will also DENY dissemination access to users who do NOT have the ROLE of "student," "administrator," or "professor."

4.3 Repository Policies to *tighten* the API-A defaults at the *datastream* level

Policy	Service	XACML Policy File	Policy Description
4.3.1	API-A	deny-apia-datastream-all-to-all-users.xml	This policy will DENY access to ALL datastreams. Specifically, it will DENY access to ALL USERS making requests to the getDatastreamDissemination method of API-A.
4.3.2	API-A	deny-apia-datastream-DC-to-all-users.xml	This policy will DENY access to Dublin Core datastreams. Specifically, it will DENY access to ALL users making getDatastreamDissemination requests on API-A to obtain datastreams with an identifier of 'DC.'
4.3.3	API-A	deny-apia-datastream-DC-to-tomcat-group-ALT1.xml	This policy will DENY access to Dublin Core datastreams. Specifically, it will deny access to USER GROUPS making getDatastreamDissemination requests on API-A for datastreams with a datastream identifier of 'DC.' User groups are defined using custom roles in the tomcat-users.xml file.
4.3.4	API-A	deny-apia-datastream-DC-to-tomcat-group-ALT2.xml	This policy will DENY access to Dublin Core datastreams. Specifically, it will deny access to USER GROUPS making getDatastreamDissemination requests on API-A for datastreams with a datastream identifier of 'DC.' User groups are defined using custom roles in the tomcat-users.xml file.
4.3.5	API-A	deny-apia-datastream-MRSID-if-not-tomcat-role.xml	This policy will DENY access to MRSID image datastreams by controlling access to the getDatastreamDissemination method of the Fedora Access Service (API-A). Specifically, it will DENY access to users who are NOT of particular ROLES when the requested resource is a datastream with identifier of 'MRSID.'
4.3.6	API-A	deny-apia-datastream-TEISOURCE-to-tomcat-user.xml	This policy will DENY access to TEI datastreams by controlling access to the getDatastreamDissemination method of the Fedora Access Service (API-A). The TEI datastream is identified as a Resource where the Fedora datastream id has the value of 'TEISOURCE.' This policy will DENY

access to a SPECIFIC USER based on login id (as registered in the tomcat-users.xml file).

4.4 Repository Policies to *tighten* the API-A defaults at the *dissemination* level

Policy	Service	XACML Policy File	Policy Description
4.4.1	API-A	deny-apia-dissem-demo1-getMedium-to-all-users.xml	This policy will DENY access to the 'demo:1/getMedium' dissemination (defined on a disseminator that subscribes to the demo:1 behavior definition. Specifically, it will DENY access to ALL users making getDissemination requests on API-A for the 'demo:1/getMedium' dissemination.
4.4.2	API-A	deny-apia-dissem-demo1-getMedium-to-ldap-group.xml	This policy will DENY access to the 'demo:1/getMedium' dissemination (defined on a disseminator that subscribes to the demo:1 behavior definition. Specifically, it will DENY access to users of particular LDAP-defined GROUPS who are making getDissemination requests on API-A for the 'demo:1/getMedium' dissemination.
4.4.3	API-A	deny-apia-dissem-demo1-getMedium-if-not-tomcat-role.xml	This policy will DENY access to the 'demo:1/getMedium' dissemination (defined on a disseminator that subscribes to the demo:1 behavior definition. Specifically, it will DENY access to users who are NOT of particular ROLES who are making getDissemination requests on API-A for the 'demo:1/getMedium' dissemination.
4.4.4	API-A	deny-apia-dissem-demo1-getMedium-to-tomcat-user.xml	This policy will DENY access to disseminations that are available on objects via a disseminator subscribing to the 'demo:2' behavior definition. Specifically, it will DENY access to a particular user (as registered in the tomcat-users.xml file).
4.4.5	API-A	deny-apia-dissem-DualResImage-to-all-users.xml	This policy will DENY access to ALL disseminations that are available on objects via a particular disseminator (one that subscribes to an image-based behavior definition whose PID is 'demo:2/DualResImage'. Specifically, it will DENY access to ALL users making getDissemination requests on this disseminator.

4.5 Repository Policies to *loosen* the API-M defaults at the *service interface* level

Policy	Service	XACML Policy File	Policy Description
4.5.1	API-M	permit-apim-by-ldap-group.xml	
4.5.2	API-M	permit-apim-by-tomcat-group.xml	
4.5.3	API-M	permit-apim-by-tomcat-user.xml	
4.5.4	API-A/API-M	permit-if-owner.xml	If the logged-in user is the owner of an object, permit all actions. Note: This policy also works if the object has multiple owners and the logged-in user is one of them.

5 Custom Policies - Sample Object-Specific Policies

5.1 Object-specific policies with multiple policy rules

Object-specific policies are policies that refer to one particular digital object. An object-specific policy is stored in the "POLICY" datastream of the digital object to which it pertains.

Policy	Service	XACML Policy File	Policy Description
5.1.1	N/A	demo-5.xml	By using multiple policy rules , this policy shows how to deny access to all raw datastreams in the object except to particular users (e.g., the object owners). It also shows how to deny access to a particular disseminations to selected user roles.
5.1.2	N/A	demo-11.xml	By using multiple policy rules , this policy shows how to deny access to particular datastreams in the object. 1) The policy will DENY everyone except professors and researchers access to particular source datastreams of the demo:11 object by controlling access to the getDatastreamDissemination method of the Fedora Access Service (API-A). 2) The policy will DENY everyone except students, professors, and researchers, access to all disseminations of demo:11. 3) This policy will also DENY ALL access to the demo:11 object to a SPECIFIC USER based on login id (as registered in the tomcat-users.xml file). NOTE: The net effect of the policy permits open access to the descriptive metadata datastream of demo:11.
5.1.3	N/A	demo-26.xml	By using multiple policy rules , this policy shows how to deny access to particular datastreams in the object. The policy will DENY visitors access to the TEI and FOP source datastreams of the demo:26 object by controlling access to the getDatastreamDissemination method of the Fedora Access Service (API-A). These datastreams are open to all other kinds of users, and Disseminations are open to all users. This is an object-specific policy. It could be stored inside the demo:26 digital object in the POLICY datastream OR in the directory for object-specific policies. (The directory location is set in the Authorization module configuration in the Fedora server configuration file (fedora.fcfg).

XACML Vocabulary and Examples

- [XACML Example Object Policies](#)

- XACML Example Repository Policies
- Vocabulary

XACML Example Object Policies

- demo-5.xml
- demo-11.xml
- demo-26.xml

XACML Example Repository Policies

- permit-anything.xml
- permit-if-owner.xml
- schema-broken.xml

- Draconian Restrictions
- API-M Loosen Defaults
- API-A Tighten Defaults

API-A Tighten Defaults

- API-A Restrict All Methods
- API-A Restrict Datastreams
- API-A Restrict Disseminations
- API-A Restrict Objects By Attribute

API-A Restrict All Methods

- deny-apia-except-by-owner.xml
- deny-apia-if-not-tomcat-role.xml
- deny-apia-to-ldap-group.xml
- deny-apia-to-tomcat-user.xml

API-A Restrict Datastreams

- deny-apia-datastream-all-to-all-users.xml
- deny-apia-datastream-DC-to-all-users.xml
- deny-apia-datastream-DC-to-tomcat-group-ALT1.xml
- deny-apia-datastream-DC-to-tomcat-group-ALT2.xml
- deny-apia-datastream-MRSID-if-not-tomcat-role.xml
- deny-apia-datastream-TEISOURCE-to-tomcat-user.xml

API-A Restrict Disseminations

- deny-apia-dissem-demo1-getMedium-if-not-tomcat-role.xml
- deny-apia-dissem-demo1-getMedium-to-all-users.xml
- deny-apia-dissem-demo1-getMedium-to-ldap-group.xml
- deny-apia-dissem-demo1-getMedium-to-tomcat-user.xml
- deny-apia-dissem-DualResImage-to-all-users.xml

API-A Restrict Objects by Attribute

- deny-objects-by-cmodel-to-ldap-group.xml
- deny-objects-by-pids-to-tomcat-role.xml
- deny-objects-hide-datastreams-if-not-tomcat-role.xml

API-M Loosen Defaults

- API-M Permit All Methods

API-M Permit All Methods

- permit-apim-by-ldap-group.xml
- permit-apim-by-tomcat-group.xml
- permit-apim-by-tomcat-user.xml

Draconian Restrictions

- deny-apia-to-all-users.xml
- deny-apim-to-all-users.xml

- deny-if-in-ip-address-range.xml
- deny-if-modified-after.xml
- deny-if-not-authenticated.xml
- deny-if-not-in-ip-address-range.xml
- deny-object-to-all-users.xml

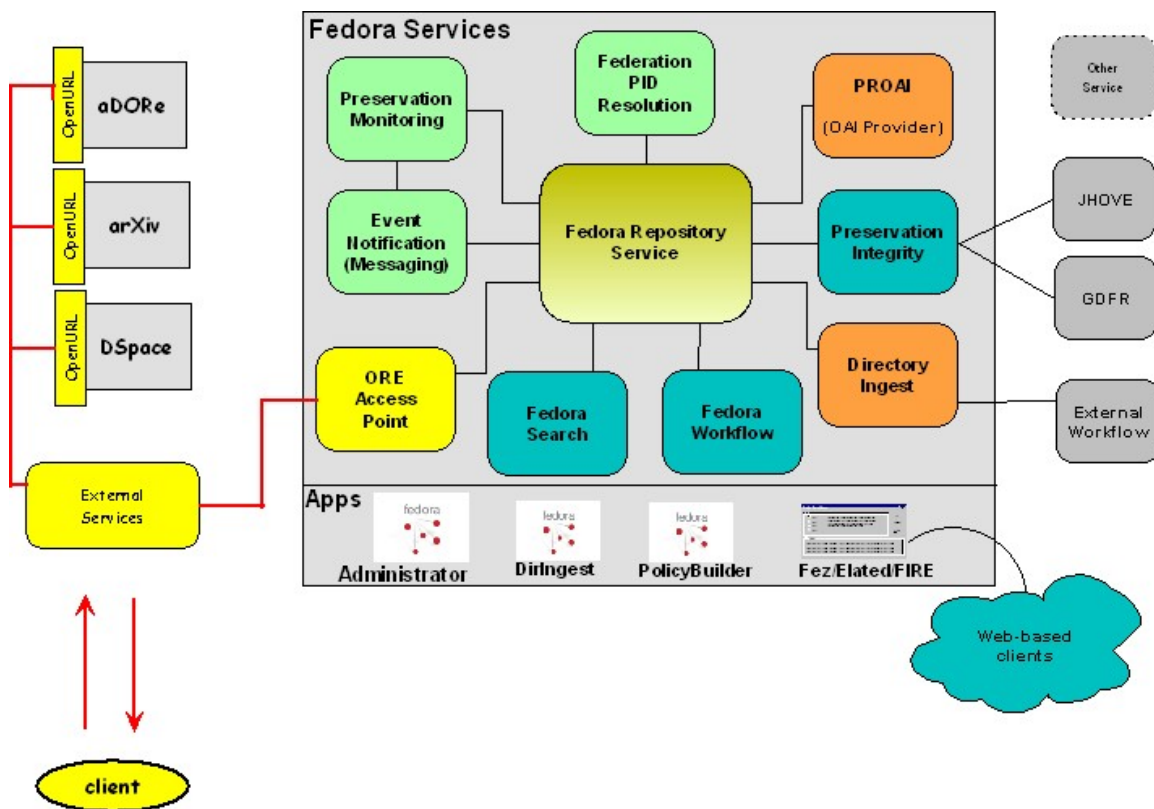
Service Framework

Introduction

As of Fedora 2.1, the Fedora Service Framework was introduced to facilitate the integration of new services with the Fedora repository. The framework takes a service-oriented architecture approach to building new functionality around a Fedora repository. While the Fedora repository, itself, exposes its functionality as a set of web service interfaces, all of these interfaces belong to the Fedora web application that runs in its own Tomcat. The new Fedora Service Framework allows new services to be built around the core repository - as stand-alone web applications that run independently of the Fedora repository. While Fedora repository functionality can still be extended with new modules, the intent is to keep the repository service focused on the core functions of a repository. Yet, there are many other services that are beneficial companions to a repository, such as specialized ingest services, workflow services, preservation services, and many others. These are the kinds of services that the framework is intended to support. There are two main benefits to the service framework approach: (1) it allows new functionality to be added as atomic, modular services that can interact with Fedora repositories, yet not be part of the repository, (2) it makes co-development of new services for Fedora easier since each service can be independently developed and plugged into the framework. As of Fedora 2.1, the Fedora development team has released an initial set of services ([Directory Ingest](#) and [OAI Provider](#) described below), and will continue to develop new services over the course of Fedora Phase 2 (2005-2007) and beyond, especially services for workflow and preservation. Services that are part of the framework will be packaged as part of the Fedora open-source software distribution and will be kept up to date with new versions of the core Fedora repository service. Members of the Fedora community will be collaborating on the development of services and will contribute back to the Fedora Project. Further documentation will be provided to establish guidelines on how services should be designed to effectively plug into the framework. In the mean time, developers of new services can follow the design patterns of the Directory Ingest and OAI Provider services.

Framework and Services

The Fedora Service Framework establishes a means for coupling new services with the core Fedora repository service. The framework allows for the creation of atomic, modular services that can interact with the Fedora repository or each other. The diagram below depicts the Fedora Service Framework as it is envisioned to evolve during 2005-2007. The repository service was the first deliverable of the Fedora Project (2002-present). In 2006, the next two services were introduced with Fedora 2.1: the OAI Provider Service and the Directory Ingest services. In January 2007, the Fedora Search Service is being deployed as part of the framework. The Search Service (known as GSearch) was contributed to the Fedora Project by community member Gert Schmeltz Pedersen of the Danish Technical University. During the rest of Phase 2 of the Fedora Project, both the Fedora development team, and the Fedora community will develop the other services to fit into the framework. In 2007, development will begin on the Fedora Preservation, Event/Messaging, and Workflow services. During this phase of development we will move Fedora towards an "enterprise" architectural pattern.



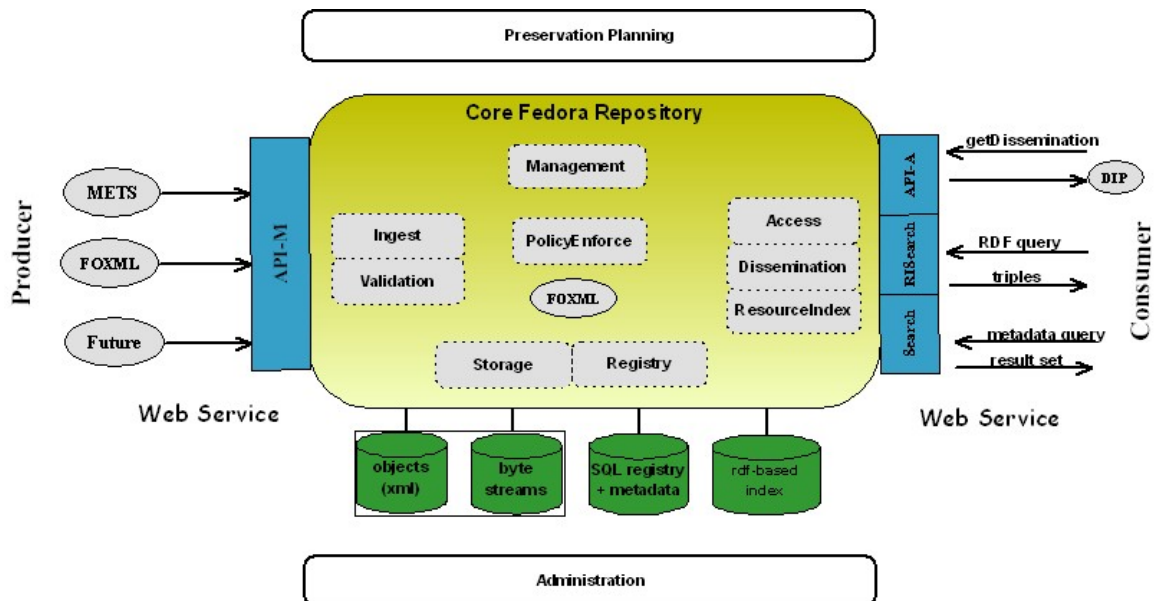
The Fedora Service Framework can evolve to include new services conceived of by the Fedora community. Listed below is a brief description of each service, links to specifications when available, and status.

- **Repository Service:** at the heart of Fedora is the repository service that enables the creation, management, storage, access, and reuse of digital objects.
- **OAI Provider Service:** a configurable OAI Provider service for harvesting metadata out of a Fedora repository via OAI-PMH. The service can be configured to harvest any type Datastream or dissemination from objects in the repository. It also supports OAI sets.
- **Directory Ingest Service:** a service to ingest a hierarchical directory of files into a Fedora repository. The service will accept a Submission Information Package (SIP) in the form of a .zip archive that contains directories of files along with a METS-based manifest file that describes the directory hierarchy. The default parent-child relationships that characterize a directory hierarchy can be overridden and refined to have other semantic meaning (e.g., collection-member, folder-document). Upon receipt of the SIP, the Directory Ingest service will process the .zip archive and create a Fedora digital object in the repository for every file and every directory, plus it will record the relationships among them in Fedora's RDF-based relationships datastream. *A new web-based client for creating the SIP is now available (see SIP Creator). This client will enable a user select files from a file system, add metadata about files, and assert semantically-meaningful relationships, and ultimately submit the SIP to a Fedora repository.*
- **Search Service:** a configurable search service that can index any datastream or dissemination of Fedora digital objects. The service is pluggable, and will provide adapters for Lucene and Zebra as the default backend search engines. Other search adapters can be developed to plug into other engines.
- **Workflow and Orchestration Service:** (planned 2007, under specification by Fedora Workflow Working Group and Fedora Development Team)
- **Preservation Integrity Service:** (planned 2007, currently under specification by Fedora Preservation Working Group)
- **Preservation Monitoring and Alerting Services:** (planned 2007, currently under specification by Fedora Preservation Working Group)
- **Event Notification Service (Messaging):** (planned 2007)
- **Persistent Identifier Resolution Service**
- **Object Reuse and Exchange (ORE) Access Point:** an interface to a Fedora repository to facilitate cross-repository interoperability (2007-2008) The Fedora Service Framework provides building blocks for higher-level customized services and user applications.

Core Repository Service

Fedora Core Repository Service can be run as a stand-alone service, or it can be situated within the Fedora Service Framework, where a suite of companion services can be loosely coupled with the repository to provide additional functionality for integrating the repository into a broader service-oriented architecture (SOA) pattern. The core repository can be accessed via web service interfaces to its core functionality. The core repository service actually has several web service APIs: an interface for repository management (API-M); an interface for repository access (API-A); interface for basic repository search; and an interface for RDF-based search of the Resource Index. All of these web service interfaces are available on the Fedora repository server web application that runs in Tomcat. The repository service is built in a modular manner, so that each inner function is implemented as a java-based module. The inner modules are configurable, and they can be replaced with alternate implementations.

The Fedora repository service is the core service in the Fedora Service Framework, and was depicted in the above diagram in the center of all other services. Below, the Fedora repository service is depicted in more detail, with its inner modules exposed, and all repository interfaces. The diagram depicts the repository service from the perspective of how it maps to the [Open Archival Information System \(OAIS\)](#) reference model which has been approved as an ISO standard.



Versioning

Introduction

As of v1.2, Fedora for the first time included content versioning functionality. Content versioning is one of the preservation-oriented features of Fedora. Using this new capability, Fedora repositories can maintain a record of how digital objects have changed over time. This is achieved by the Fedora repository storing former versions of content, and by creating audit trail records about changes. Also, with the advent of content versioning, the Fedora Access interfaces now support date-time stamped requests, so that a client can "go back in time" and see a digital object as it looked in the past.

As of v2.2, the versioning capabilities of Fedora have been enhanced. Although the default operation for digital object in a Fedora Repository is still that any modification of a Datastream will create and store a new version of that Datastream, it is now possible to have a greater degree of control over the versioning capability. Now whenever a Datastream is created (either through the createDatastream API function, or as a part of ingesting an entire digital object) the user can specify whether that specific Datastream is to be versioned or not.

Additionally, there are now API functions that allow the user greater control of which modifications of Datastreams stored for a digital object are important enough to merit maintaining the version in perpetuity, and which are less important administrative changes which need not be preserved.

Preservation

The ability to version content of Fedora data objects provides repositories with the ability to preserve not only the data for any given data object over time, but also to preserve the look and feel of the dissemination of that data object, since the original content is linked to the original delivery mechanisms used to disseminate that content.

How It Works by Default

In the more versatile versioning system now provided by Fedora, the default operation is that any modifications made to a Datastream through the Fedora management interface (API-M) will automatically result in a new version of that Datastream being created by Fedora. Fedora will not create a new version of the whole digital object, instead it will version these specific components within the digital object container. This has the benefit of the digital object PID remaining constant, and not having to keep track of multiple distinct versions of a digital object.

The Fedora repository typically will maintain all versions of all Datastreams, thereby creating a history of how objects change over time. Additionally, Fedora maintains an audit trail record of the nature of the object change events (e.g., who, what, when, why).

While the components of data objects are versioned in Fedora v2.0 through 2.2, versioning of Service Definitions and Service Deployments) is in scope for future releases of the software.

Disabling Versioning

As of v2.2, the Fedora versioning system allows a user to selectively *disable* versioning for a given Datastream of a digital object (via the new API function setDatastreamVersionable). Subsequently, any modifications made to that specific Datastream will *replace* the most recent stored version with the result of the modification. This replacement only applies to the most recent version of that Datastream, any versions that were in existence before versioning was disabled will remain unchanged. Thereafter if the user decides that new modifications ought to cause new versions to be stored, versioning can be turned back on (again through the API function setDatastreamVersionable).

Purging Old Versions

To provide more control over what versions of a particular Datastream are to be kept, the API function purgeDatastream has been made more versatile. Previously purgeDatastream could only be used to delete the oldest version (or versions) of a Datastream. If, for instance a Datastream had six different versions that had been created over its lifetime (numbered 1 through 6 from oldest to most recent) and a user wanted to get rid of the fourth stored version, the only action the user could take previously would be to delete versions 1 through 4.

Now rather than specifying single date, and deleting all versions older than that date, the purgeDatastream function allows a start date and end date to be specified which will allow any single previous version (or any contiguous range of versions) to be deleted.

Fedora Administrator

Any component of an object that may be modified is shown in Fedora Administrator as either an editable text box or by means of an Edit button on the active pane. Any modification to a data object component results in a new version of that component being created and stored in the XML of the data object.

Previous versions of an edited Datastream may be viewed, but cannot be edited. A slider bar is provided to indicate that previous versions of a Datastream exist. Versions are represented on this timeline by small vertical lines. Clicking on the bar will move the slider to the points on the timeline. Additionally the slider may be dragged along the timeline to show all versions.

The following screen shots show a versioned image Datastream. Note that a dropdown box is present on the Datastream pane indicating whether versioning is enabled for that particular Datastream (ie. whether Updates will create new version or Updates will replace the most recent version):

Fedora Administrator - fedoraAdmin@localhost:8080


File Tools Builders Window Help

Object - demo:5

Properties Datastreams Disseminators

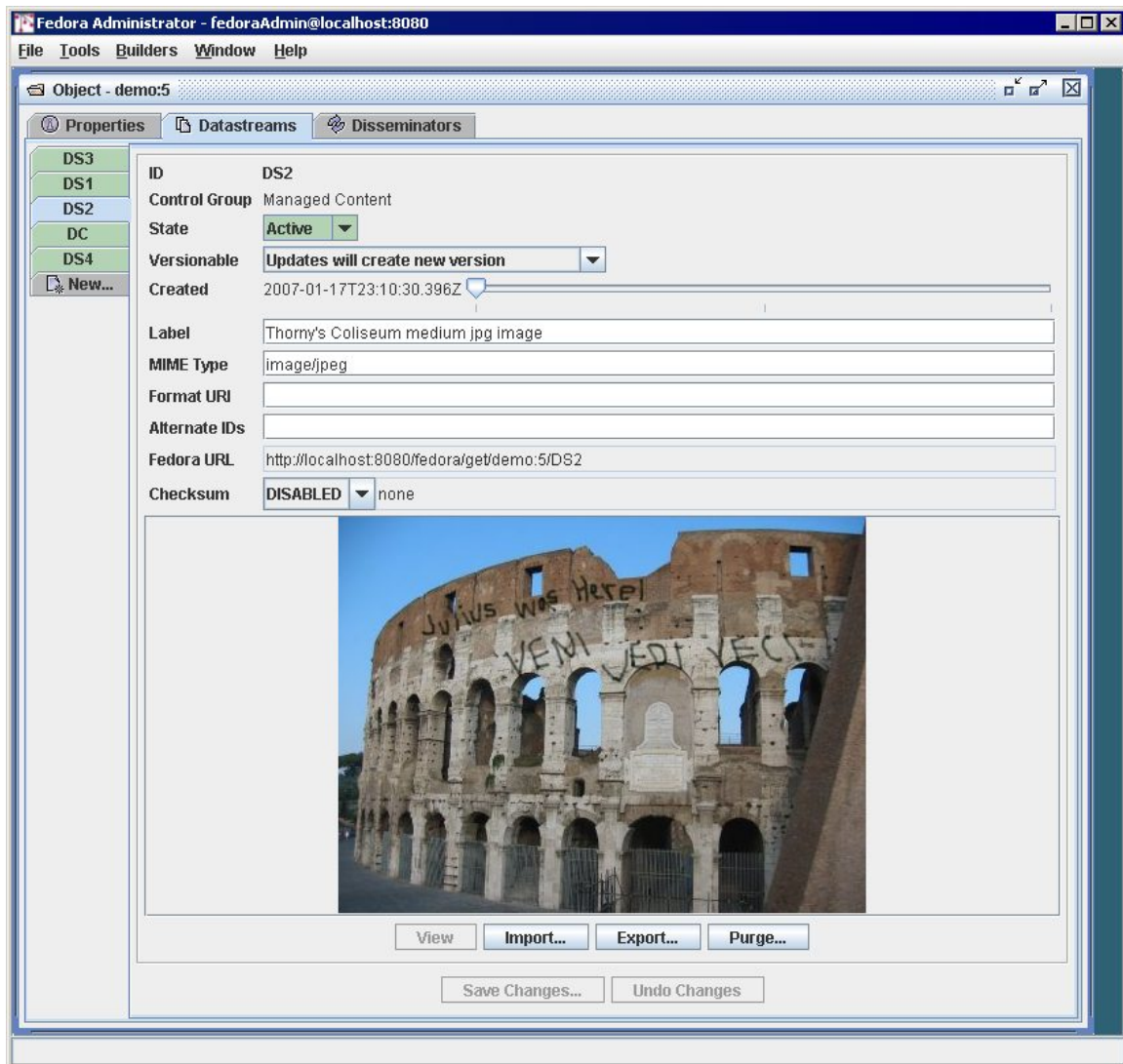
DS3
DS1
DS2
DC
DS4
New...

ID DS2
Control Group Managed Content
State Active
Versionable Updates will create new version
Created 2007-01-17T23:00:12.091Z
Label Thorny's Coliseum medium.jpg image
MIME Type image/jpeg
Format URI
Alternate IDs
Fedora URL http://localhost:8080/fedora/get/demo:5/DS2/2007-01-17T23:00:12.091Z
Checksum DISABLED



View Export... Purge...

Save Changes... Undo Changes



Search and Retrieval in Fedora Administrator

Retrieval of a modified data object requires using the Advanced Search tab on the Search Repository dialog and entering a specific date/time stamp as a search criteria.

All components are given a created date (cDate) value at ingest. Whenever the component is modified, the modified date (mDate) value is updated. By searching on the modified date, it is possible to retrieve a result set of modified data objects. Wild card values are allowed in the date/time search criteria. For example:

```
mDate<=2003-08-28T*
and
mDate<=2003-08-28T23:59:59
```

are both valid search criteria.

In the case of Dublin Core Metadata Datastreams, the modified date is stored in the dcmDate parameter. To retrieve data objects with modified Dublin Core Metadata Datastreams, the user must search using this parameter name.

API-A-Lite Web Interface

Retrieval of specific versions of objects via API-A-Lite also depends on entering date/time parameter values. This section gives the syntax for the getDissemination and getObjectSyntax methods.

If no date/time value is used in the URL, the most current version of a data object is always returned as the result of a search.

getDissemination syntax: `http://hostname:port/fedora/get/PID/sDefPID/methodName*[/dateTime][?parmArray]*`

This syntax requests a dissemination of the specified object using the specified method of the associated behavior definition object. The result is returned as a MIME-typed stream.

- **hostname** - required hostname of the Fedora server.
- **port** - required port number on which the Fedora server is running. **fedora** - required name of the Fedora access service.
- **fedora** - a required parameter specifying the Fedora servlet path.
- **get** - a required parameter specifying the Fedora servlet path.
- **PID** - required persistent identifier of the digital object.
- **sDefPID** - required persistent identifier of the service definition object to which the digital object subscribes.
- **methodName** - required name of the method to be executed.
- **dateTime** - value indicating dissemination of a version of the digital object at the specified point in time. Proper syntax is YYYY-MM-DDTHH:MM:SS where HH is 24-hour clock.
- **parmArray** - optional array of method parameters consisting of name/value pairs in the form parm1=value1&parm2=value2...

For example: `http://localhost:8080/fedora/get/ronda:2/demo:1/getItem/2003-08-28T00:01:01`

getObjectProfile syntax: `http://hostname:port/fedora/get/PID*/[dateTime][?xml=BOOLEAN]*`

This syntax requests an object profile for the specified digital object. The XML parameter determines the type of output returned. If the parameter is omitted or has a value of "false," a MIME-typed stream consisting of an HTML table is returned providing a browser-savvy means of viewing the object profile. If the value specified is "true," then a MIME-typed stream consisting of XML is returned.

- **hostname** - required hostname of the Fedora server.
- **port** - required port number on which the Fedora server is running. **fedora** - required name of the Fedora access service.
- **fedora** - a required parameter specifying the Fedora servlet path.
- **get** - a required parameter specifying the Fedora servlet path.
- **PID** - required persistent identifier of the digital object.
- **dateTime** - value indicating dissemination of a version of the digital object at the specified point in time. Proper syntax is YYYY-MM-DDTHH:MM:SS where HH is 24-hour clock.
- **xml** - an optional parameter indicating the requested output format. A value of "true" indicates a return type of text/xml; the absence of the xml parameter or a value of "false" indicates format is to be text/html.

For example: `http://localhost:8080/fedora/get/ronda:2/2003-08-28T00:01:01`

Web Service Interfaces

Primary APIs

These APIs allow you to create, read, modify, and delete Fedora digital objects. The REST and SOAP APIs below generally expose the same functionality, so you can choose the one that's most convenient to you.

- [REST API \(HTTP\)](#)
- [API-A and API-M \(SOAP\)](#)

Optional APIs

- [Resource Index \(RDF\) Search \(HTTP\)](#)
- [Basic \(Dublin Core Only\) OAI-PMH Provider \(HTTP\)](#)

Deprecated APIs

The following APIs have been replaced by the [REST API](#) and are deprecated as of Fedora 3.4. They still work, but will be removed in a future release.

- [API-A-LITE](#)
- [API-M-LITE](#)
- [Basic Search](#)

API-A

Introduction

The Fedora Access service defines an open interface for accessing digital objects. The access operations include methods to do reflection on a digital object (i.e., to discover the kinds of disseminations that are available on the object), and to request disseminations. The major function of the Fedora Access service is to fulfill a client's request for dissemination. To support disseminations, the underlying repository system must evaluate the services specified for a digital object, and figure out how to call it. The service may be internal to the repository, or it may be a web service external to the repository. The underlying repository system facilitates all external service bindings on behalf of the client, simply returning a dissemination result via the access service layer.

Methods

Repository Access

describeRepository

Gets information that describes the repository.

Returns:

- *RepositoryInfo*** *String repositoryName* - The name of the Repository. Set in fedora.fcfg. Default "Fedora Repository"
- *String repositoryVersion* - The version of Fedora running. Fedora 3.0 returns "3.0"
- *String repositoryBaseURL* - The repository base url set in fedora.fcfg. Default "http://localhost:8080/fedora"
- *String repositoryPIDNamespace* - The prefix to use for newly generated PIDs
- *String defaultExportFormat*
- *String OAINameSpace* - The oai namespace. Default "example.org"
- *String[] adminEmailList* - The email to the administrator. Default "bob@example.org" and "sally@example.org". Defined in fedora.fcfg.
- *String samplePID* - An example pid, to show how to refer to objects. "doms:100"
- *String sampleOAIIdentifier* - An example oai identifier, to show how to refer to records. Example: "oai:example.org:doms:100"
- *String sampleSearchURL* - The url to the search service for the repository. Default "http://localhost:8080/fedora/search"
- *String sampleAccessURL* - The url to an example object in the repository. Default "http://localhost:8080/fedora/get/demo:5"
- *String sampleOAIURL* - The url to an oai record. Default "http://localhost:8080/fedora/oai?verb=Identify"
- *String[] retainPIDs* - The list of pid prefixes, that cause the pid to not be autogenerated.

Object Access

findObjects

Lists the specified fields of each object matching the given criteria.

Input parameters:

- *String[] resultfields* The names of the fields to return.
- *int maxResults* The maximum number of results to return in one FieldSearchResult. Further results can be queried with the `resumeFindObjects` method.
- *FieldSearchQuery query*: The terms or conditions for the search. Either terms or conditions are used, not both.
 - *String terms*: The search terms
 - *Condition[] conditions*: The conditions on the results
 - *String property*: The property to condition. Possible fields are the same as for resultfields.
 - *Operator operator*: Possible values are: "has", "eq", "lt", "le", "gt" and "ge"
 - *String value*: The value the constrained property must adhere to

The possible values for *resultfield* are the following:

- Key fields: pid, label, state, ownerId, cDate, mDate, dcmDate
- Dublin core fields: title, creator, subject, description, publisher, contributor, date, format, identifier, source, language, relation, coverage, rights

Returns:

- *FieldSearchResult*** *ListSession listsession* The information necessary for resuming the search.
 - *String token*: The token to be used in `resumeFindObjects`
 - *int cursor*: The index of the first object in this resultlist, in the complete resultlist.
 - *int completeListSize*: The size of the complete resultlist.
 - *String expirationDate*: The expirationdate for the token. The last time when the search can be resumed.
- *ObjectFields[] resultlist*: the specified fields of each object matching the given criteria.
 - *String pid* Only set if the relevant field was set in resultfields
 - *String label* Only set if the relevant field was set in resultfields
 - *String state* Only set if the relevant field was set in resultfields
 - *String ownerId* Only set if the relevant field was set in resultfields
 - *String cDate* Only set if the relevant field was set in resultfields
 - *String mDate* Only set if the relevant field was set in resultfields
 - *String dcmDate* Only set if the relevant field was set in resultfields
 - *String[] title* Only set if the relevant field was set in resultfields
 - *String[] creator* Only set if the relevant field was set in resultfields
 - *String[] subject* Only set if the relevant field was set in resultfields
 - *String[] description* Only set if the relevant field was set in resultfields
 - *String[] publisher* Only set if the relevant field was set in resultfields
 - *String[] contributor* Only set if the relevant field was set in resultfields
 - *String[] date* Only set if the relevant field was set in resultfields
 - *String[] type* Only set if the relevant field was set in resultfields
 - *String[] format* Only set if the relevant field was set in resultfields

- *String[] identifier* Only set if the relevant field was set in resultfields
- *String[] source* Only set if the relevant field was set in resultfields
- *String[] language* Only set if the relevant field was set in resultfields
- *String[] relation* Only set if the relevant field was set in resultfields
- *String[] coverage* Only set if the relevant field was set in resultfields
- *String[] rights* Only set if the relevant field was set in resultfields

Note:

The only way to get the Object State (or OwnerID) is via a findObjects call. Use these parameters to query the state of a object.

- *String[] resultfields* = ["pid", "state"]
- *int maxResults* = null
- *FieldSearchQuery query*:
 - *Condition[] conditions*:
 - *String property*: "pid"
 - *Operator operator*: "eq"
 - *String value*: "The pid of the object you want to find"

resumeFindObject

Gets the next list of results from a truncated [findObjects](#) response.

Input parameters:

- *String token*: The token of the session in which the remaining results can be obtained.

Returns:

- *FieldSearchResult** ListSession listsession* The information necessary for resuming the search.
 - *String token*: The token to be used in resumeFindObject
 - *int cursor*: The index of the first object in this resultlist, in the complete resultlist.
 - *int completeListSize*: The size of the complete resultlist.
 - *String expirationDate*: The expirationdate for the token. The last time when the search can be resumed.
- *ObjectFields[] resultlist*: the specified fields of each object matching the given criteria.
 - *String pid* Only set if the relevant field was set in resultfields
 - *String label* Only set if the relevant field was set in resultfields
 - *String state* Only set if the relevant field was set in resultfields
 - *String ownerId* Only set if the relevant field was set in resultfields
 - *String cDate* Only set if the relevant field was set in resultfields
 - *String mDate* Only set if the relevant field was set in resultfields
 - *String dcmDate* Only set if the relevant field was set in resultfields
 - *String[] title* Only set if the relevant field was set in resultfields
 - *String[] creator* Only set if the relevant field was set in resultfields
 - *String[] subject* Only set if the relevant field was set in resultfields
 - *String[] description* Only set if the relevant field was set in resultfields
 - *String[] publisher* Only set if the relevant field was set in resultfields
 - *String[] contributor* Only set if the relevant field was set in resultfields
 - *String[] date* Only set if the relevant field was set in resultfields
 - *String[] type* Only set if the relevant field was set in resultfields
 - *String[] format* Only set if the relevant field was set in resultfields
 - *String[] identifier* Only set if the relevant field was set in resultfields
 - *String[] source* Only set if the relevant field was set in resultfields
 - *String[] language* Only set if the relevant field was set in resultfields
 - *String[] relation* Only set if the relevant field was set in resultfields
 - *String[] coverage* Only set if the relevant field was set in resultfields
 - *String[] rights* Only set if the relevant field was set in resultfields

getObjectHistory

Gets a list of timestamps that correspond to modification dates of components. This currently includes changes to Datastreams and disseminators.

Input parameters:

- *String pid* The pid of the object.

Returns:

- *String[]* An array containing the list of timestamps indicating when changes were made to the object.

getObjectProfile

Profile of an object, which includes key metadata fields and URLs for the object Dissemination Index and the object Item Index. Can be thought of as a default view of the object.

Input parameters:

- *String pid* The pid of the object.
- *String asOfDateTime* The date/time stamp specifying the desired version of the object. If null, the current version of the object (the most recent time) is assumed.

Returns:

- *ObjectProfile* Contains these fields
 - *String pid* The pid of the object
 - *String objLabel* The label of the object
 - *String[] objModels* The pids of the content models of the object
 - *String objCreateDate* The creation date
 - *String objLastModDate* The last modification time
 - *String objDissIndexViewURL* The REST url for the Dissemination index, as known from the built in search service
 - *String objItemIndexViewURL* The REST url for the Datastream index, as known from the built in search service

Note:

There are two general object properties, State and OwnerID, which are not part of the ObjectProfile. The way to get these are through the [findObjects](#) method.

Datastream Access

getDatastreamDissemination

Gets the content of a datastream.

Input parameters:

- *String pid* The PID of the object.
- *String dsID* The datastream ID.
- *String asOfDateTime* A date/Time indicating the version of the datastream to retrieve. If null, Fedora will use the most recent version.

Returns:

- *MIMETypedStream** String MIMETYPE* The mimetype of the stream
 - *byte[] stream* The contents of the Stream
 - *Property[] header* The header will be empty, or if applicable, contain the http header as name/value pairs.
 - *String name*
 - *String value*

listDatastreams

Lists the datastreams of an object.

Input parameters:

- *String pid* The pid of the object.
- *String asOfDateTime* The date/time stamp specifying the desired version of the object. If null, the current version of the object (the most recent time) is assumed.

Returns:

- *DatastreamDef[]* A datastream definition object, containing the following values
 - *String ID* The datastream id - "DC" for the DC datastream
 - *String label* The datastream label
 - *String MIMETYPE* The mimetype of the datastream, if any

Dissemination Access

getDissemination

Disseminates the content produced by executing the method specified in the service definition associated the specified digital object.

Input parameters:

- *String pid* The pid of the object.
- *String serviceDefinitionPid* The PID of the Service Definition object.
- *String methodName* The name of the method to be executed.
- *Property[] parameters* name-value pairs.
 - *String name*
 - *String value*

- *String asOfDateTime* The versioning date/time. If null, Fedora will use the most recent version.

Returns:

- *MIMETypedStream** String MIMETYPE* The mimetype of the stream
 - *byte[] stream* The contents of the Stream
 - *Property[] header* The header will be empty, or if applicable, contain the http header as name/value pairs.
 - *String name*
 - *String value*

listMethods

Lists all the methods that the object supports.

Each method can take a number of parameters. Each parameter for a method has a name, and a type. The possible values of a parameter depends on its type. It can be bound to a datastream in the object, it can have a hardcoded value or it can be defined by the caller.

Each parameter is defined to be passed by reference or passed by value.

Input parameters:

- *String pid* The pid of the object.
- *String asOfDateTime* The date/time stamp specifying the desired version of the object. If null, the current version of the object (the most recent time) is assumed.

Returns:

- *ObjectMethodDef[]** String PID* The pid of the data object
 - *String serviceDefinitionPID* The pid of the service definition object
 - *String methodName* the name of the method
 - *MethodParmDef[] methodParmDefs* An array of the method parameters
 - *String paramName* The name of the parameter.
 - *String parmType* The type of the parameter. Restricted to "fedora:datastreamInputType", "fedora:userInputType" or "fedora:defaultInputType"
 - *String parmDefaultValue* If the parmType is default, this is the value that will be used. Null if other type.
 - *String[] parmDomainValues* If the parameter can be defined by the user, these are the possible values. If Null, the parameter can take any value. Null if other type.
 - *boolean parmRequired* False, if this parameter can be left out of a call.
 - *String parmLabel* The label for the parameter. Can be null.
 - *String parmPassBy* The method of passing the parameter. Restricted to "URL_REF" (if the parameter is pass by reference - by an url) and "VALUE" (if the parameter is pass by value)
 - *String asOfDate* The timestamp/version of the method definition

WSDL

When running your own Fedora server, the API-A WSDL is available at **/wsdl?api=API-A**.

Example:

```
http://localhost:8080/fedora/wsdl?api=API-A
```



Warning

The Apache Axis library also automatically provides an INCORRECT WSDL DOCUMENT at `/fedora/services/access?wsdl`. Please DO NOT USE this; it is automatically generated and will result in unexpected behavior for your clients. The recommended URL, `/fedora/wsdl?api=API-A` produces a copy of the API-A WSDL exactly as it was intended.

API-A-LITE



Deprecation of "LITE" APIs

As of release 3.4-RC1 of Fedora, the "LITE" APIs are deprecated. You are encouraged to migrate any existing code to use the new REST API. The "LITE" APIs will be removed in a future release of Fedora.

- [Introduction](#)
- [Client Syntax](#)
 - [describeRepository](#)
 - [getDatastreamDissemination](#)
 - [getDissemination](#)

- getObjectHistory
 - getObjectProfile
 - findObjects
- resumeFindObjects
 - listDatastreams
 - listMethods
- WSDL

Introduction

The Fedora API-A-Lite interface is implemented as an HTTP service that consists of the following methods:

- **describeRepository** – provides information about the Fedora repository server, including name, version, base URL
- **getDatastreamDissemination** – gets the specified datastream's contents
- **getDissemination** – gets the specified dissemination request
- **getObjectProfile** – gets the Object Profile of the specified object
- **getObjectHistory** – gets the change history of the specified object
- **findObjects** (search) – performs a search on the repository given specified search criteria and returns a result set for matching objects
- **listDatastreams** – gets a list of datastreams for the specified object
- **listMethods** – gets a list of disseminator methods for the specified object
- **resumeFindObjects** (search) – gets the next set of items ("hits") in a search result set. Used when a prior findObjects request specified that the result set should be returned with a maximum number of hits at a time.

For more information on the method definitions, refer to the API descriptions located at <http://www.fedora.info/definitions/1/0/api/>

Client Syntax



URL encoding of parameters

Note that the LITE APIs (unlike the new [REST API](#)) do not require, and do not accept, URL-encoding of parameters within the path. Therefore PIDs should be specified as the raw PID (namespace:name) value, and should not be encoded.

describeRepository

Syntax:

```
http://hostname:port/fedora/describe*[{xml=BOOLEAN}]*
```

This syntax requests information about a Fedora repository, including repository name, version, base URL, PID syntax, OAI identifier syntax, admin emails, and sample request URLs. The `xml` parameter determines the type of output returned. If the parameter is omitted or has a value of "false", a MIME-typed stream consisting of an HTML table is returned providing a browser-savvy means of viewing the object profile. If the value specified is "true", then a MIME-typed stream consisting of XML is returned.

- **hostname** – required hostname of the Fedora server.
- **port** – required port number on which the Fedora server is running.
- **fedora** – required name of the Fedora web application.
- **describe** – a required parameter specifying the Fedora servlet path for the describe request.
- **xml** – an optional parameter indicating the requested output format. A value of "true" indicates a return type of text/xml; the absence of the xml parameter or a value of "false" indicates format is to be text/html.

Example:

Get repository information using the describe request as XML:

```
http://localhost:8080/fedora/describe?xml=true
```

getDatastreamDissemination

Syntax:

```
http://hostname:port/fedora/get/PID/DSPID*[/dateTime]*
```

This syntax requests a dissemination of the specified datastream within the specified object. The result is returned as a MIME-typed stream.

- **hostname** – required hostname of the Fedora server.
- **port** – required port number on which the Fedora server is running.

- fedora – required name of the Fedora web application.
- get – a required parameter specifying the Fedora servlet path.
- PID – required persistent identifier of the digital object.
- DSPID – required identifier of the Datastream.
- **dateTime** – optional value indicating dissemination of a version of the digital object at the specified point in time. Proper syntax is YYYY-MM-DDTHH:MM:SS.SSSZ where HH is 24-hour clock and SSS is milliseconds.

Examples:

Get the Dublin Core (DC) Datastream of demo object demo:5:

`http://localhost:8080/fedora/get/demo:5/DC`

Get the thumbnail Datastream (DS1) of demo object demo:5:

`http://localhost:8080/fedora/get/demo:5/THUMBRES_IMG`

getDissemination

Syntax:

`http://hostname:port/fedora/get/PID/sDefPID/methodName*[/dateTime][?parmArray]*`

This syntax requests a dissemination of the specified object using the specified method of the associated service definition object. The result is returned as a MIME-typed stream.

- hostname – required hostname of the Fedora server.
- port – required port number on which the Fedora server is running.
- fedora – required name of the Fedora web application.
- get – a required parameter specifying the Fedora servlet path.
- PID – required persistent identifier of the digital object.
- sDefPID – required persistent identifier of the service definition object which defines the service.
- methodName – required name of the method to be executed.
- **dateTime** – optional value indicating dissemination of a version of the digital object at the specified point in time. Proper syntax is YYYY-MM-DDTHH:MM:SS.SSSZ where HH is 24-hour clock and SSS is milliseconds.
- **parmArray** - optional array of method parameters consisting of name/value pairs in the form parm1=value1&parm2=value2...

Example:

Get the Dissemination for a data object with a PID of demo:5 and associated service definition object with a PID of demo:1 and methodName of getThumbnail:

`http://localhost:8080/fedora/get/demo:5/demo:1/getThumbnail`

getObjectHistory

Syntax:

`http://hostname:port/fedora/getObjectHistory/PID*[/?xml=BOOLEAN]*`

This syntax requests the change history for the specified digital object. The change history provides a list of timestamps that indicate when components (e.g., Datastreams, etc.) in the digital object were created or modified. These timestamps can be used in a dissemination request to view the object as it existed at a specific point in time. The xml parameter determines the type of output returned. If the parameter is omitted or has a value of "false", a MIME-typed stream consisting of an html table is returned providing a browser-savvy means of viewing the object history. If the value specified is "true", then a MIME-typed stream consisting of XML is returned.

- hostname – required hostname of the Fedora server.
- port – required port number on which the Fedora server is running.
- fedora – required name of the Fedora web application.
- getObjectHistory – a required parameter specifying the Fedora servlet path.
- PID – required persistent identifier of the digital object.
- **xml** – an optional parameter indicating the requested output format. A value of "true" indicates a return type of text/xml; the absence of the xml parameter or a value of "false" indicates format is to be text/html.

Examples:

Get the object history of demo object demo:5 and display the results as html:

```
http://localhost:8080/fedora/getObjectHistory/demo:5
```

Get the object history of demo object demo:5 and display the results as xml:

```
http://localhost:8080/fedora/getObjectHistory/demo:5?xml=true
```

getObjectProfile

Syntax:

```
http://hostname:port/fedora/get/PID*[/dateTime][?xml=BOOLEAN]*
```

This syntax requests an object profile for the specified digital object. The `xml` parameter determines the type of output returned. If the parameter is omitted or has a value of "false", a MIME-typed stream consisting of an HTML table is returned providing a browser-savvy means of viewing the object profile. If the value specified is "true", then a MIME-typed stream consisting of XML is returned.

- `hostname` – required hostname of the Fedora server.
- `port` – required port number on which the Fedora server is running.
- `fedora` – required name of the Fedora web application.
- `get` – a required parameter specifying the Fedora servlet path.
- `PID` – required persistent identifier of the digital object.
- `dateTime` – optional value indicating dissemination of a version of the digital object at the specified point in time. Proper syntax is YYYY-MM-DDTHH:MM:SS.SSSZ where HH is 24-hour clock and SSS is milliseconds.
- `xml` – an optional parameter indicating the requested output format. A value of "true" indicates a return type of text/xml; the absence of the `xml` parameter or a value of "false" indicates format is to be text/html.

Examples:

Get the ObjectProfile for a data object with a PID of demo:5 as HTML:

```
http://localhost:8080/fedora/get/demo:5
```

Get the ObjectProfile for a data object with a PID of demo:5 as XML:

```
http://localhost:8080/fedora/get/demo:5?xml=true
```

findObjects

Syntax:

```
http://hostname:port/fedora/search?  
[terms=TERMS][query=QUERY][&maxResults=MAXRESULTS][&xml=true][&pid=true][&label=true][&state=true][&ownerId=true]
```

This syntax essentially performs a search upon the objects in the repository. It finds objects that meet the criteria specified in the request. The criteria are evaluated against an index of the repository that contains unqualified Dublin Core and Fedora-specific metadata elements. The syntax provides a client with the ability to specify the search criteria as either a phrase (a simple keyword search), or as a set of name value pairs (a field-based search).

- `hostname` – required hostname of the Fedora server.
- `port` – required port number on which the Fedora server is running.
- `fedora` – required name of the Fedora web application.
- `search` – a required parameter specifying the Fedora servlet path for the findObjects request.
- `terms` – a phrase represented as a sequence of characters (including the ? and * wildcards) for the search. If this sequence is found in any of the fields for an object, the object is considered a match. Do NOT use this parameter in combination with the "query" parameter.
- `query` – a sequence of space-separated conditions. A condition consists of a metadata element name followed directly by an operator, followed directly by a value. Valid element names are (pid, label, state, ownerId, cDate, mDate, dcmDate, title, creator, subject, description, publisher, contributor, date, type, format, identifier, source, language, relation, coverage, rights). Valid operators are: contains (-), equals (=), greater than (>), less than (<), greater than or equals (>=), less than or equals (<=). The contains (-) operator may be

used in combination with the ? and * wildcards to query for simple string patterns. Space-separators should be encoded in the URL as %20. Operators must be encoded when used in the URL syntax as follows: the (=) operator must be encoded as %3D, the (>) operator as %3E, the (<) operator as %3C, the (>=) operator as %3E%3D, the (<=) operator as %3C%3D, and the (-) operator as %7E. Values may be any string. If the string contains a space, the value should begin and end with a single quote character ('). If all conditions are met for an object, the object is considered a match. Do NOT use this parameter in combination with the "terms" parameter. See example URLs at the end of this document for usage.

- **maxResults** – the maximum number of results that the server should provide at once. If this is unspecified, the server will default to a small value.
- **xml** – whether to return the result as an xml document. If this is given as true, the result will be in xml. Otherwise, the result will be provided in a simple html document.
- **pid** – if true, the Fedora persistent identifier (PID) element of matching objects will be included in the response.
- **label** – if true, the Fedora object label element of matching objects will be included in the response.
- **state** – if true, the Fedora object state element of matching objects will be included in the response.
- **ownerId** – if true, each matching objects' owner id will be included in the response.
- **cDate** – if true, the Fedora create date element of matching objects will be included in the response.
- **mDate** – if true, the Fedora modified date of matching objects will be included in the response.
- **dcmDate** – if true, the Dublin Core modified date element(s) of matching objects will be included in the response.
- **title** – if true, the Dublin Core title element(s) of matching objects will be included in the response.
- **creator** – if true, the Dublin Core creator element(s) of matching objects will be included in the response.
- **subject** – if true, the Dublin Core subject element(s) of matching objects will be included in the response.
- **description** – if true, the Dublin Core description element(s) of matching objects will be included in the response.
- **publisher** – if true, the Dublin Core publisher element(s) of matching objects will be included in the response.
- **contributor** – if true, the Dublin Core contributor element(s) of matching objects will be included in the response.
- **date** – if true, the Dublin Core date element(s) of matching objects will be included in the response.
- **type** – if true, the Dublin Core type element(s) of matching objects will be included in the response.
- **format** – if true, the Dublin Core format element(s) of matching objects will be included in the response.
- **identifier** – if true, the Dublin Core identifier element(s) of matching objects will be included in the response.
- **source** – if true, the Dublin Core source element(s) of matching objects will be included in the response.
- **language** – if true, the Dublin Core language element(s) of matching objects will be included in the response.
- **relation** – if true, the Dublin Core relation element(s) of matching objects will be included in the response.
- **coverage** – if true, the Dublin Core coverage element(s) of matching objects will be included in the response.
- **rights** – if true, the Dublin Core right element(s) of matching objects will be included in the response.

Examples:

Find objects in the repository that are indexed with the keyword "fedora." The result set should provide the PID and Dublin Core title elements for each object:

```
http://localhost:8080/fedora/search?terms=fedora&pid=true&title=true
```

Find objects in the repository where the Dublin Core title contains the word "Rome" and the Dublin Core creator contains the word "Staples". The result set should provide the PID, plus the Dublin Core creator and title elements for each object:

```
http://localhost:8080/fedora/search?query=title%7ERome%20creator%7EStaples&pid=true&title=true&creator=true
```

Find objects in the repository whose PID ends with the number 1. The result set should provide a max of 50 hits at a time, and it should provide the PID and Dublin Core title element for each object. The result set should be returned as xml:

```
http://localhost:8080/fedora/search?query=pid%7E*1&maxResults=50&xml=true&pid=true&title=true
```

resumeFindObjects

Syntax:

```
http://hostname:port/fedora/search*?sessionToken=SESSIONID[&xml=BOOLEAN]*
```

This syntax requests the next members of a result set from a prior invocation of a findObjects request. If the findObjects request was run and there are more "hits" in the result set than the maxResults setting on the findObjects request, then the resumeFindObjects request is used to obtain the next group of items in the result set. Fields to be included as part of the result set are the same as those available for findObjects, and must be specified as part of the syntax for resumeFindObjects as well (i.e. if the parameters ?pid=true&title=true are part of the findObjects query, those same parameters should be included as part of the resumeFindObjects query.)

- **hostname** – required hostname of the Fedora server.
- **port** – required port number on which the Fedora server is running.
- **fedora** – required name of the Fedora web application.
- **search** – a required parameter specifying the Fedora servlet path for the findObjects request.

- **sessionToken** – the identifier of the session to which the search results are being returned.
- **xml** – boolean indicating whether to return the search results as xml. If this is given as true, the result will be in xml. Otherwise, the result will be provided in a simple HTML document.

listDatastreams

Syntax:

```
http://hostname:port/fedora/listDatastreams/PID*[/dateTime][?xml=BOOLEAN]*
```

This syntax requests a list of Datastreams contained in the digital object.

- **hostname** – required hostname of the Fedora server.
- **port** – required port number on which the Fedora server is running.
- **fedora** – required name of the Fedora web application.
- **listDatastreams** – a required parameter specifying the Fedora servlet path.
- **PID** – required persistent identifier of the digital object.
- **dateTime** – optional value indicating dissemination of a version of the digital object at the specified point in time. Proper syntax is YYYY-MM-DDTHH:MM:SS.SSSZ where HH is 24-hour clock and SSS is milliseconds.
- **xml** – an optional parameter indicating the requested output format. A value of "true" indicates a return type of text/xml; the absence of the xml parameter or a value of "false" indicates format is to be text/html.

Examples:

List the Datastreams in a data object with PID of demo:5 as HTML:

```
http://localhost:8080/fedora/listDatastreams/demo:5
```

List the Datastreams in a data object with PID of demo:5 as XML:

```
http://localhost:8080/fedora/listDatastreams/demo:5?xml=true
```

listMethods

Syntax:

```
http://hostname:port/fedora/listMethods/PID*[/dateTime][?xml=BOOLEAN]*
```

This syntax requests a list of methods available in the digital object.

- **hostname** – required hostname of the Fedora server.
- **port** – required port number on which the Fedora server is running.
- **fedora** – required name of the Fedora web application.
- **listMethods** – a required parameter specifying the Fedora servlet path.
- **PID** – required persistent identifier of the digital object.
- **dateTime** – optional value indicating dissemination of a version of the digital object at the specified point in time. Proper syntax is YYYY-MM-DDTHH:MM:SS.SSSZ where HH is 24-hour clock and SSS is milliseconds.
- **xml** – an optional parameter indicating the requested output format. A value of "true" indicates a return type of text/xml; the absence of the xml parameter or a value of "false" indicates format is to be text/html.

Examples:

List the methods for a data object with PID of demo:5 as HTML:

```
http://localhost:8080/fedora/listMethods/demo:5
```

List the methods for a data object with PID of demo:5 as XML:

```
http://localhost:8080/fedora/listMethods/demo:5?xml=true
```

WSDL

When running your own Fedora server, the API-A-LITE WSDL is available at `/wsdl?api=API-A-LITE`.

Example:

```
http://localhost:8080/fedora/wsdl?api=API-A-LITE
```

API-M

- Introduction
- Methods
 - Datastream Management
 - addDatastream
 - compareDatastreamChecksum
 - getDatastream
 - getDatastreamHistory
 - getDatastreams
 - modifyDatastreamByReference
 - modifyDatastreamByValue
 - setDatastreamState
 - setDatastreamVersionable
 - purgeDatastream
 - Relationship Management
 - addRelationship
 - getRelationships
 - purgeRelationship
 - Object Management
 - modifyObject
 - purgeObject
 - export
 - getNextPID
 - getObjectXML
 - ingest
 - validate
- WSDL

Introduction

The Fedora Management service defines an open interface for administering the repository, including creating, modifying, and deleting digital objects, or components within digital objects. The Management service interacts with the underlying repository system to read content from and write content to the digital object and datastream storage areas. The Management service exposes a set of operations that enable a client to view and manipulate digital objects from an abstract perspective, meaning that a client does not need to know anything about underlying storage formats, storage media, or storage management schemes for objects. Also, the underlying repository system handles the details of storing datastream content within the repository, as well as mediating connectivity for datastreams that reference external content.



Ensure DC, RELS-EXT and RELS-INT are versionable if using Managed Content

Due to an outstanding bug [FCREPO-849](#), if you use Managed Content for DC, RELS-EXT or RELS-INT then please make sure these datastreams are versionable (the default setting for versionable is "true", so if you haven't specified this datastream property then you are safe). Ensure that you don't inadvertently set this property to "false" for these datastreams when using the API methods.



Deprecation of "force" parameter

The "force" parameter on modify/purge operations is deprecated, and will be removed in a future release. This parameter has never been used in practice.

Methods

Datastream Management

addDatastream

Creates a new Datastream in the object.

Input parameters:

- *String pid* The PID of the object.

- *String dsID* The datastream ID (64 characters max). If null, Fedora will generate the value.
- *String[] altIDs* Alternate identifiers for the datastream. Can be null.
- *String dsLabel* The label for the datastream. Can be null.
- *boolean versionable* Enable versioning of the datastream.
- *String MIMEType* The mime-type of the datastream. Can be null.
- *String formatURI* The format URI of the datastream. Can be null
- *String dsLocation* Location of managed, redirect, or external referenced datastream content.
- *String controlGroup* One of "X", "M", "R", or "E" (Inline XML, Managed Content, Redirect, or External Referenced). Required
- *String dsState* One of "A", "D", or "I" (Active, Deleted, or Inactive).
- *String checksumType* The algorithm used to compute the checksum. Possible values are MD5, SHA-1, SHA-256, SHA-386, SHA-512, DISABLED
- *String checksum* The value of the checksum represented as a hexadecimal string. Can be null.
- *String logMessage* A log message.

Returns:

- *String* The datastreamID of the newly added datastream.

compareDatastreamChecksum

Verifies that the Datastream content has not changed since the checksum was initially computed.

Input parameters:

- *String pid* The PID of the object.
- *String dsID* The datastream ID.
- *String versionDate* A date/time indicating the version of the datastream to verify. If null, Fedora will use the most recent version.

Returns:

- *String* The checksum if there is no difference, a message indicating checksum failure otherwise.

getDatastream

Gets the specified datastream.

Input parameters:

- *String pid* The pid of the object.
- *String dsID* The datastream ID.
- *String asOfDateTime* The date/time stamp specifying the desired version of the object. If null, the current version of the object (the most recent time) is assumed.

Returns:



Managed Content datastream size

Since Fedora 3.4, the size of datastreams is returned for inline XML and managed content datastreams (but not for external or referenced content). Prior to this the size is valid only for inline XML datastreams. Please note that a valid size for managed content datastreams will only be returned for datastreams that are new or have been modified after installing Fedora 3.4. Datastreams created in earlier versions will continue to return a size of zero. A tool for calculating and storing the size of these datastreams will be provided in a future release.

- *Datastream:*
 - *DatastreamControlGroup controlGroup* - String restricted to the values of "X", "M", "R", or "E" (InlineXML,Managed Content,Redirect, or External Referenced).
 - *String ID* - The datastream ID (64 characters max).
 - *String versionID* - The ID of the most recent datastream version
 - *String[] altIDs* - Alternative IDs for the datastream, if any.
 - *String label* - The Label of the datastream.
 - *boolean versionable* - Whether the datastream is versionable.
 - *String MIMEType* - The mime-type for the datastream, if set.
 - *String formatURI* - The format uri for the datastream, if set.
 - *String createDate* - The date the first version of the datastream was created.
 - *long size* - The size of the datastream in Fedora. Only valid for inline XML metadata and managed content datastreams.
 - *String state* - The state of the datastream. Will be "A" (active), "I" (inactive) or "D" (deleted).
 - *String location* - If the datastream is an external reference or redirect, the url to the contents. TODO: Managed?
 - *String checksumType* - The algorithm used to compute the checksum. One of "DEFAULT", "DISABLED", "MD5", "SHA-1", "SHA-256", "SHA-385", "SHA-512".
 - *String checksum* - The value of the checksum represented as a hexadecimal string.

getDatastreamHistory

Gets all versions of a datastream, sorted from most to least recent.

Input parameters:

- *String pid* The pid of the object.
- *String dsID* The datastream ID

Returns:

- *Datastream[]* See [getDatastream](#)

getDatastreams

Gets all datastreams in the object

Input parameters:

- *String pid* The pid of the object.
- *String asOfDateTime* The date/time stamp specifying the desired version of the object. If null, the current version of the object (the most recent time) is assumed.
- *String dsState* One of "A", "D", or "I" (Active, Deleted, or Inactive).

Returns:

- *Datastream[]* See [getDatastream](#)

modifyDatastreamByReference

Change the referenced location for a datastream. This operation is only relevant for managed, redirect or external reference datastreams (controlgroup E,M,R).

Input parameters:

- *String pid* The PID of the object.
- *String dsID* The datastream ID.
- *String[] altIDs* Alternate identifiers for the datastream, if any.
- *String dsLabel* The label for the datastream.
- *String MIMEType* The mime type.
- *String formatURI* Optional format URI of the datastream.
- *String dsLocation* Location of managed, redirect, or external referenced datastream content.
- *String checksumType* The algorithm used to compute the checksum. One of "DEFAULT", "DISABLED", "MD5", "SHA-1", "SHA-256", "SHA-385", "SHA-512".
- *String checksum* The value of the checksum represented as a hexadecimal string.
- *String logMessage* A log message.
- *boolean force* ~~Deprecated. Force the update even if it would break a data contract.~~

Returns:

- *String* The timestamp of the operation according to the server, in ISO8601 format.

modifyDatastreamByValue

Modifies an existing Datastream in an object, by value. This operation is only valid for Inline XML Datastreams (i.e. controlGroup "X").

Input parameters:

- *String pid* The PID of the object.
- *String dsID* The datastream ID.
- *String[] altIDs* Alternate identifiers for the datastream, if any.
- *String dsLabel* The label for the datastream.
- *String MIMEType* The mime type.
- *String formatURI* Optional format URI of the datastream.
- *byte[] dsContent* The content of the datastream.
- *String checksumType* The algorithm used to compute the checksum. One of "DEFAULT", "DISABLED", "MD5", "SHA-1", "SHA-256", "SHA-385", "SHA-512".
- *String checksum* The value of the checksum represented as a hexadecimal string.
- *String logMessage* A log message.
- *boolean force* ~~Deprecated. Force the update even if it would break a data contract.~~

Returns:

- *String* The timestamp of the operation according to the server, in ISO8601 format.

setDatastreamState

Sets the state of a Datastream to the specified state value.

Input parameters:

- *String pid* The PID of the object.
- *String dsID* The datastream ID.
- *String dsState* One of "A", "D", or "I" (Active, Deleted, or Inactive).
- *String logMessage* A log message.

Returns:

- *String* The timestamp of the operation according to the server, in ISO8601 format.

setDatastreamVersionable

Selectively turn versioning on or off for selected datastream. When versioning is disabled, subsequent modifications to the datastream replace the current datastream contents and no versioning history is preserved. To put it another way: No new datastream versions will be made, but all the existing versions will be retained. All changes to the datastream will be to the current version.

Input parameters:

- *String pid* The PID of the object.
- *String dsID* The datastream ID.
- *Boolean versionable* Enable versioning of the datastream.
- *String logMessage* A log message.

Returns:

- *String* The timestamp of the operation according to the server, in ISO8601 format.

purgeDatastream

Permanently removes one or more versions of a Datastream from an object.

Input parameters:

- *String pid* The PID of the object.
- *String dsID* The datastream ID.
- *String startDT* The (inclusive) start date-time stamp of the range. If null, this is taken to be the lowest possible value, and thus, the entire version history up to the endDT be purged.
- *String endDT* The (inclusive) ending date-time stamp of the range. If null, this is taken to be the greatest possible value, and thus, the entire version history back to the startDT will be purged.
- *String logMessage* A log message.
- *Boolean force* ~~Deprecated. Force the update even if it would break a data contract.~~

Returns:

- *String* The timestamp of the operation according to the server, in ISO8601 format.

Relationship Management



pid parameter changes

From version 3.3 onwards, the "pid" parameter of the relationships methods has been changed to "subject". The parameter identifies the subject of a relationship and should be either a Fedora object URI (eg info:fedora/demo:333) or a Fedora object's datastream URI (eg info:fedora/demo:333/DS1). Subjects that identify a Fedora object will result in operations on RELS-EXT, and subjects that identify a datastream will result in operations on RELS-INT. The pid format for a subject (eg demo:333) is still accepted, but is deprecated as of version 3.3.

WSDL note: The WSDL for versions 3.3 onwards continues to identify this parameter as "pid" to avoid breaking any existing SOAP clients. The parameter is interpreted as a relationship's subject as above. The SOAP parameter name will be changed to "subject" in a future release.

The relationship management methods manipulate the content of the RELS-EXT and RELS-INT datastreams. The datastream to be modified is determined from the subject of the relationship, ie a subject of info:fedora/demo:333 will result in changes to RELS-EXT in demo:333, and a subject of info:fedora/demo:333/DS1 will result in changes to RELS-INT in demo:333. These modifications will also be propagated to the Resource Index if it is enabled.

addRelationship

Creates a new relationship in the object. Adds the specified relationship to the object's RELS-EXT or RELS-INT Datastream. If the Resource

Index is enabled, the relationship will be added to the Resource Index.

A rdf tuple consist of an object or datastream (the subject), having a predicate relating it to a target (the object). The object can either be a literal value, or a URI (which can identify for example a Fedora object or a datastream).

Input parameters:

- *String subject* The subject. Either a Fedora object URI (eg info:fedora/demo:333) or a datastream URI (eg info:fedora/demo:333/DS1).
- *String relationship* The predicate.
- *String object* The object (target).
- *boolean isLiteral* A boolean value indicating whether the object is a literal.
- *String datatype* The datatype of the literal. Optional.

Returns:

- *boolean* True if and only if the relationship was added.

getRelationships

Get the relationships asserted in the object's RELS-EXT or RELS-INT Datastream that match the given criteria.

Input parameters:

- *String subject* The subject. Either a Fedora object URI (eg info:fedora/demo:333) or a datastream URI (eg info:fedora/demo:333/DS1).
- *String relationship* The predicate to match. A null value matches all predicates.

Returns:

- *RelationshipTuple[]** *String subject* - The subject of the relation. Either a Fedora object URI (eg info:fedora/demo:333) or a datastream URI (eg info:fedora/demo:333/DS1).
 - *String predicate* - The predicate relating the subject and the object. Includes the namespace of the relation.
 - *String object* - The URI of the object (target) of the relation
 - *boolean isLiteral* - If true, the subject should be read as a literal value, not a URI
 - *String datatype* - If the subject is a literal, the datatype to parse the value as. Optional.

purgeRelationship

Delete the specified relationship. This method will remove the specified relationship(s) from the RELS-EXT or RELS-INT datastream. If the Resource Index is enabled, this will also delete the corresponding triples from the Resource Index.

Input parameters:

- *String subject* The subject. Either a Fedora object URI (eg info:fedora/demo:333) or a datastream URI (eg info:fedora/demo:333/DS1).
- *String relationship* The predicate, null matches any predicate.
- *String object* The object, null matches any object.
- *boolean isLiteral* A boolean value indicating whether the object is a literal.
- *String datatype* The datatype of the literal. Optional.

Returns:

- *boolean* True if and only if the relationship was purged.

Object Management

modifyObject

Modify an object.

Input parameters:

- *String pid* The PID of the object.
- *String state* The new state, "A", "I" or "D". Null leaves unchanged
- *String ownerId* The ownerId for the object.
- *String logMessage* A log message.

Returns:

- *String* The timestamp of the operation according to the server, in ISO8601 format.

purgeObject

Permanently removes an object from the repository.

Parameters:

Input parameters:

- *String pid* The PID of the object.
- *Boolean force* Force the purge, even if it would break a dependency
- *String logMessage* A log message.

Returns:

- *String* The timestamp of the operation according to the server, in ISO8601 format.

export

Exports the entire digital object in the specified XML format, and encoded appropriately for the specified export context.

Parameters:

- *pid*: The pid of the object.
- *format*: The XML format to export, one of "info:fedora/fedora-system:FOXML-1.1", "info:fedora/fedora-system:FOXML-1.0", "info:fedora/fedora-system:METSFedoraExt-1.1", "info:fedora/fedora-system:METSFedoraExt-1.0", "info:fedora/fedora-system:ATOM-1.1", or "info:fedora/fedora-system:ATOMZip-1.1".
- *context*: The export context, which determines how datastream URLs and content are represented. One of "public", "migrate", or "archive". Returns: the digital object in the requested XML format.

getNextPID

Retrieves the specified number of next available pid(s) for a given pid namespace.

Parameters:

- *numPIDs*: The number of pids to retrieve.
- *pidNamespace*: The namespace of the requested pid(s). Returns: An array of the requested next available pid(s).

getObjectXML

Gets the serialization of the digital object to XML appropriate for persistent storage in the repository, ensuring that any URLs that are relative to the local repository are stored with the Fedora local URL syntax. The Fedora local URL syntax consists of the string "local.fedora.server" standing in place of the actual "hostname:port" on the URL. Managed Content (M) datastreams are stored with internal identifiers in dsLocation. Also, within selected inline XML datastreams (i.e., WSDL and SERVICE_PROFILE) any URLs that are relative to the local repository will also be stored with the Fedora local URL syntax.

Parameters:

- *pid*: The PID of the object. Returns: The digital object in Fedora's internal storage format.

ingest

Creates a new digital object in the repository. If the XML document does not specify the PID attribute of the root element, the repository will generate and return a new pid for the object resulting from this request. That pid will have the namespace of the repository. If the XML document specifies a pid, it will be assigned to the digital object provided that 1. it conforms to the Fedora pid Syntax, 2. it uses a namespace that matches the "retainPIDs" value configured for the repository, and 3. it does not collide with an existing pid of an object in the repository.

Parameters:

- *byte[] objectXML* The digital object in an XML submission format.
- *String format* The XML format of objectXML, one of "info:fedora/fedora-system:FOXML-1.1", "info:fedora/fedora-system:FOXML-1.0", "info:fedora/fedora-system:METSFedoraExt-1.1", "info:fedora/fedora-system:METSFedoraExt-1.0", "info:fedora/fedora-system:ATOM-1.1", or "info:fedora/fedora-system:ATOMZip-1.1".
- *String logMessage* A log message.

Returns:

- *String* The pid of the newly created object.
- *String asOfDateTime* A dateTime indicating the version of the datastream to verify. If null, Fedora will use the most recent version.

validate

Validates an object against its content models

Input parameters:

- *String pid* The PID of the object.

- *String asOfDateTime* indicates that the result should be relative to the digital object and the repository as it existed at the given date and time. Formatted like yyyy-MM-dd or yyyy-MM-ddTHH:mm:ssZ. If null is given, the most recent version will be used.

Returns:

- *Validation* The validation datastructure
 - *String pid* The pid of the object
 - *Boolean valid* true if the object was valid
 - *Calendar asOfDateTime* equals the asOfDateTime that was given as a parameter
 - *String[] contentModels* The content models of the object
 - *String[] objectProblems* List of problems with the object itself
 - *DatastreamProblem[] datastreamProblems* list of datastream problems
 - *String datastreamID* The datastream ID
 - *String[] problem* List of problems with this datastream

WSDL

When running your own Fedora server, the API-M WSDL is available at `/wsdl?api=API-M`.

Example:

```
http://localhost:8080/fedora/wsdl?api=API-M
```



Warning

The Apache Axis library also automatically provides an INCORRECT WSDL DOCUMENT at `/fedora/services/management?wsdl`. Please DO NOT USE this; it is automatically generated and will result in unexpected behavior for your clients. The recommended URL, `/fedora/wsdl?api=API-M` produces a copy of the API-M WSDL exactly as it was intended.

API-M-LITE



Deprecation of "LITE" APIs

As of release 3.4-RC1 of Fedora, the "LITE" APIs are deprecated. You are encouraged to migrate any existing code to use the new REST API. The "LITE" APIs will be removed in a future release of Fedora.

- [Introduction](#)
- [Client Syntax](#)
 - [getNextPID](#)
 - [upload](#)
- [WSDL](#)

Introduction

The Fedora API-M-Lite interface is implemented as an HTTP service that provides a simple URI-oriented definition for the Management service of a Fedora repository. It is intended to support a REST-like style of access to the Fedora Management web service (in contrast to a traditional SOAP web service definition). This means that we define a simple URL syntax that can be used to issue management service requests. API-M-LITE does not provide service bindings for all of the operations defined in Fedora's full Management service (API-M). Currently, API-M-LITE only provides bindings for the method listed below. Additional methods will be implemented for API-M-Lite in future releases of Fedora.

- `getNextPID` - gets a list of the requested next available PIDs.

For more information on the method definitions, refer to the API descriptions located at <http://www.fedora.info/definitions/1/0/api/>

Client Syntax

getNextPID

Syntax:

```
http://hostname:port/fedora/management/getNextPID? [numPIDs=NUMPIDS&] [namespace=NAMESPACE&] [xml=BOOLEAN&]
```

This syntax requests a list of the next available PIDs. The result is returned as a MIME-typed stream. Items enclosed in square brackets are

optional.

- hostname – required hostname of the Fedora server.
- port – required port number on which the Fedora server is running.
- fedora – a required parameter specifying the Fedora servlet path.
- management – a required parameter specifying the Fedora management servlet path.
- numPIDs – an optional parameter specifying the number of PIDs to generate. If omitted, the value defaults to one.
- namespace – an optional parameter specifying the namespace to be used in generating the PIDs. If omitted, the value defaults to the value of the pidNamespace parameter in the fedora.fcfg configuration file.
- xml – an optional parameter indicating the format of the response. A value of "false" indicates the response format is HTML. A value of "true" indicates the response format is XML. If omitted, the value defaults to a value of "false".

Examples:

Generate a single PID using the default namespace configured in the fedora.fcfg file and return the results as HTML:

```
http://localhost:8080/fedora/management/getNextPID?
```

Generate five PIDs using the default namespace configured in the fedora.fcfg file and return the results as HTML:

```
http://localhost:8080/fedora/management/getNextPID?numPIDs=5
```

Generate 5 PIDs using a namespace value of "my-namespace" and return the results as HTML:

```
http://localhost:8080/fedora/management/getNextPID?numPIDs=5&namespace=my-namespace
```

Generate 5 PIDs using the namespace value of "my-namespace" and return the results as XML:

```
http://localhost:8080/fedora/management/getNextPID?numPIDs=5&namespace=my-namespace&xml=true
```

upload

Syntax:

```
http://hostname:port/fedora/management/upload?file=(file)
```

This syntax requests an upload of a file to the Fedora server. The result is a temporary plain text URI which can be used to reference the uploaded file within subsequent API-M method calls, in which case Fedora will resolve the URI to the uploaded file.

An upload request **must** use HTTP POST for submission.

The uploaded file will only be available on the server for a short time, the default timeout being five minutes. You can set a higher value by adding the "uploadStorageMinutes" parameter to your `fedora.fcfg`. This parameter goes in the Management module's configuration section, and specifies the number of minutes after which uploaded content will be automatically deleted if not used.

- hostname – required hostname of the Fedora server.
- port – required port number on which the Fedora server is running.
- fedora – a required parameter specifying the Fedora servlet path.
- management – a required parameter specifying the Fedora management servlet path.
- file – a multipart encoded file

Example:

To perform an upload using an HTML form:

```
<form method="post" action="http://localhost:8080/fedora/management/upload"
  enctype="multipart/form-data">
  File to upload: <input type="file" name="file" size="50">
  <input type="submit">
</form>
```

WSDL

When running your own Fedora server, the API-M-LITE WSDL is available at `/wsdl?api=API-M-LITE`.

Example:

```
http://localhost:8080/fedora/wsdl?api=API-M-LITE
```

REST API

- Introduction
- [API-A Methods](#)
 - describeRepository
 - findObjects
 - getDatastreamDissemination
 - getDissemination
 - getObjectHistory
 - getObjectProfile
 - listDatastreams
 - listMethods
 - resumeFindObjects
- [API-M Methods](#)
 - addDatastream
 - addRelationship
 - compareDatastreamChecksum
 - export
 - getDatastream
 - getDatastreamHistory
 - getDatastreams
 - getNextPID
 - getObjectXML
 - getRelationships
 - ingest
 - modifyDatastream
 - modifyObject
 - purgeDatastream
 - purgeObject
 - purgeRelationship
 - setDatastreamState
 - setDatastreamVersionable
 - Validate
- [Utility Methods](#)
 - Upload
- WADL

Introduction

The Fedora REST API exposes a subset of the Fedora Access and Management APIs as a RESTful (Representational State Transfer) Web Service. For release 3.2, the Fedora REST API has been upgraded to Beta status. With this change the REST API is no longer optional, it is enabled by default as are all of the other Fedora APIs. The primary reasons behind Beta status are the need for more robust testing, as well as the understanding that in a future release the REST API will likely subsume API-A-Lite and API-M-Lite, thus changing the API somewhat.

For examples of how to use the REST API programmatically, please refer to the [TestRESTAPI test class](#).



Ensure DC, RELS-EXT and RELS-INT are versionable if using Managed Content

Due to an outstanding bug [FCREPO-849](#), if you use Managed Content for DC, RELS-EXT or RELS-INT then please make sure these datastreams are versionable (the default setting for versionable is "true", so if you haven't specified this datastream property then you are safe). Ensure that you don't inadvertently set this property to "false" for these datastreams when using the API methods.



2xx Responses only please

The HTTP Response portion of each method description listed below indicates the response on success. Unsuccessful calls will produce non-200 response codes appropriate to the error case. If, however, your client software has difficulty processing non-200 responses (such as is the case with Adobe's Flash Player) adding the query parameter 'flash=true' to any method will ensure that all responses are in the 200 range. In the event of an error, the response code will be set to 200 and the response body will include the error message followed by "::ERROR".



POST Replacement

If the client with which you are working does not support use of the PUT and/or DELETE HTTP methods but does allow you to set headers on the HTTP request, you can use POST replacement to make PUT and DELETE calls. To do this, simply set the X-HTTP-Method-Override request header to the correct method value (PUT or DELETE) and perform a POST request. Your request will be handled by the REST API as if it were a PUT or DELETE.



Removal of .xml shortcut

For release 3.3 the `.xml` shortcut has entirely been removed from the REST API due to functional inconsistencies (see [here](#) for more details. If your client uses this shortcut please change it to use the format parameter (`?format=xml`).



URL-Encoding

The REST API requires that parameters - including path parameters - are URL-encoded. Particularly this is important if you have any PIDs that use [escaped-octets](#) in the PID name. In this case the "%" character should be URL-encoded as "%25", eg a PID "changeme:1234%2F56" should be URL-encoded as "changeme:1234%252F56". The ":" PID namespace separator character does not require URL-encoding as it has no special meaning when used in the path component of HTTP URIs; however some software library URL-encoding methods will URL-encode this to %3A - that's not a problem, both ":" and "%3A" will be accepted by the REST API.

API-A Methods

describeRepository

Not implemented

findObjects

URL Syntax

`/objects ? [terms | query] [maxResults] [resultFormat] [pid] [label] [state] [ownerId] [cDate] [mDate] [dcmDate] [title] [creator] [subject] [description] [publisher] [contributor] [date] [type] [format] [identifier] [source] [language] [relation] [coverage] [rights]`

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
terms	a phrase represented as a sequence of characters (including the ? and * wildcards) for the search. If this sequence is found in any of the fields for an object, the object is considered a match. Do NOT use this parameter in combination with the "query" parameter		

query	a sequence of space-separated conditions. A condition consists of a metadata element name followed directly by an operator, followed directly by a value. Valid element names are (pid, label, state, ownerId, cDate, mDate, dcmDate, title, creator, subject, description, publisher, contributor, date, type, format, identifier, source, language, relation, coverage, rights). Valid operators are: contains (), equals (=), greater than (>), less than (<), greater than or equals (>=), less than or equals (<=). The contains () operator may be used in combination with the ? and * wildcards to query for simple string patterns. Space-separators should be encoded in the URL as %20. Operators must be encoded when used in the URL syntax as follows: the (=) operator must be encoded as %3D, the (>) operator as %3E, the (<) operator as %3C, the (>=) operator as %3E%3D, the (<=) operator as %3C%3D, and the (~) operator as %7E. Values may be any string. If the string contains a space, the value should begin and end with a single quote character ('). If all conditions are met for an object, the object is considered a match. Do NOT use this parameter in combination with the "terms" parameter		
maxResults	the maximum number of results that the server should provide at once. If this is unspecified, the server will default to a small value	25	
resultFormat	the preferred output format	html	xml, html
pid	if true, the Fedora persistent identifier (PID) element of matching objects will be included in the response	false	true, false
label	if true, the Fedora object label element of matching objects will be included in the response	false	true, false
state	if true, the Fedora object state element of matching objects will be included in the response	false	true, false
ownerId	if true, each matching objects' owner id will be included in the response>false,true, false	false	true, false
cDate	if true, the Fedora create date element of matching objects will be included in the response	false	true, false
mDate	if true, the Fedora modified date of matching objects will be included in the response	false	true, false
dcmDate	if true, the Dublin Core modified date element(s) of matching objects will be included in the response	false	true, false
title	if true, the Dublin Core title element(s) of matching objects will be included in the response	false	true, false
creator	if true, the Dublin Core creator element(s) of matching objects will be included in the response	false	true, false
subject	if true, the Dublin Core subject element(s) of matching objects will be included in the response	false	true, false
description	if true, the Dublin Core description element(s) of matching objects will be included in the response	false	true, false
publisher	if true, the Dublin Core publisher element(s) of matching objects will be included in the response	false	true, false
contributor	if true, the Dublin Core contributor element(s) of matching objects will be included in the response	false	true, false
date	if true, the Dublin Core date element(s) of matching objects will be included in the response	false	true, false
type	if true, the Dublin Core type element(s) of matching objects will be included in the response	false	true, false
format	if true, the Dublin Core format element(s) of matching objects will be included in the response	false	true, false
identifier	if true, the Dublin Core identifier element(s) of matching objects will be included in the response	false	true, false
source	if true, the Dublin Core source element(s) of matching objects will be included in the response	false	true, false
language	if true, the Dublin Core language element(s) of matching objects will be included in the response	false	true, false

relation	if true, the Dublin Core relation element(s) of matching objects will be included in the response	false	true, false
coverage	if true, the Dublin Core coverage element(s) of matching objects will be included in the response	false	true, false
rights	if true, the Dublin Core rights element(s) of matching objects will be included in the response	false	true, false

Examples

/objects?terms=demo&pid=true&subject=true&label=true&resultFormat=xml

/objects?query=title%7Erome%20creator%7E Staples&pid=true&title=true&creator=true

/objects?query=pid%7E*1&maxResults=50&format=true&pid=true&title=true

getDatastreamDissemination

URL Syntax

/objects/{pid}/datastreams/{dsID}/content ? [asOfDateTime] [download]

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
{dsID}	datastream identifier		
asOfDateTime	indicates that the result should be relative to the digital object as it existed at the given date and time		yyyy-MM-dd or yyyy-MM-ddTHH:mm:ssZ
download	If true, a content-disposition header value "attachment" will be included in the response, prompting the user to save the datastream as a file. A content-disposition header value "inline" will be used otherwise. The filename used in the header is generated by examining in order: RELS-INT for the relationship fedora-model:downloadFilename, the datastream label, and the datastream ID. The file extension (apart from where the filename is specified in RELS-INT) is determined from the MIMETYPE. The order in which these filename sources are searched, and whether or not to generate an extension from the MIMETYPE, is configured in fedora.fcfg. The file used to map between MIMETYPES and extensions is mime-to-extensions.xml located in the server config directory.		

Examples

/objects/demo:29/datastreams/DC/content

/objects/demo:29/datastreams/DC/content?asOfDateTime=2008-01-01

getDissemination

URL Syntax

/objects/{pid}/methods/{sdefPid}/{method} ? [method parameters]

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
{sdefPid}	persistent identifier of the sDef defining the methods		
{method}	method to invoke		
method parameters	any parameters required by the method		

Examples

/objects/demo:29/methods/demo:27/resizeImage?width=100

/objects/demo:SmileyEarring/methods/demo:DualResolution/fullSize

getObjectHistory

URL Syntax

/objects/{pid}/versions ? [format]

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
format	the preferred output format	html	xml, html

Examples

/objects/demo:29/versions

/objects/demo:29/versions?format=xml

getObjectProfile

URL Syntax

/objects/{pid} ? [format] [asOfDateTime]

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
format	the preferred output format	html	xml, html
asOfDateTime	indicates that the result should be relative to the digital object as it existed on the given date		yyyy-MM-dd or yyyy-MM-ddTHH:mm:ssZ

Examples

/objects/demo:29

/objects/demo:29?format=xml

/objects/demo:29?asOfDateTime=2008-01-01

listDatastreams

URL Syntax

/objects/{pid}/datastreams ? [format] [asOfDateTime]

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
format	the preferred output format	html	xml, html
asOfDateTime	indicates that the result should be relative to the digital object as it existed on the given date		yyyy-MM-dd or yyyy-MM-ddTHH:mm:ssZ

Examples

/objects/demo:35/datastreams

/objects/demo:35/datastreams?format=xml&asOfDateTime=2008-01-01T05:15:00Z

listMethods

URL Syntax

1. /objects/{pid}/methods ? [format] [asOfDateTime]
2. /objects/{pid}/methods/{sdefPid} ? [format] [asOfDateTime]

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
{sdefPid}	persistent identifier of the SDef defining the methods		
format	the preferred output format	html	xml, html
asOfDateTime	indicates that the result should be relative to the digital object as it existed on the given date		yyyy-MM-dd or yyyy-MM-ddTHH:mm:ssZ

Examples

/objects/demo:29/methods

/objects/demo:29/methods?format=xml&asOfDateTime=2008-01-01T05:15:00Z

/objects/demo:29/methods/demo:27

/objects/demo:29/methods/demo:27?format=xml&asOfDateTime=2008-01-01T05:15:00Z

resumeFindObject

URL Syntax

/objects ? [sessionToken] [all findObjects options]

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
sessionToken	the identifier of the session to which the search results are being returned		
all findObjects options	all of the same options are available for resumeFindObject as for findObjects		

Examples

/objects?terms=*&format=xml&pid=true&subject=true&label=true&sessionToken=xyz\\

API-M Methods

addDatastream

URL Syntax

/objects/{pid}/datastreams/{dsID} ? [controlGroup] [dsLocation] [altIDs] [dsLabel] [versionable] [dsState] [formatURI] [checksumType] [checksum] [mimeType] [logMessage]

HTTP Method

POST

HTTP Response

201

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
{dsID}	datastream identifier		
controlGroup	one of "X", "M", "R", or "E" (Inline *X*ML, *M*anaged Content, *R*edirect, or *E*ternal Referenced)	X	X, M, R, E
dsLocation	location of managed or external datastream content		
altIDs	alternate identifiers for the datastream		
dsLabel	the label for the datastream		
versionable	enable versioning of the datastream	true	true, false
dsState	one of "A", "I", "D" (*A*ctive, *I*nactive, *D*eleted)	A	A, I, D
formatURI	the format URI of the datastream		
checksumType	the algorithm used to compute the checksum	DEFAULT	DEFAULT, DISABLED, MD5, SHA-1, SHA-256, SHA-385, SHA-512
checksum	the value of the checksum represented as a hexadecimal string		
mimeType	the MIME type of the content being added, this overrides the Content-Type request header		
logMessage	a message describing the activity being performed		
multipart file as request content	datastream file (for Managed datastreams)		

Examples

POST: /objects/demo:29/datastreams/NEWDS?controlGroup=X&dsLabel=New (with Multipart file)

POST: /objects/demo:29/datastreams/NEWDS?controlGroup=M&dsLocation=<http://example:80/newds>&dsLabel=New

addRelationship

URL Syntax

/objects/{pid}/relationships/new ? [subject] [predicate] [object] [isLiteral] [datatype]

HTTP Method

POST

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
subject	subject of the relationship. Either a URI for the object or one of its datastreams	URI of this object	
predicate	predicate of the relationship		
object	object of the relationship		
isLiteral	true if the object of the relationship is a literal, false if it is a URI		true, false
datatype	if the object is a literal, the datatype of the literal (optional)		

Examples

POST

/objects/demo:29/relationships/new?subject=info%3afedora%2fdemo%3a29%2fDC&predicate=http%3a%2f%2fwww.example.org%2fre

compareDatastreamChecksum

See [getDatastream](#)

export

URL Syntax

/objects/{pid}/export ? [format] [context] [encoding]

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
format	the XML format to export	info:fedora/fedora-system:FOXML-1.1	info:fedora/fedora-system:FOXML-1.1, info:fedora/fedora-system:FOXML-1.0, info:fedora/fedora-system:METSFedoraExt-1.1, info:fedora/fedora-system:METSFedoraExt-1.0, info:fedora/fedora-system:ATOM-1.1, info:fedora/fedora-system:ATOMZip-1.1
context	the export context, which determines how datastream URLs and content are represented	public	public, migrate, archive
encoding	the preferred encoding of the exported XML	UTF-8	

Examples

/objects/demo:29/export

/objects/demo:29/export?context=migrate

getDatastream

URL Syntax

/objects/{pid}/datastreams/{dsID} ? [asOfDateTime] [format] [validateChecksum]

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
{dsID}	datastream identifier		
format	the preferred output format	html	xml, html
asOfDateTime	indicates that the result should be relative to the digital object as it existed on the given date		yyyy-MM-dd or yyyy-MM-ddTHH:mm:ssZ
validateChecksum	verifies that the Datastream content has not changed since the checksum was initially computed. If asOfDateTime is null, Fedora will use the most recent version.	false	true, false

Examples

/objects/demo:29/datastreams/DC

/objects/demo:29/datastreams/DC?format=xml

/objects/demo:29/datastreams/DC?format=xml&validateChecksum=true

getDatastreamHistory

URL Syntax

/objects/{pid}/datastreams/{dsid}/versions ? [format]

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
format	the preferred output format	html	xml, html

Examples

GET: /objects/changeme:1/datastreams/DC/versions

GET: /objects/changeme:1/datastreams/DC/versions?format=xml

getDatastreams

Not implemented

getNextPID

URL Syntax

/objects/nextPID ? [numPIDs] [namespace] [format]

HTTP Method

POST

HTTP Response

200

Parameters

Name	Description	Default	Options
numPIDs	the number of pids to retrieve	1	
namespace	the namespace of the requested pid(s)	the default namespace of the repository	
format	the preferred output format	html	xml, html

Examples

POST: /objects/nextPID

POST: /objects/nextPID?numPIDs=5&namespace=test&format=xml

getObjectXML

URL Syntax

/objects/{pid}/objectXML

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		

Examples

/objects/demo:29/objectXML

getRelationships

URL Syntax

/objects/{pid}/relationships ? [subject] [predicate] [format]

HTTP Method

GET

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
subject	subject of the relationship(s). Either a URI for the object or one of its datastreams	URI of this object	
predicate	predicate of the relationship(s), if missing returns all predicates		
format	format of the response	rdf/xml	xml (returns rdf/xml), rdf/xml, n-triples, turtle, sparql

Examples

/objects/demo:29/relationships

/objects/demo:29/relationships?subject=info%3afedora%2fdemo%3a29%2fDC

ingest

URL Syntax

/objects/ [{pid}] new ? [label] [format] [encoding] [namespace] [ownerId] [logMessage] [ignoreMime]

HTTP Method

POST

HTTP Response

201

Request Content

text/xml

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the object to be created	new (see below)	
new	indicator that either a new PID should be created for this object or that the PID to be used is encoded in the XML included as the body of the request		
label	the label of the new object		
format	the XML format of the object to be ingested	info:fedora/fedora-system:FOXML-1.1, info:fedora/fedora-system:FOXML-1.0, info:fedora/fedora-system:METSFedoraExt-1.1, info:fedora/fedora-system:METSFedoraExt-1.0, info:fedora/fedora-system:ATOM-1.1, info:fedora/fedora-system:ATOMZip-1.1	
encoding	the encoding of the XML to be ingested. If this is specified, and given as anything other than UTF-8, you must ensure that the same encoding is declared in the XML. For example, if you specify "ISO-88591" as the encoding, the XML should start with: <?xml version="1.0" encoding="ISO-8859-1"?>	UTF-8	
namespace	the namespace to be used to create a PID for a new empty object; if object XML is included with the request, the namespace parameter is ignored	the default namespace of the repository	
ownerId	the id of the user to be listed at the object owner		
logMessage	a message describing the activity being performed		
ignoreMime	indicates that the request should not be checked to ensure that the content is XML prior to attempting an ingest. This is provided to allow for client applications which do not indicate the correct Content-Type when submitting a request.	false	true, false
XML file as request content	file to be ingested as a new object		

Notes

Executing this request with no request content will result in the creation of a new, empty object (with either the specified PID or a system-assigned PID). The new object will contain only a minimal DC datastream specifying the dc:identifier of the object.

Examples

POST: /objects/new

POST: /objects

POST: /objects/new?namespace=demo

POST: /objects/test:100?label=Test

modifyDatastream

URL Syntax

/objects/{pid}/datastreams/{dsID} ? [dsLocation] [altIDs] [dsLabel] [versionable] [dsState] [formatURI] [checksumType] [checksum] [mimeType] [logMessage] [ignoreContent] [lastModifiedDate]

HTTP Method

PUT

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
{dsID}	datastream identifier		
dsLocation	location of datastream content		
altIDs	alternate identifiers for the datastream		
dsLabel	the label for the datastream		
versionable	enable versioning of the datastream	the "versionable" property of the existing datastream	true, false
dsState	one of "A", "I", "D" (*A*ctive, *I*nactive, *D*eleted)	A	A, I, D
formatURI	the format URI of the datastream		
checksumType	the algorithm used to compute the checksum	DEFAULT	DEFAULT, DISABLED, MD5, SHA-1, SHA-256, SHA-385, SHA-512
checksum	the value of the checksum represented as a hexadecimal string		
mimeType	the MIME type of the content being added, this overrides the Content-Type request header		
logMessage	a message describing the activity being performed		
ignoreContent	tells the request handler to ignore any content included as part of the request, indicating that you do not intend to update the datastream content. This is primarily provided to allow the use of client tools which always require content to be included as part of PUT requests.	false	true, false
lastModifiedDate	date/time of the last (known) modification to the datastream, if the actual last modified date is later, a 409 response is returned		
multipart file as request content	file to replace existing datastream (for Managed datastreams)		

Examples

PUT: /objects/demo:35/datastreams/HIGH (with Multipart file)

PUT: /objects/demo:35/datastreams/HIGH?dsLocation=<http://example:80/highDS?logMessage=Update>

modifyObject

URL Syntax

/objects/{pid} ? [label] [ownerId] [state] [logMessage] [lastModifiedDate]

HTTP Method

PUT

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
label	the new object label		
ownerId	the id of the user to be listed at the object owner		
state	the new object state - *A*ctive, *I*nactive, or *D*eleted	A	A, I, D
logMessage	a message describing the activity being performed		
lastModifiedDate	date/time of the last (known) modification to the datastream, if the actual last modified date is later, a 409 response is returned		

Examples

PUT: /objects/demo:29?label=Updated

PUT: /objects/demo:29?state=D?logMessage=Deleted

purgeDatastream

URL Syntax

/objects/{pid}/datastreams/{dsID} ? [startDT] [endDT] [logMessage]

HTTP Method

DELETE

HTTP Response

200 with a string array of the date-time stamps of the versions purged

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
{dsID}	datastream identifier		
startDT	the (inclusive) start date-time stamp of the range. If not specified, this is taken to be the lowest possible value, and thus, the entire version history up to the endDT will be purged		yyyy-MM-dd or yyyy-MM-ddTHH:mm:ssZ
endDT	the (inclusive) ending date-time stamp of the range. If not specified, this is taken to be the greatest possible value, and thus, the entire version history back to the startDT will be purged		yyyy-MM-dd or yyyy-MM-ddTHH:mm:ssZ
logMessage	a message describing the activity being performed		

Examples

DELETE: /objects/demo:35/datastreams/HIGH

purgeObject

URL Syntax

/objects/{pid} ? [logMessage]

HTTP Method

DELETE

HTTP Response

204

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
logMessage	a message describing the activity being performed		

Examples

DELETE: /objects/demo:29

purgeRelationship

URL Syntax

/objects/{pid}/relationships ? [subject] [predicate] [object] [isLiteral] [datatype]

HTTP Method

DELETE

HTTP Response

200

Return body

Text indicating if the relationship was successfully purged: true or false

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
subject	subject of the relationship. Either a URI for the object or one of its datastreams	URI of this object	
predicate	predicate of the relationship		
object	object of the relationship		
isLiteral	true if the object of the relationship is a literal, false if it is a URI		true, false
datatype	if the object is a literal, the datatype of the literal (optional)		

Examples

DELETE

/objects/demo:29/relationships?subject=info%3afedora%2fdemo%3a29%2fDC&predicate=http%3a%2f%2fwww.example.org%2frels%2

setDatastreamState

URL Syntax

/objects/{pid}/datastreams/{dsID} ? [dsState]

HTTP Method

PUT

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
{dsID}	datastream identifier		
dsState	one of "A", "I", "D" (*A*ctive, *I*nactive, *D*eleted)	A	A, I, D

Examples

PUT: /objects/demo:35/datastreams/HIGH?dsState=D

setDatastreamVersionable

URL Syntax

/objects/{pid}/datastreams/{dsID} ? [versionable]

HTTP Method

PUT

HTTP Response

200

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
{dsID}	datastream identifier		
versionable	enable versioning of the datastream	true	true, false

Examples

PUT: /objects/demo:35/datastreams/HIGH?versionable=false

Validate

URL Syntax

/objects/{pid}/validate ? [asOfDateTime]

HTTP Method

GET

HTTP Response

200 (OK) if the validation could be carried out (even if the object is not valid)

404 If some object or datastream could not be carried out

401 If the user credentials was insufficient

400 If the parameters are misformed

409 If one of the relevant objects were locked

500 If something else failed on the server

Return Body

XML, adhering to this schema

```

<xs:schema targetNamespace="http://www.fedora.info/definitions/1/0/access/"
  xmlns="http://www.fedora.info/definitions/1/0/access/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="validation">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="asOfDateTime"/>
        <xs:element ref="contentModels"/>
        <xs:element ref="problems"/>
        <xs:element ref="datastreamProblems"/>
      </xs:sequence>
      <xs:attribute name="pid" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="valid" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:boolean"/>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="asOfDateTime">
    <xs:simpleType>
      <xs:restriction base="xs:dateTime"/>
    </xs:simpleType>
  </xs:element>
  <xs:element name="contentModels">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="model" minOccurs="0" maxOccurs="unbounded" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="problems">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="problem" minOccurs="0" maxOccurs="unbounded"
type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="datastreamProblems">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="datastream" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="datastream">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="problem" minOccurs="0" maxOccurs="unbounded"
type="xs:string"/>
      </xs:sequence>
      <xs:attribute name="datastreamID" use="required">
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Parameters

Name	Description	Default	Options
{pid}	persistent identifier of the digital object		
asOfDateTime	indicates that the result should be relative to the digital object and the repository as it existed at the given date and time		yyyy-MM-dd or yyyy-MM-ddTHH:mm:ssZ

Examples

GET /objects/validate/demo:29?asOfDateTime=2008-01-01

Returns HTTP 200 with the body

```
<?xml version="1.0" encoding="UTF-8"?>
<validation pid="demo:29" valid="true">
  <asOfDateTime>2008-01-01T00:00:00.000Z</asOfDateTime>
  <contentModels>
    <model>info:fedora/fedora-system:FedoraObject-3.0</model>
  </contentModels>
  <problems>
  </problems>
  <datastreamProblems>
  </datastreamProblems>
</validation>
```

GET /objects/validate/demo:fail

Returns HTTP 200 with the body. Here the error was a misspelled element in the DC datastream, "dc:titel"

```
<?xml version="1.0" encoding="UTF-8"?>
<validation pid="demo:fail" valid="false">
  <asOfDateTime></asOfDateTime>
  <contentModels>
    <model>info:fedora/fedora-system:FedoraObject-3.0</model>
  </contentModels>
  <problems>
  </problems>
  <datastreamProblems>
    <datastream datastreamID="DC">

      <problem>Encountered schema validation error while parsing datastream 'DC' with the schema
from content model 'fedora-system:FedoraObject-3.0'. The error was 'cvc-complex-type.2.4.a:
Invalid
content was found starting with element 'dc:titel'. One of
{'http://purl.org/dc/elements/1.1/':title, "http://purl.org/dc/elements/1.1/":creator,
"http://purl.org/dc/elements/1.1/":subject,
"http://purl.org/dc/elements/1.1/":description, "http://purl.org/dc/elements/1.1/":publisher,
"http://purl.org/dc/elements/1.1/":contributor, "http://purl.org/dc/elements/1.1/":date,
"http://purl.org
/dc/elements/1.1/":type, "http://purl.org/dc/elements/1.1/":format,
"http://purl.org/dc/elements/1.1/":identifier, "http://purl.org/dc/elements/1.1/":source,
"http://purl.org/dc/elements/1.1/":language,
"http://purl.org/dc/elements/1.1/":relation, "http://purl.org/dc/elements/1.1/":coverage,
"http://purl.org/dc/elements/1.1/":rights}' is expected.</problem>
    </datastream>
  </datastreamProblems>
</validation>
```

Utility Methods

Upload

URL Syntax

/upload

HTTP Method

POST

HTTP Response

202 and a URI for the uploaded file

Parameters

Multipart file as request content

Examples

POST: /upload (with Multipart file)

WADL

When running your own Fedora server, the REST API WADL is available at **/objects/application.wadl**.

Example:

```
http://localhost:8080/fedora/objects/application.wadl
```

Basic Search



Deprecation of "LITE" APIs

As of release 3.4-RC1 of Fedora, the "LITE" APIs are deprecated and will be removed in a future release. You are encouraged to migrate any existing code that uses the <http://localhost:8080/fedora/search> endpoint to use the <http://localhost:8080/fedora/objects> endpoint instead. This is the search endpoint from the new REST API.

- [Introduction](#)
- [Search Interface Syntax](#)
- [Search Interface Web Form](#)
- [Search Results](#)
- [Simple Search Query Help](#)
- [Fielded Search Query Help](#)
- [Search Field Name Help](#)

Introduction

The Search Interface provides a mechanism for searching and browsing the Fedora repository. Upon ingestion, metadata from the Fedora System Metadata section and the Dublin Core (DC) Metadata section of the object are indexed in a relational database, and may be searched using this interface. The DC Metadata section is an optional Implementor-Defined XML Metadata datastream in the object, where the *Datastream ID* is DC, and the XML conforms to the schema at http://www.openarchives.org/OAI/2.0/oai_dc.xsd. If a Dublin Core metadata datastream is not provided, Fedora will construct a minimal DC datastream consisting of the elements dc:title and dc:identifier. The value for dc:title will be obtained from the object's label (if present in the object) and the value for dc:identifier will be assigned to the object's persistent identifier or PID.

The search interface provides both simple and advanced searching via a web form included with the repository software. All queries are case insensitive. Simple Search enables queries of words and phrases occurring anywhere in than object's indexed metadata fields. Advanced Search enables fielded searching across any combination of metadata elements using string comparison operators (= and ~) for string fields, and value comparison operators (=, >, , <,) for date fields (dc:date fields may be treated as both). The wildcards, * and ? may be used in any string-based query.

Search Interface Syntax

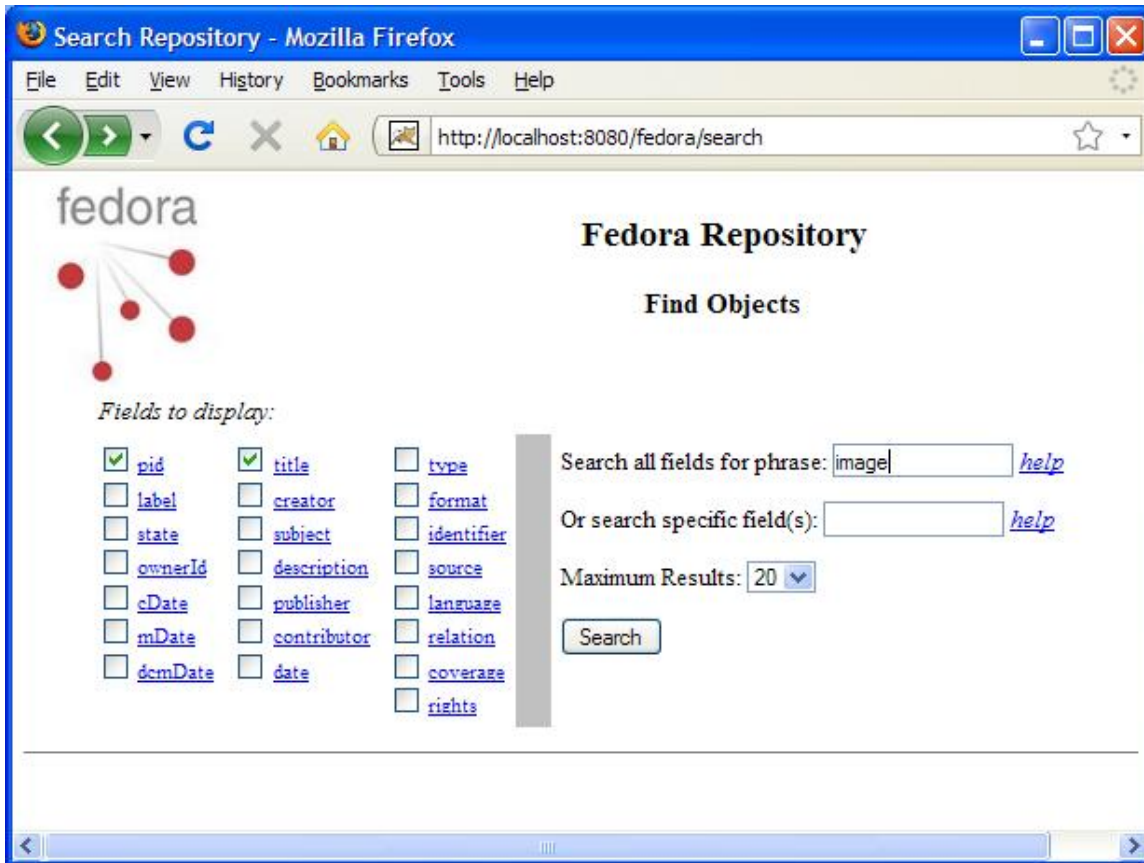
The Fedora Search Interface is implemented as a java servlet and can be accessed using the following syntax:

```
http://hostname:port/fedora/search
```


- **hostname** – required hostname of the Fedora server
- **port** – required port number on which the Fedora server is running
- **fedora** – a required part of the Fedora servlet path
- **search** – a required part of the Fedora servlet path

Search Interface Web Form

An example of the Search Interface Web Form is displayed below. To perform a search, check the box next to the fields that you would like returned in the search results, enter your search query in either the simple search or advanced search text box, and click the submit button. In the example, the label fields of *pid* and *label* are checked and the query is to search for the word "image" in any of the indexed fields.



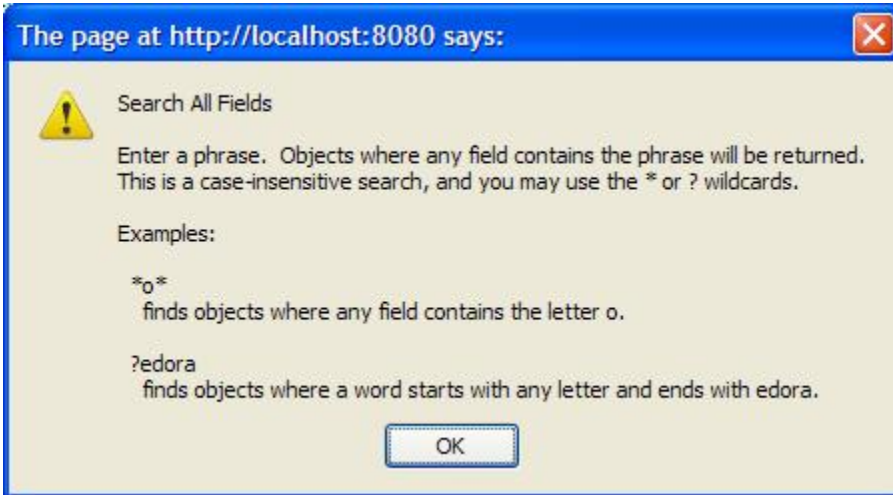
Search Results

The results of the sample search are displayed below. Note that the entries returned for the PID field are hyperlinks. Clicking on the PID hyperlink will invoke the Default Disseminator on the specified object and display the Object Profile for the object. The Search Interface provides an easy means of browsing and examining objects in the repository.

pid	label
demo:5	Image of Coliseum in Rome
demo:29	Image of Coliseum in Rome
demo:7	Image of Architectural Drawings for Pavillion III, University of Virginia
demo:6	Pavillion III, IVA Image Collection - University of Virginia
demo:11	Exhibit Intro: Architectural drawings, Pavillion III, IVA Image Collection - University of Virginia
demo:10	Column Detail, Pavillion III, IVA Image Collection - University of Virginia

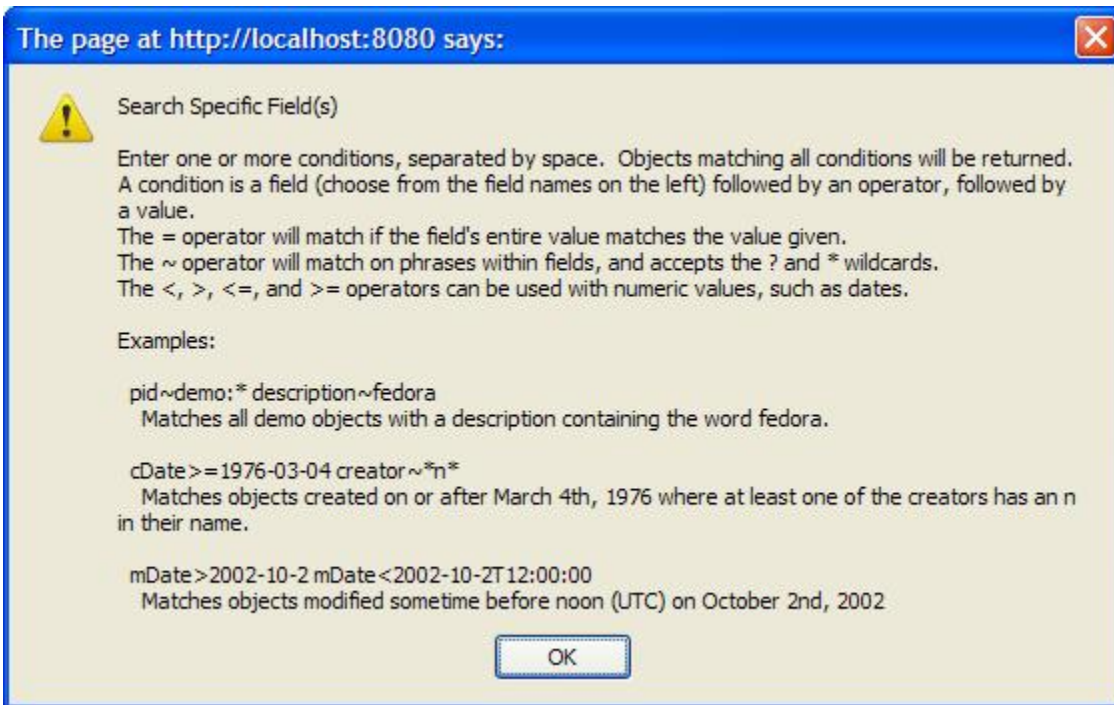
Simple Search Query Help

The Simple Search query searches both the Dublin Core metadata and the Fedora System Metadata fields. Clicking on the the *help* hyperlink next to the *Search all fields for phrase* text box will display additional help about Simple Search as displayed below.



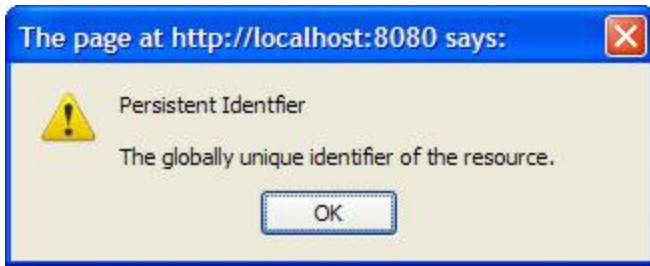
Fielded Search Query Help

The Fielded Search query enables searching of specific fields provided in the query. Clicking on the the *help* hyperlink next to the *search specific field(s)* text box will display additional help about Fielded Search as displayed below.



Search Field Name Help

Help is also available on each search field name if you are unfamiliar with the search field labels. Clicking on the hyperlink associated with each search field label provides additional help on that specific search field name. An example of the help provided for the *pid* search field is displayed below.



Basic OAI-PMH Provider

- [Introduction](#)
- [Interface Syntax](#)

Introduction

The [OAI Protocol for Metadata Harvesting \(OAI-PMH\)](#) is a standard for sharing metadata across repositories. Every Fedora digital object has a primary Dublin Core record that conforms to the schema at: http://www.openarchives.org/OAI/2.0/oai_dc.xsd. This metadata is accessible using Fedora's OAI-PMH Provider Interface. Currently, only the Dublin Core metadata for each object may be disseminated via this interface. The Fedora OAI-PMH Provider Interface uses the standard OAI-PMH syntax and will accept any valid OAI-PMH Version 2 request.

Interface Syntax

The Fedora OAI-PMH Provider Interface is implemented as a java servlet and can be accessed using the following syntax:

```
http://hostname:port/fedora/oai?verb=oai-pmh2.0-verb
```

- **hostname** – required hostname of the Fedora server.
- **port** – required port number on which the Fedora server is running.
- **fedora** – a required part of the Fedora servlet path.
- **oai** – a required part of the Fedora servlet path.
- **verb** – required keyword of the OAI-PMH 2.0 syntax.
- **oai-pmh2.0-verb** – required valid OAI-PMH 2.0 verb.

For example, to request an OAI `_Identify_verb` request, use a URL similar to the following:

```
http://localhost:8080/fedora/oai?verb=Identify
```

and the Fedora repository will respond with an XML-encoded mime-typed stream providing identification information about the Fedora OAI provider.

Resource Index Search

- [Introduction](#)
- [User Interface](#)
 - [Find Tuples](#)
 - [Language](#)
 - [Response](#)
 - [Limit](#)
 - [Advanced](#)
 - [Find Triples](#)
 - [SPO](#)
 - [Response Formats](#)
 - [Using Templates](#)
 - [Show Aliases](#)
- [Application Interface](#)
 - [Syntax for Requesting Tuples](#)
 - [Syntax for Requesting Triples](#)

Introduction

The Resource Index Search Service (RISearch) is a web service that exposes the contents of a repository's [Resource Index](#) guide for outside use. This document introduces the use of this service through a web browser interface, then describes how to access it programmatically.

User Interface

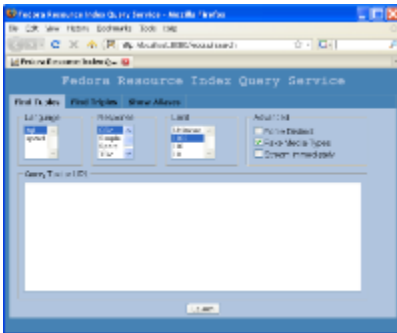
When your Fedora server is running, the RISEarch service will be available under `/fedora/risearch`. For example:

```
http://localhost:8080/fedora/risearch
```

The user interface consists of three tabs: Find Tuples, Find Triples, and Show Aliases. A detailed description of each of these tabs follows.

Find Tuples

The "Find Tuples" tab shown below is used to run tuple queries against the resource index. A *tuple query* is one that returns a list of named values.



When you enter a query and click "Launch", a new browser window will display the results.

To get an idea of how it works, try the following iTQL query, which asks for information about all Service Definition objects in the repository:

```
select $object $modified from <#ri>
where $object <fedora-model:hasModel> <info:fedora/fedora-system:ServiceDefinition-3.0>
and $object <fedora-view:lastModifiedDate> $modified
```

In response, you should see something like this:

```
"object","modified"
info:fedora/demo:1,2009-02-16T19:39:28.859Z
info:fedora/demo:12,2009-02-16T19:39:17.843Z
info:fedora/demo:19,2009-02-16T19:39:20.375Z
info:fedora/demo:22,2009-02-16T19:39:20.671Z
info:fedora/demo:Collection,2009-02-16T19:39:24.89Z
info:fedora/demo:8,2009-02-16T19:39:34.281Z
info:fedora/demo:DualResolution,2009-02-16T19:39:25.093Z
info:fedora/demo:27,2009-02-16T19:39:26.578Z
```

This is a list of comma-separated values, each row representing the URI and modified date of the objects that matched the query.

Above the query text box, you can alter several settings for a query. These settings are described below.

Language

Indicates the query language to use. Currently, the options are **SPARQL** and **iTQL** (a full-featured RDF query language supported by [Mulgara](#)).

Response

Indicates the desired response format. Valid options include:

- **CSV** – Comma-separated values
- **Simple** – A simple easy-to-read text format that shows datatype information, when present
- **Sparql** – The W3C standard query response
- **TSV** – Tab-separated values
- **Count** – A count of the item returned by the query

Limit

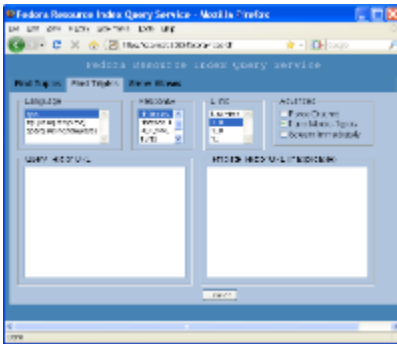
The maximum number of results to return. It is useful to set this low when testing queries.

Advanced

- **Force Distinct** – Whether to force duplicate results to be dropped. Note: iTQL never returns duplicates.
- **Fake Media-Types** – Whether to send incorrect Content-Type HTTP response headers with the responses (to trick browsers into displaying the results instead of popping up a "Save As/Open With" window).
- **Stream Immediately** – Whether to stream the results right away (faster), or to save them to a temporary file before sending them to the client. The default behavior (to save the results before streaming) will give a more informative error message if a query fails.

Find Triples

The "Find Triples" tab shown below is used to run triple queries against the resource index. A *triple query* is one that returns a list of RDF statements (aka triples).



This tab works in much the same way as the "Find Tuples" tab, but supports different response formats and provides a means to convert tuple query results to triples. It also exposes another query language: SPO.

SPO

This is a very simple RDF query language, where queries consist of a specific subject (or an asterisk, indicating "any"), a specific predicate (or an asterisk), and a specific object (or an asterisk). The easiest way to learn SPO is by example:

Get all triples in the repository

```
***
```

Get all triples where the object is demo:1

```
** <info:fedora/demo:1>
```

Get all triples where the subject is demo:1 and the object is fedora-system:ServiceDefinition-3.0

```
<info:fedora/demo:1> * <info:fedora/fedora-system:ServiceDefinition-3.0>
```

Response Formats

A variety of RDF formats are supported:

- **N-Triples** – A subset of Notation 3 defined in the [RDF Test Cases](#) document
- **Notation 3** – The original RDF text format, defined by Tim Berners-Lee in [An RDF language for the Semantic Web](#)
- **RDF/XML** – The "RDF/XML" format, defined in the [RDF/XML Syntax Specification](#)
- **Turtle** – A newer subset of Notation 3, defined in Dave Beckett's [Turtle - Terse RDF Triple Language](#)
- **count** – A count of the item returned by the query

Using Templates

Templates are used to convert tuple query results to triples. A template consists of one or more *triple binding patterns* that reference the binding variables in an iTQL query.

The easiest way to understand how this works is by example.

In this example, we'll show how to extract a subgraph from the resource index using iTQL. Enter the following query text:

```
select $a $r $b from <#ri>
where $a <fedora-model:hasModel> <info:fedora/fedora-system:FedoraObject-3.0>
and $a $r $b
and $b <fedora-model:hasModel> <info:fedora/fedora-system:FedoraObject-3.0>
```

This query by itself returns all relationships between data objects in a repository. The binding variables are \$a, \$r, and \$b. Now enter the following in the template text box:

```
$a $r $b
```

When you launch the query, you'll see a list of triples: the sub-graph of all object-to-object relationships in the repository. (If you don't see anything, you should ingest the demo objects which include some sample relationships).

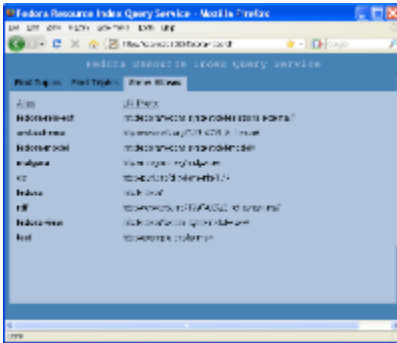
Now try the following template instead. This demonstrates how to derive new statements from those in the resource index:

```
$a <urn:example:isRelatedTo> $b
$b <urn:example:isRelatedTo> $a
```

Running the query will now show *two* statements for every object-to-object relationship in the resource index graph.

Note: When using templates to transform tuples to triples, some duplicates may be returned. These can be avoided by checking "Force Distinct".

Show Aliases



This tab shows the aliases that can be used in queries and what URI prefixes they map to.

Aliases are just shortcuts that help make queries easier to write. For example, in a query you can write `<fedora-model:state>` instead of `<info:fedora/fedora-system:def/model#state>`.

Application Interface

The RISearch service can be programmatically accessed via HTTP GET or POST. To avoid character encoding issues, POST should always be used when the query is passed in by value and contains non-ASCII characters.

As with the user interface, it can be invoked to retrieve tuples or triples. The syntax is described below.

Note:

- Square brackets ("[] ") indicate that the parameter is optional.
- As with all HTTP parameters, unsafe URI characters should be URI-escaped. For readability purposes, URI escaping is not shown below.
- The *query* and *template* parameters optionally take the value by *reference* – that is, a URL to a query or template can be given instead of the actual text.
- The *flush* parameter tells the resource index to ensure that any recently-added/modified/deleted triples are flushed to the triplestore before executing the query. This option can be desirable in certain scenarios, but for performance reasons, should be used sparingly when a process is making many API-M calls to Fedora in a short period of time: We have found that Mulgara generally achieves a much better overall update rate with large batches of triples.

Syntax for Requesting Tuples

```
http://localhost:8080/fedora/risearch?type=tuples
      &flush=[*true* (default is false)]
      &lang=*itql|sparql*
      &format=*CSV|Simple|Sparql|TSV*
      &limit=[*1* (default is no limit)]
      &distinct=[*on* (default is off)]
      &stream=[*on* (default is off)]
      &query=*QUERY_TEXT_OR_URL*
```

Syntax for Requesting Triples

```
http://localhost:8080/fedora/risearch?type=triples
      &flush=[*true* (default is false)]
      &lang=*spo|itql|sparql*
      &format=*N-Triples|Notation 3|RDF/XML|Turtle*
      &limit=[*1* or more (default is no limit)]
      &distinct=[*on* (default is off)]
      &stream=[*on* (default is off)]
      &query=*QUERY_TEXT_OR_URL*
      &template=[*TEMPLATE_TEXT_OR_URL* (if applicable)]
```

Fedora Framework Services

- [Generic Search Service \(GSearch\)](#)
A service to index datastreams and disseminations with a Lucene or Zebra backend
- [Directory Ingest Service \(DirIngest\)](#)
A service to ingest directories of files into Fedora
 - [SIP Creator](#)
A GUI for preparing SIPs for use with DirIngest
- [OAI-PMH Provider Service \(PROAI\)](#)
A service for a configurable, flexible OAI-PMH Provider
- [SWORD-Fedora](#)
The Simple Web-service Offering Repository Deposit service for Fedora

Fedora Clients

Client Command-line Utilities

Example SOAP Client

Fedora Administrator

Fedora Web Administrator

Messaging Client

REST API Java Client

Client Command-line Utilities

Table of Contents

1. [Introduction](#)
2. [fedora-dsinfo](#)
3. [fedora-export](#)
4. [fedora-find](#)
5. [fedora-ingest](#)
6. [fedora-ingest-demos](#)
7. [fedora-convert-demos](#)
8. [fedora-purge](#)
9. [fedora-modify](#)
10. [fedora-batch-build](#)
11. [fedora-batch-ingest](#)
12. [fedora-batch-buildingest](#)
13. [fedora-validate-objects](#)

Introduction

The Fedora client distribution comes with several command-line utilities that can be used to run some common operations without bringing up the GUI or writing your own SOAP client. A description and usage instructions for each follows.

This guide assumes you have correctly installed the Fedora client distribution as per the install guide, including having set up your PATH and FEDORA_HOME appropriately. The command-line scripts are located in `FEDORA_HOME/client/bin/`. In Windows, these commands resolve to batch files (`.bat`); in Unix, they resolve to shell scripts (`.sh`).



Java memory settings

Note that the memory settings used by the client command-line utilities are set in the script `env-client.sh` (or `env-client.bat` on Windows). If you have any problems with running out of memory you can change the default values in this file.

Note: There are also [server command-line utilities](#) which perform server rebuilding, validation, and other functions.

fedora-dsinfo

fedora-dsinfo [host] [port] [user] [password] [pid] [protocol] [context]

Where:

- **host** - the hostname of the Fedora server; default is localhost
- **port** - the port number on which the Fedora server is running; default is 8080
- **user** - the Fedora user (e.g. fedoraAdmin)
- **password** - the Fedora user's password
- **pid** - the pid of the object whose datastream information should be shown
- **protocol** - how to connect to repository, either http or https
- **context** - an `_optional_` parameter indicating the webapp context. This is only necessary if the Fedora server was installed under a context name other than 'fedora' (see [Alternative Webapp Context Configuration](#)).

Example:

Display key information about each of an object's datastreams.

```
fedora-dsinfo localhost 8080 fedoraAdmin fedoraAdmin demo:5 http
```

fedora-export

fedora-export [host:port] [user] [password] [pid | ftyps] [format] [econtext] [path] [protocol] [context]

Where:

- **host:port** - the repository's hostname and the port separated by colon
- **user** - the Fedora user (e.g. fedoraAdmin)
- **password** - the Fedora user's password
- **pid | ftyps** - Either the identifier (PID) of the object to export from the repository OR the types of objects to export (FTYPS). FTYPS can be any combination of the characters O, D, and M, specifying which Fedora object type(s) should be exported. O=regular data objects, D=behavior definitions, M=behavior mechanisms.
- **format** - The XML format to export. Valid options are: `info:fedora/fedora-system:FOXML-1.1` (for FOXML 1.1), `info:fedora/fedora-system:FOXML-1.0` (for FOXML 1.0), `info:fedora/fedora-system:METSFedoraExt-1.1` (for METS), `info:fedora/fedora-system:ATOM-1.0` (for ATOM), or `default`.
- **econtext** - The export context (which indicates what use case the output should be prepared for. Valid options are: `public`, `migrate`, `archive`, or `default`.
- **path** - the export directory

- **protocol** - how to connect to repository, either http or https
- **context** - an `_optional_` parameter indicating the webapp context. This is only necessary if the Fedora server was installed under a context name other than 'fedora' (see [Alternative Webapp Context Configuration](#)).

Examples:

Export demo:1 for migration in FOXML 1.1 format (from example.com:80 to the current directory).

```
fedora-export example.com:80 fedoraAdmin fedoraAdmin demo:1 info:fedora/fedora-system:FOXML-1.1 migrate . http
```

Export all objects in the default export format and context (from example.com:80 to directory /tmp/fedoradump).

```
fedora-export example.com:80 fedoraAdmin fedoraAdmin DMO default default /tmp/fedoradump http
```

fedora-find

fedora-find [host] [port] [user] [password] [fields] [phrase] [protocol] [context]

- **user** - the Fedora user (e.g. fedoraAdmin)
- **password** - the Fedora user's password

Where:

- **host** - the hostname of the Fedora server; default is localhost
- **port** - the port number on which the Fedora server is running; default is 8080
- **fields** - A space-delimited list of fields. These are the fields that will be displayed for each object that matches the searchString. See <http://host:port/fedora/search> for a complete list of displayable fields and descriptions of each.
- **phrase** - A simple text string to search all fields for. This may include wildcard characters and is case-insensitive.
- **protocol** - how to connect to repository, either http or https
- **context** - an `_optional_` parameter indicating the webapp context. This is only necessary if the Fedora server was installed under a context name other than 'fedora' (see [Alternative Webapp Context Configuration](#)).

Example:

A simple way to search a repository's indexed fields. More advanced searches can be done with the web-based search interface at <http://host:port/fedora/search>

, or the Admin GUI's search interface (shows the pid, Fedora object type, title, and description fields of each object that has the word fedora somewhere in it's indexed fields).

```
fedora-find localhost 8080 "pid fType title description" "fedora" http
```

fedora-ingest

fedora-ingest f[file] [path] [format] [targetHost:targetPort] [targetUser] [targetPassword] [targetProtocol] [log] [context]

fedora-ingest d[ir] [path] [format] [targetHost:targetPort] [targetUser] [targetPassword] [targetProtocol] [log] [context]

fedora-ingest r[epos] [sourceHost:sourcePort] [sourceUser] [sourcePassword] [pid | *] [targetHost:targetPort] [targetUser] [targetPassword] [sourceProtocol] [targetProtocol] [log] [context]

Where:

- **f[file]** or **d[ir]** or **r[epos]** - indicates whether the ingest is from a file, directory, or repository as source.
- **path** - the local file or directory name
- **format** - the XML format of the ingest file(s). Valid options are: *info:fedora/fedora-system:FOXML-1.1* (for FOXML 1.1), *info:fedora/fedora-system:FOXML-1.0* (for FOXML 1.0), *info:fedora/fedora-system:METSFedoraExt-1.1* (for METS), or *info:fedora/fedora-system:ATOM-1.0* (for ATOM) Objects ingested in a format other than FOXML 1.1 format are updated to conform to FOXML 1.1 as part of the ingest process
- **sourceHost/targetHost** - the source or target repository's hostname
- **sourcePort/targetPort** - the source or target repository's port number
- **sourceUser/targetUser** - the id of the source or target repository user
- **sourcePassword/targetPassword** - the password of the source or target repository user
- **pid | *** - Either the identifier (PID) of the object to export from the repository OR * to indicate all objects from the source repository
- **sourceProtocol** - the protocol to communicate with source repository, either http or https
- **targetProtocol** - the protocol to communicate with target repository, either http or https
- **log** - the optional log message. If unspecified, the log message will indicate the source of the object(s)
- **context** - an `_optional_` parameter indicating the webapp context. This is only necessary if the Fedora server was installed under a context name other than 'fedora' (see [Alternative Webapp Context Configuration](#)). NOTE: This parameter can only be used if the 'log'

parameter has been used as well.

Examples:

Ingest obj1.xml (encoded in foxml1.1 format) from the current directory into the repository at myrepo.com:80 as user 'jane' with password 'jpw'. The log message will be system-generated, indicating the source path+filename.

```
fedora-ingest f obj1.xml info:fedora/fedora-system:FOXML-1.1 myrepo.com:80 jane jpw http
```

Traverse the entire directory structure of c:\archive, and ingests any file. It assumes all files will be in the FOXML 1.1 format and will fail on ingests of files that are not of this format. All log messages will be the quoted string.

```
fedora-ingest d c:\archive info:fedora/fedora-system:FOXML-1.1 myrepo.com:80 jane janepw http ""
```

Ingest the object whose pid is 'demo:1' from the source repository 'screpo.com:8081' into the target repository 'myrepo.com:80'. The object will be exported from the source repository in the default export format configured at the source. All log messages will be empty.

```
fedora-ingest r jrepo.com:8081 mike mpw demo:1 myrepo.com:80 jane jpw http http ""
```

Same as above, but ingests all data objects (type O).

```
fedora-ingest r jrepo.com:8081 mike mpw O myrepo.com:80 jane jpw http http ""
```

fedora-ingest-demos

fedora-ingest-demos [host] [port] [user] [password] [protocol] [context]

Where:

- **host** - the hostname of the Fedora server; default is localhost
- **port** - the port number on which the Fedora server is running; default is 8080
- **user** - the Fedora user (e.g. fedoraAdmin)
- **password** - the Fedora user's password
- **protocol** - how to connect to repository, either http or https
- **context** - an `_optional_` parameter indicating the webapp context. This is only necessary if the Fedora server was installed under a context name other than 'fedora' (see [Alternative Webapp Context Configuration](#)).

Example:

A convenient script to ingest all included demo objects for a new Fedora installation. See the Demo guide for descriptions of these objects. Note for Fedora 3.1 and below: This script should not be run until you have ensured that the hostname and port numbers in the demo objects have been corrected (with the `fedora-convert-demos` script) to use the actual hostname and port of your Fedora server.

```
fedora-ingest-demos localhost 8080 fedoraAdmin fedoraAdmin http
```

fedora-convert-demos



Information

This script is available in Fedora 3.1 and below only. In Fedora 3.2+, the demo objects don't need conversion because they use the special `"http://local.fedora.server/fedora/"` syntax where appropriate. This causes the Fedora server to *automatically* translate the URLs based on where it is hosted (e.g. <http://example.org/foo/>), at runtime.

fedora-purge

fedora-purge [host:port] [user] [password] [pid] [protocol] [log] [context]

Where:

- **host** - the hostname of the Fedora server; default is localhost

- **port** - the port number on which the Fedora server is running; default is 8080
- **user** - the Fedora user (e.g. fedoraAdmin)
- **password** - the Fedora user's password
- **pid** - the PID of the object to permanently remove
- **protocol** - the protocol to communicate with repository, either http or https
- **log** - an *optional* log message explaining the removal
- **context** - an *optional* parameter indicating the webapp context. This is only necessary if the Fedora server was installed under a context name other than 'fedora' (see [Alternative Webapp Context Configuration](#)).

Example:

Permanently removes an object from the repository.

```
fedora-purge localhost 8080 fedoraAdmin fedoraAdmin demo:6 http "It was just a test object"
```

fedora-modify

The fedora-modify command line utility enables the running of the batch modify utility from the command line. The default behavior is to only validate the directives. To actually execute them, you must specify a value for execute-directives. A sample file showing all the available directives is available in FEDORA_HOME/client/demo/batch-demo/modify-batch-directives.xml.

fedora-modify [host:port] [user] [password] [directives-filepath] [log-filepath] [protocol] [execute-directives] [context]

Where:

- **host:port** - the hostname and port of the target Fedora server
- **user** - the Fedora administrator username (e.g., fedoraAdmin)
- **password** - the password for the Fedora administrator user
- **directives-filepath** - the full path to the file containing the batch modify directives
- **log-filepath** - the full path to the file where logs will be written
- **protocol** - the protocol to communicate with repository, either http or https
- **execute-directives** - an *optional* parameter indicating whether to actually process the directives. This can have any value, and when specified, means that the directives should actually be executed. If unspecified, the directives will be validated only.
- **context** - an *optional* parameter indicating the webapp context. This is only necessary if the Fedora server was installed under a context name other than 'fedora' (see [Alternative Webapp Context Configuration](#)). NOTE: This parameter can only be used if the 'execute-directives' has been specified as well.

fedora-batch-build

The fedora-batch-build command line utility enables the running of the batch build utility of the Administrator GUI client from the command line. The batch build utility creates a "batch" of Fedora objects based on the specified template file and the corresponding directory of object-specific files. Refer to the documentation on the Batch Utility for more details on how to use the batch build utility.

fedora-batch-build [object-template-file] [object-specific-dir] [object-directory] [log-filepath] [log-format]

Where:

- **object-template-file** - the full path to the batch template file
- **obj-specific-dir** - the full path to the directory containing the object-specific files
- **object-directory** - the full path to the directory where the generated objects will be built
- **log-filepath** - the full path to the file where logs will be written
- **log-format** - the format of the log file. Valid values are text or xml.

fedora-batch-ingest

The fedora-batch-ingest command line utility enables the running of the batch ingest utility of the Administrator GUI client from the command line. The batch ingest utility ingests a "batch" of Fedora objects from the specified directory into the repository. Refer to the documentation on the Batch Utility for more details on how to use the batch ingest utility

fedora-batch-ingest [object-directory] [log-filepath] [log-format] [format] [host:port] [user] [password] [protocol] [context]

Where:

- **obj-directory** - the full path to the directory containing the objects to be ingested
- **log-filepath** - the full path to the file where logs will be written
- **log-format** - the format of the log file. Valid values are text or xml
- **format** - the XML format of the ingest file(s). Valid options are: *info:fedora/fedora-system:FOXML-1.1* (for FOXML 1.1), *info:fedora/fedora-system:FOXML-1.0* (for FOXML 1.0)
- **host:port** - the hostname and port of the target Fedora server
- **user** - the Fedora administrator username (e.g., fedoraAdmin)

- **password** - the password for the Fedora administrator user
- **protocol** - the protocol to communicate with Fedora server, either http or https.
- **context** - an `_optional_` parameter indicating the webapp context. This is only necessary if the Fedora server was installed under a context name other than 'fedora' (see [Alternative Webapp Context Configuration](#)).

fedora-batch-buildingest

The `fedora-batch-buildingest` command line utility enables the running of the batch build & ingest utility of the Administrator GUI client from the command line. The batch build & ingest utility creates a "batch" of Fedora objects based on the specified template file and the corresponding directory of object-specific files and then ingests them into the repository.

fedora-batch-buildingest [object-template-file] [object-specific-dir] [object-directory] [log-filepath] [log-format] [host:port] [user] [password] [protocol] [context]

Where:

- **object-template-file** - the full path to the batch template file.
- **obj-specific-dir** - the full path to the directory containing the object-specific files.
- **object-directory** - the full path to the directory where the generated objects will be built.
- **log-filepath** - the full path to the file where logs will be written.
- **log-format** - the format of the log file. Valid values are text or xml.
- **host:port** - the hostname and port of the target Fedora server.
- **user** - the Fedora administrator username (e.g., `fedoraAdmin`).
- **password** - the password for the Fedora administrator user.
- **protocol** - the protocol to communicate with Fedora server, either http or https.
- **context** - an `_optional_` parameter indicating the webapp context. This is only necessary if the Fedora server was installed under a context name other than 'fedora' (see [Alternative Webapp Context Configuration](#)).

fedora-validate-objects

The `fedora-validate-objects` command line utility runs a validation task against a set of objects in the specified repository. Access to the repository is provided via the `-serverurl`, `-username` and `-password` parameters. The set of objects is specified using either a "terms" phrase, a "query" sequence, or a "pidfile" (one of these is required, but more than one is not allowed). The "terms" and "query" parameters correspond to the strings used by the `FindObjects` method of [API-A-LITE](#). The PID file is a plain text file containing one PID per line, ignoring blank lines and comment lines (lines that start with '#'). The output of the validator can be controlled with the Log4J properties file. This can be used to suppress certain categories of messages or to restrict output by severity level.

fedora-validate-objects -serverurl [server-base-url] -username [user] -password [password] {-terms [terms] | -query query | -pidfile [path]} -logConfig [log4j-properties-file]

Where:

- **server-base-url** - the full URL used to connect with the Fedora server: e.g. <http://localhost:8080/fedora>
- **user** - the Fedora administrator username (e.g., `fedoraAdmin`)
- **password** - the password for the Fedora administrator user
- **terms** - a "terms" string, as for the "FindObjects" method of API-A
- **query** - a "query" string, as for the "FindObjects" method of API-A
- **pidfile** - the path to a text file containing the PIDs of the objects to be validated, one PID per line
- **log4j-properties-file** - the full path to a Log4J properties file. Optional.

```
# In this example, messages regarding objects with no content model are disabled.

# An appender for non-validator logging.
log4j.appender.STDOUT=org.apache.log4j.ConsoleAppender
log4j.appender.STDOUT.layout=org.apache.log4j.PatternLayout
log4j.appender.STDOUT.layout.ConversionPattern=%d{yyyy-MM-dd' 'HH:mm:ss.SSS} %p (%c) %m%n
log4j.rootLogger=INFO, STDOUT

# An appender for validator logging.
log4j.appender.VALIDATOR=org.apache.log4j.ConsoleAppender
log4j.appender.VALIDATOR.layout=org.apache.log4j.PatternLayout
log4j.appender.VALIDATOR.layout.ConversionPattern=%p [%c] %m%n

# The "root" of the Validator logging categories.
log4j.logger.Validator=INFO, VALIDATOR
log4j.additivity.Validator=false

# Set some categories for special treatment.
log4j.logger.Validator.NoContentModel=OFF
```

Example SOAP Client

Introduction

The Fedora API-A interface is implemented as a SOAP service that consists of the following methods:

- **DescribeRepository** – gets information about the repository, including version
- **GetDatastreamDissemination** – gets the content's of the specified datastream
- **GetDissemination** – gets a dissemination result
- **GetObjectHistory** – gets the change history of an object consisting of a list of timestamps indicating when object components were created or modified
- **GetObjectProfile** – gets object profile
- **FindObjects** – gets the requested ObjectFields on all objects in the repository matching the given criteria
- **ListDatastreams** – lists the datastreams in the specified object
- **ListMethods** – lists the methods in the specified object
- **ResumeFindObjects** – gets the next list of results from a truncated findObjects response

For more information on the method definitions, refer to the API descriptions located at [Fedora Repository Web Service Interfaces](#).

The Fedora API-A sample SOAP client is an example of a java servlet-based client that provides a front end to the Fedora Access API-A SOAP service. The sample client is designed to provide a *browser centric* view of the Fedora Access interface. Return types from the Fedora Access SOAP service are translated into a form suitable for viewing with a web browser; i.e., MIME-typed streams. Applications that can readily handle SOAP requests and responses would most likely communicate directly with the Fedora Access SOAP service rather than use a java servlet as an intermediary. *This servlet is provided as an example of how to construct a client that uses the Fedora Access API via SOAP and currently does not implement all methods found in API-A. As noted below, the sample Soap Client does not provide implementations for the methods FindObjects and ResumeFindObjects.*

Client Syntax

Input parameters for the servlet include:

- **action_** – name of Fedora service which must be one of the following:
 - *DescribeRepository* – gets description of the repository.
 - no parameters required
 - *GetDatastreamDissemination* – gets the contents of the specified datastream.
 - requires PID_
 - requires dsID_
 - *GetDissemination* – gets a dissemination result.
 - requires PID_
 - requires bDefPID_
 - requires methodName_
 - *GetObjectHistory* – gets object change history.
 - requires PID_
 - *GetObjectProfile* – gets object profile.
 - requires PID_
 - *FindObjects* – gets the requested ObjectFields on all objects in the repository matching the given criteria. **(NOT CURRENTLY IMPLEMENTED)**
 - *ListDatastreams* – lists the datastreams in the specified object.
 - requires PID_
 - *ListMethods* – lists the methods for the specified object.
 - requires PID_
 - *ResumeFindObjects* – gets the next list of results from a truncated findObjects response. **(NOT CURRENTLY IMPLEMENTED)**
- **PID_** – persistent identifier of the digital object
- **bDefPID_** – persistent identifier of the Service Definition object
- **methodName_** – name of the method
- **dsID_** – datastream ID
- **asOfDateTime_** – versioning datetime stamp (optional). Proper syntax is YYYY-MM-DDTHH:MM:SS where HH is 24-hour clock.
- **xml_** – boolean switch used in conjunction with GetBehaviorDefinitions, GetBehaviorMethods, GetObjectMethods, GetObjectProfile and DescribeRepository methods that determines whether output is formatted as XML or as HTML; value of "true" indicates that the results are to be returned as XML; value of false or omission indicates display in HTML table format. (optional)
- **userParms** – service methods may require or provide optional parameters that may be input as arguments to a service method; these parameters are entered as name/value pairs like the other servlet parameters (optional)

Note that all servlet parameter names that are implementation specific end with the underscore character ("_"). This is done to avoid possible name clashes with user-supplied method parameter names. As a general rule, user-supplied parameters should never contain names that end with the underscore character to prevent possible name conflicts.

Configuration

The sample client can be run as an integral part of the Fedora server or as a standalone client on a different machine or server. If running outside

the Fedora server, its only requirements are a servlet engine using Java Servlet API 2.4 or higher, the endpoint of a functional Fedora Server, and the servlet path mapping used to access the soap client. The endpoint and servlet path info are configured in the `soapclient.properties` file that is located in the WEB-INF directory of the soapclient webapp module in the Fedora distribution directory tree. The properties file specifies three variables:

- **fedoraEndpoint** – the URL of the Fedora API-A SOAP service; default is "http://localhost:8080/fedora/services/access"
- **soapClientServletPath** – the servlet path used to access the soapclient; default is "/soapclient/apia"
- **soapClientMethodParmResolverServletPath** – the servlet mapping used to access the MethodParmResolverServlet which is a utility servlet of the soap client; default is "/soapclient/getAccessParmResolver"

The web.xml file for the soap client webapp uses these default settings. If the webapp is ported to another servlet engine and any of the servlet mappings change, the web.xml and the soapclient.properties file must both be updated accordingly. The servlets will dynamically get the values in the soapclient.properties file so no changes should be needed in the source code.

When the Fedora server is built, it will automatically create the soapclient webapp module and place it in the Fedora distribution directory. The generated SOAP type libraries that are bundled with the client are tied to the WSDL type definitions defined for Fedora API-A. Any changes made to the WSDL for API-A will require rebuilding of the Fedora server and the soap client webapp module.

Note that if you change the host name or port number on which the Fedora server is running, you must also update the value for the fedoraEndpoint parameter in the soapclient.properties file to reflect the new values. Otherwise, the soap client will be unable to connect to the Fedora server running under the new host name or port number. If you are running the soap client bundled together with the Fedora server, you need to stop the Fedora server, update the soapclient.properties file, and then restart the Fedora server so the new property values will be recognized by the soapclient webapp.

Examples

Get information about the repository:

```
http://localhost:8080/soapclient/apia?action_=DescribeRepository
```

Get the thumbnail datastream (with DSID of DS1) in data object with a PID of demo:5

```
http://localhost:8080/soapclient/apia?action_=GetDatastreamDissemination&PID_=demo:5&dsID_=DS1
```

Get the Dissemination for a data object with a PID of demo:5 and associated behavior definition object with a PID of demo:1 and methodName of getThumbnail:

```
http://localhost:8080/soapclient/apia?action_=GetDissemination&PID_=demo:5&DefPID_=demo:1&methodName_=getThumbnail
```

Get the ObjectHistory for a data object with a PID of demo:5:

```
http://localhost:8080/soapclient/apia?action_=GetObjectHistory&PID_=demo:5
```

Get the ObjectProfile for a data object with a PID of demo:5:

```
http://localhost:8080/soapclient/apia?action_=GetObjectProfile&PID_=demo:5
```

List the datastreams for data object with PID of demo:5

```
http://localhost:8080/soapclient/apia?action_=ListDatastreams&PID_=demo:5
```

List the methods for data object with PID of demo:5

```
http://localhost:8080/soapclient/apia?action_=ListMethods&PID_=demo:5
```

Fedora Administrator

- [Introduction](#)
- [Starting Fedora Administrator](#)

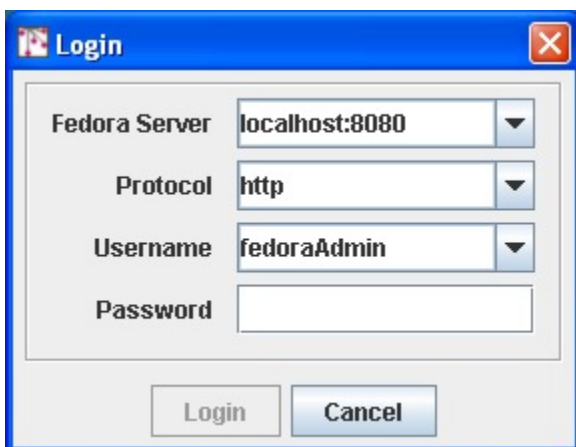
- File Menu
 - Creating a New Object
 - Create a New Data Object
 - View and Modify the Default Dublin Core Datastream
 - Create a New Datastream in the Object
 - Opening an Object for Viewing, Editing, Export, and Purge
 - The Object Properties Pane
 - The Datastreams Pane
 - Editing Datastream Content
 - Ingesting Objects
 - Ingest One Object
 - Ingest From File
 - Ingest From Repository
 - Ingest Multiple Objects
 - Ingest From Directory
 - Exporting Objects
 - Export One Object
 - Export Multiple Objects
 - Purging Objects
 - Viewing Object XML
 - Changing Repository
 - Exiting the Fedora Administrator
- Tools Menu
 - Searching and Browsing the Repository
 - Simple Search Tab
 - Advanced Search Tab
 - Search Results Window
 - Batch Processing
- Window Menu
- Help Menu
- Appendix A: Digital Object Construction

Introduction

Fedora Administrator is the direct link to API-M functionality for repository administrators. Using this tool it is possible to ingest, search for and retrieve, modify and purge objects.

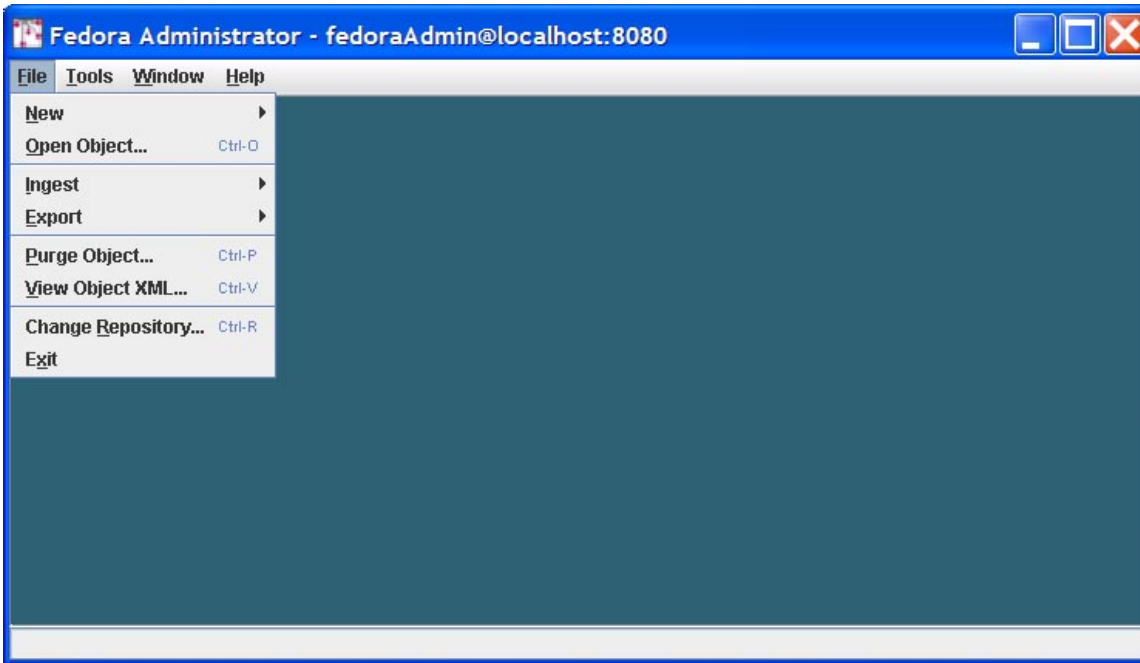
Starting Fedora Administrator

Navigate to the `$FEDORA_HOME/client/bin` directory and execute either the `fedora-admin.bat` batch file (for Windows) or the `fedora-admin.sh` script (for Unix/Linux).



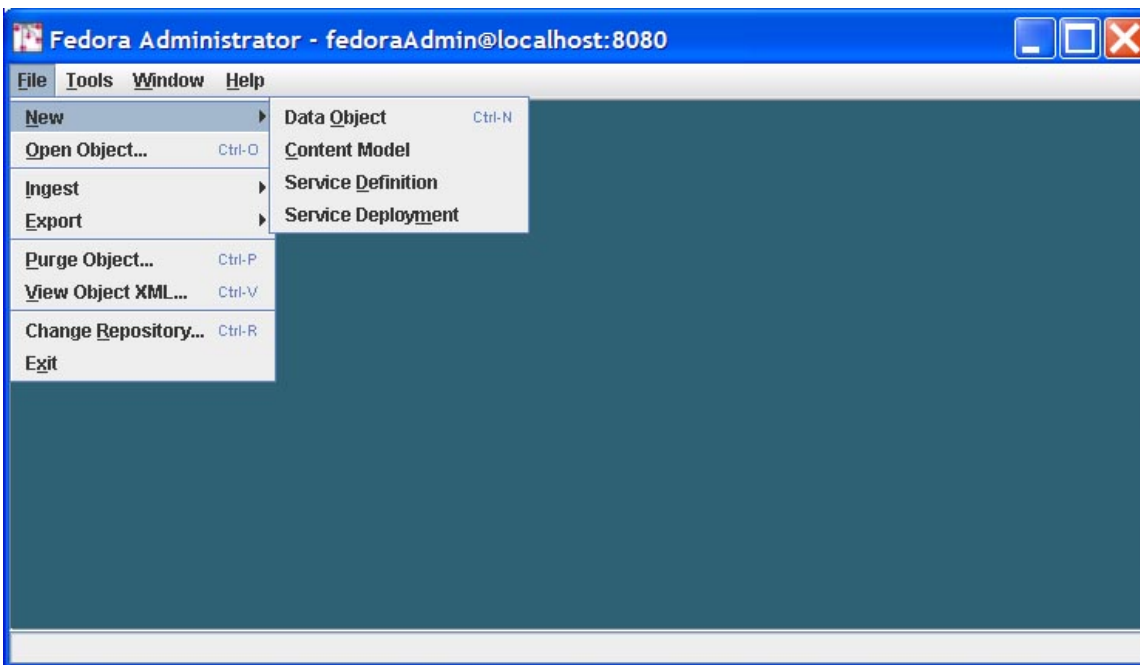
When you start the Fedora Administrator, you will be asked to choose the server to which you wish to connect, the protocol you wish to use to connect to your Fedora repository (http or https), your username and password. The server and username fields are pre-populated and the password is validated using values from `fedora.fcfg`.

File Menu



Commands on the file menu allow a repository administrator to perform operations on objects in the repository or to log in to a different repository.

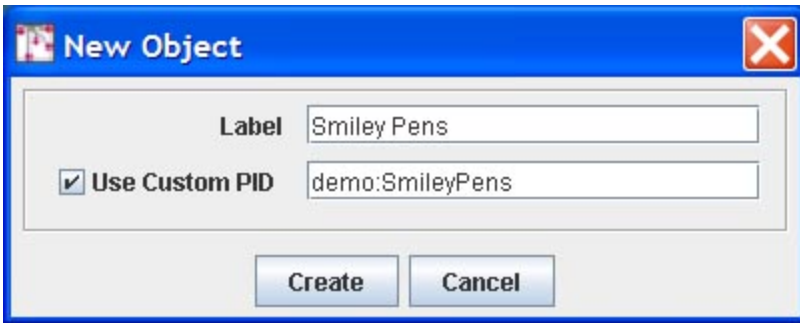
Creating a New Object



The *New* menu option allows users to build new Fedora objects from pre-existing component parts. When any of the object types (Data Object, Content Model, Service Definition, or Service Deployment) are created using the *New* option a skeletal Fedora object specific to that type is ingested into the repository. This allows the repository administrator or object owner to complete the object by defining metadata, asserting relationships, updating the provided datastreams, and adding new datastreams.

Create a New Data Object

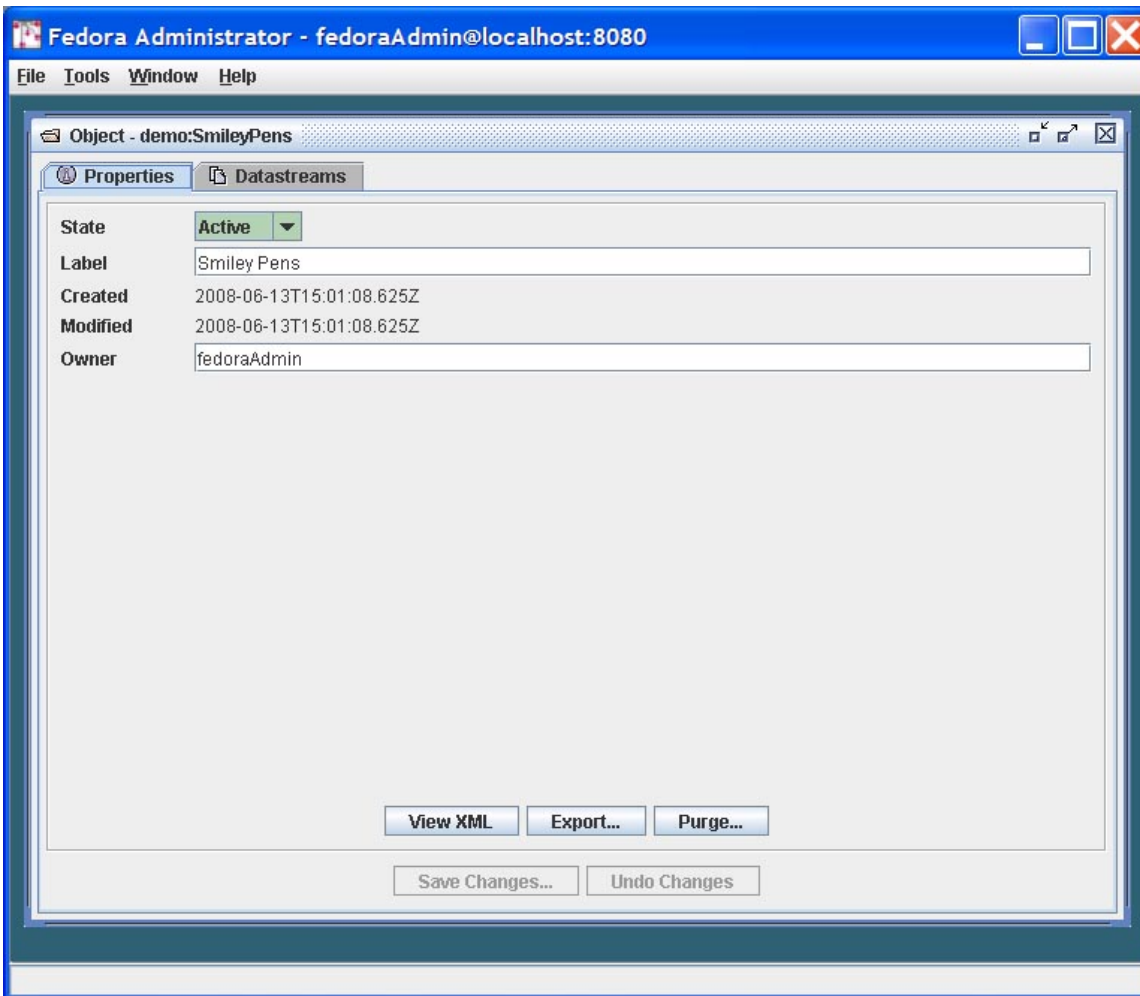
On the File Menu, select *New*.
From the New submenu, select *Data Object*.
The New Object Dialog appears.



Fill in the label for the object. If a custom PID is desired, click the check box and fill in the PID value.

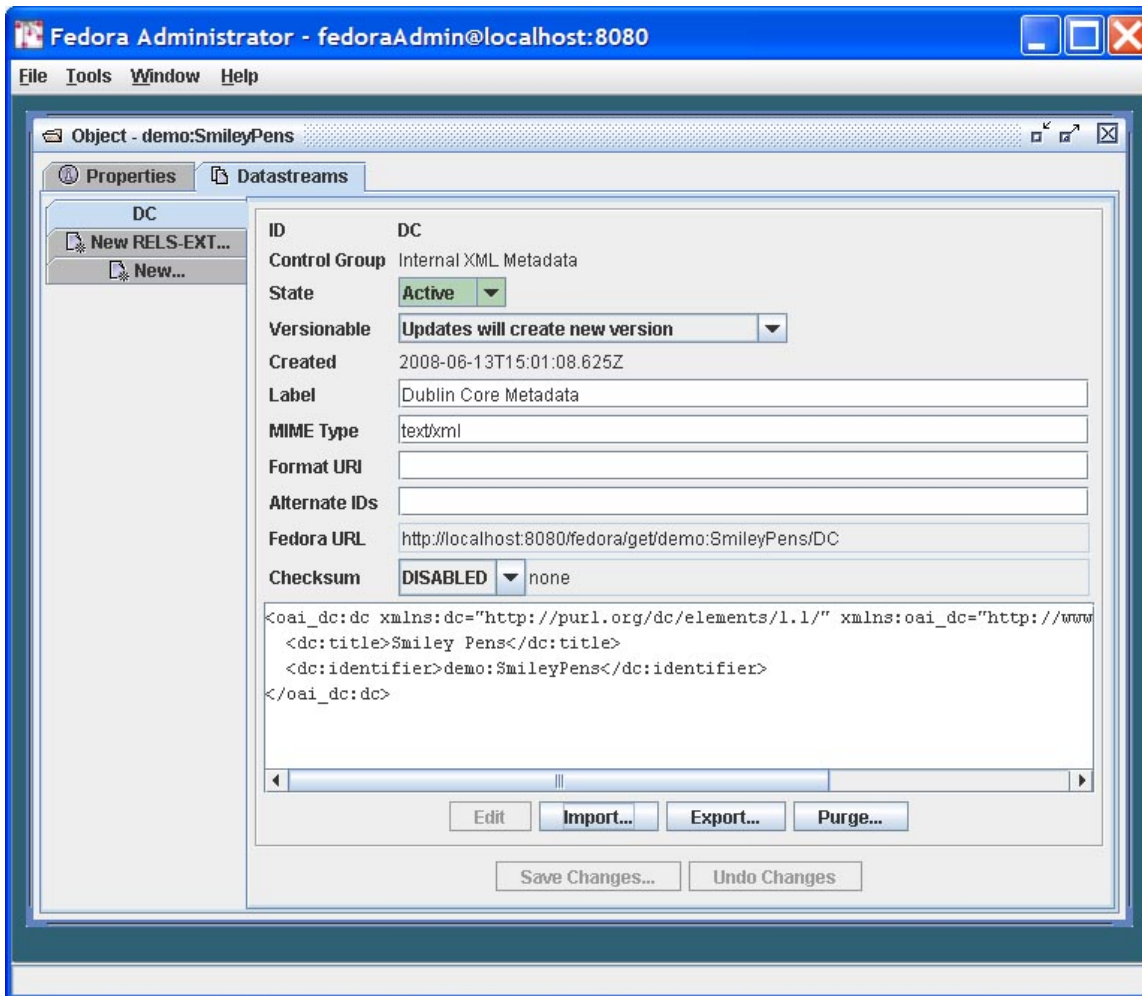
Clicking *Create* causes a Fedora data object to be created and ingested into the repository. The new object is displayed in a tabbed pane showing two tabs: *Properties and Datastreams*. You will first be presented with the Object Properties pane. On this pane you can add the following information:

- **State:** The state of the object, defined as follows:
 - *Active:* the object is available to users conditional upon any access control policy restrictions
 - *Inactive:* the object is only available to repository administrators.
 - *Deleted:* the object has been marked for permanent removal from the repository, pending review by repository administrators.
- **Label:** A description of the digital object
- **Owner:** An identifier for the owner of the object



View and Modify the Default Dublin Core Datastream

Click on the *Datastreams* tab labeled as "DC." The display will show a side tabbed pane displaying all datastreams currently in the object. When first created, all Fedora data objects will contain a default Dublin Core metadata datastream. You can edit this metadata datastream, adding fields as appropriate for the data object in question. To add new Dublin Core elements, edit the XML content for the datastream in the editing window. When done editing, click the Save Changes button on the bottom of the pane.



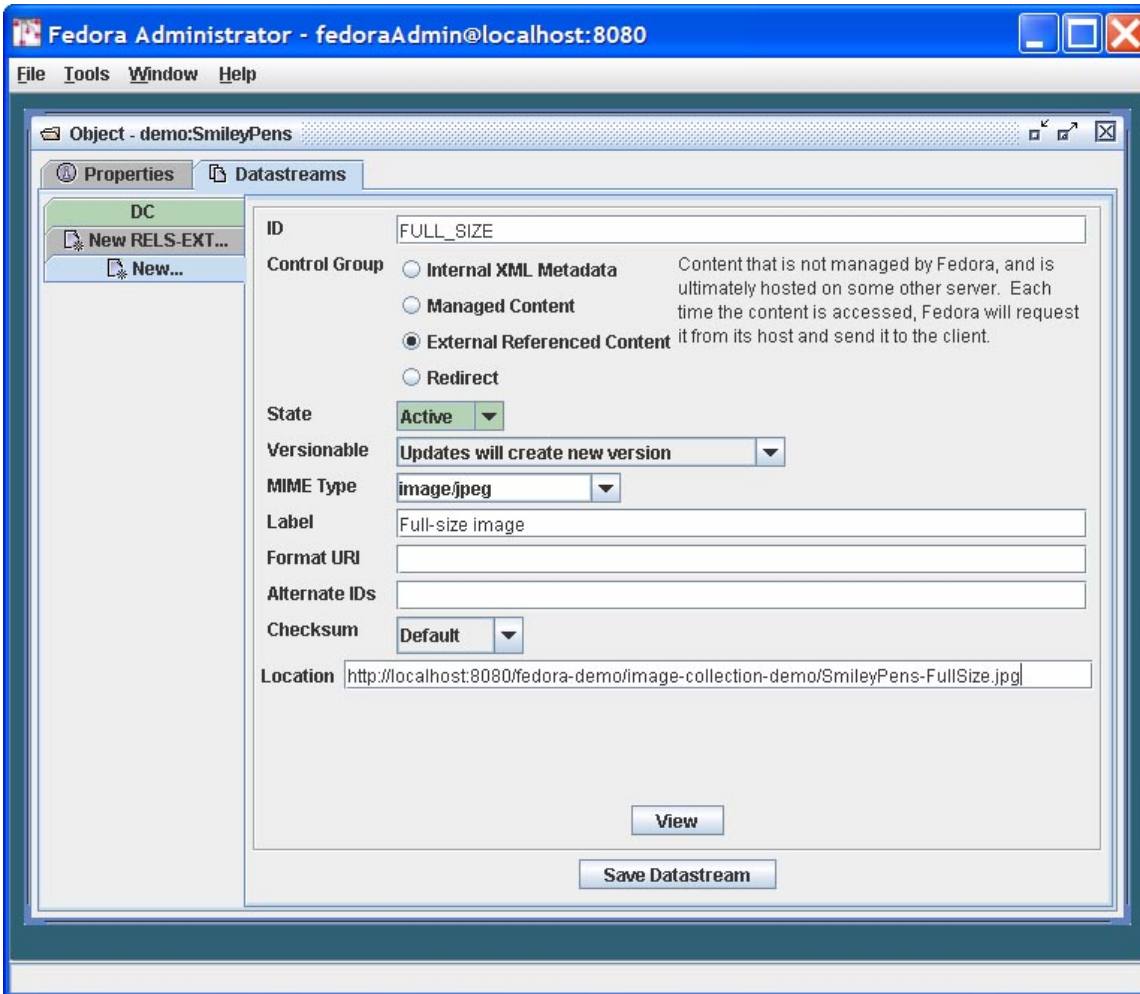
Create a New Datastream in the Object

To create a new datastream in the digital object, click on the side tab labeled "New..." You are presented with a dialog window that provides data input fields to enter all attributes for a datastream. Enter the following information:

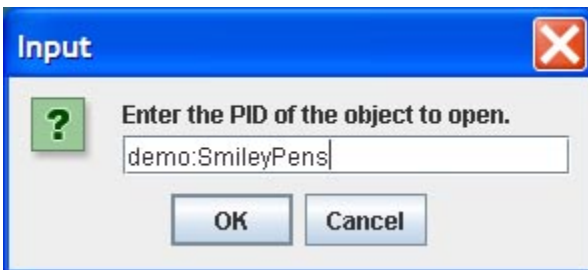
- **ID:** Enter a custom identifier for the datastream. If left blank, the system will automatically assign a unique datastream id.
- **Control Group:** Clicking through the control group options will show text defining each control group option. The Control Group lets you decide how you want the datastream content to be associated with the digital object. Datastream content can be in-lined within the object XML container file, it can be stored in the repository, or it can be stored external to the repository (and digital objects point to the external location).
 - **Internal XML Metadata:** In this case, the datastream will be stored as XML that is actually stored inline within the digital object XML file. The user may enter text directly into the editing window or data may be imported from a file by clicking *Import* and selecting or browsing to the location of the XML metadata file.
 - **Managed Content:** In this case, the datastream content will be stored in the Fedora repository and the digital object XML file will store an internal identifier to that datastream. To get content, click *Import* and select or browse to the file location of the import file. Once import is complete, you will see the imported file in a preview box on the screen.
 - **External Referenced Content:** In this case, the datastream content will be stored outside of the Fedora repository, and the digital object will store a URL to that datastream. The datastream is "by reference" since it is not actually stored inside the Fedora repository. While the datastream content is stored outside of the Fedora repository, at runtime, when an access request for this type of datastream is made, the Fedora repository will use this URL to get the content from its remote location, and the Fedora repository will mediate access to the content. This means that behind the scenes, Fedora will grab the content and stream in out to the client requesting the content as if it were served up directly by Fedora. This is a good way to create digital objects that point to distributed content, but still have the repository in charge of serving it up. To create this type of datastream, specify the URL for the datastream content in the Location URL text box.
 - **Redirect:** In this case, the datastream content is also stored outside the repository and the digital object points to its URL ("by-reference"). However, unlike the External Referenced Content scenario, the Redirect scenario signals the repository to redirect to the URL when access requests are made for this datastream. This means that the datastream will not be streamed through the Fedora repository when it is served up. This is beneficial when you want a digital object to have a datastream that is stored and served up by some external service, and you want the repository to get out of the way when it comes time to serve the content up. A good example is when you want a datastream to be content that is stored and served by a streaming media server. In such a case, you would want to pass control to the media server to actually stream the content to a client (e.g., video streaming), rather than have Fedora in the middle re-streaming the content out. To create a Redirect datastream, specify the URL for the content in the Location text box.

- **State:** Set the state for the digital object as Active, Inactive, or Deleted. The object states are defined as follows:
 - *Active:* the datastream is available to users conditional upon any access control policy restrictions
 - *Inactive:* the datastream is only available to repository administrators.
 - *Deleted:* the datastream has been marked for permanent removal from the repository, pending review by repository administrators.
- **Versionable:** You can decide on a datastream-by-datastream basis whether the Fedora repository will version the datastream when modification are made. By default all datastreams are versioned, but you can override this with the Versionable drop down settings. If you select "Updates will create new version," a new version of the datastream will be created when any change is made. All previous versions of the datastream will be maintained. If you select "Updates will replace most recent version" then any changes will overwrite the most recent version of the datastream. In the "replace" case, if there is already a previous version history of the datastream it will be maintained (it will not be lost if you change to the "replace" option at some point in the future after you have been versioning a datastream).
- **MIME Type:** Select the MIME type for the datastream content.
- **Label:** Give your datastream a descriptive label.
- **Format URI:** Optionally, you can provide a format identifier for the datastream. Examples of emerging format identifier schemes found in [PRONOM](#) and the [Global Digital Format Registry \(GDRF\)](#).
- **Alternate IDs:** Optionally, you can assign one or more alternate identifiers for you datastream. Such identifiers could be local identifiers or global identifiers such as [Handles](#) or [DOI](#).
- **Checksum:** The Fedora repository has a global setting for enabling datastream checksums to be calculated by the repository when a datastream is created or modified. See the Fedora system documentation of repository configuration for details. Also, refer to the system documentation on Checksum Features for a complete discussion of this functionality. A checksum is calculated on datastream content (it does not include the various attributes of a datastream such as its identifier, state, or mime type). In the datastream creation dialog pane, you will see a drop down list where you can select a checksum algorithm. This gives you some control for how you would like the checksum handled for a particular datastream. (Or you can let the default repository behavior take over.) The following can be accomplished using the checksum data entry box:
 - If you select "Default" in the checksum drop down list, the repository will handle everything based on how it is configured. This means that if the repository is configured to automatically checksum datastreams, it will use a default checksum algorithm configured for the repository, it will calculate a checksum for the datastream, and store it.
 - If you select a particular checksum algorithm from the drop down list box, you are essentially telling the repository that whatever it has configured for datastream checksum, you wish to override that behavior for this particular datastream. By specifying an algorithm you are telling the repository to calculate a checksum for that datastream using the algorithm that you specify.
 - When you select a checksum algorithm from the drop down box, you will notice that a text entry box appears next to it. You can leave the box blank, and the repository will use the specified algorithm and calculate a checksum values. Another option is to provide the repository with a checksum value that you have already calculated. In this case, the repository service will still calculate a new checksum when it creates the datastream, but if you provide a value from the client end, the repository will use it as an integrity check to ensure that the client and the server both agree on the checksum value for the datastream content. The repository will calculate a checksum value, using the algorithm you specify, and then it will compare its checksum calculation with the checksum value you provided in the text box. If they match, that means that the datastream content was not altered in the transmission from the client to the repository service. If they don't match, that indicates that something happened to the datastream during the transaction, and the repository will throw an exception.
- **What about the Datastream Content?** Depending on the datastream control group that you selected, the user interface will present you with a different way to get the actual content for your datastream.
 - If you selected "Internal XML Metadata" you will see an editing window at the bottom of the pane for creating XML content. Alternatively, you can use the "**Import**" button to insert XML content from a file.
 - If you selected "Managed Content" you will see an "**Import**" button on the bottom of the pane. Click this button to select a file whose content will be pulled into the repository and stored as managed datastream content. The digital object will have an internal pointer to the repository stored content.
 - If you selected "External Referenced Content" a new text entry box labeled "**Location**" will appear where you are prompted to enter a URL for content stored external to the repository. The digital object will store this URL, and the content will NOT be copied into the repository. It will remain "by-reference." You will notice a "View" button appear at the bottom of the pane that allows you to preview the content.
 - If you selected "Redirect" you will also enter a URL in the "**Location**" text entry box.

When all information is supplied in the add Datastream pane, click the **Save Datastream** button. This will send an API-M request to the Fedora repository service to add the new Datastream to the digital object. A completed Datastream pane is depicted below.



Opening an Object for Viewing, Editing, Export, and Purge



The *Open* menu requires input of an object PID for retrieval of the object. Upon retrieval, the object is displayed on two tabbed panes: Properties, Datastreams. You can edit any aspect of a digital object from these panes, in the same way information was added via the panes (as described earlier). Some additional functions are available on the Object Properties pane for an existing digital object:

The Object Properties Pane

- **Viewing XML:** From this window, the XML of the object can be viewed, but cannot be modified.
- **Export Object:** If Export is selected, the user will be prompted for a file name and location to which to write the XML file. The format for exported objects is fedora:mets.
- **Purge Object:** Purging an object completely removes it from the repository. Upon selecting the Purge option, the user will be prompted to enter a reason for the object's removal.



Warning

There is no "Are you sure" dialog with this option and the purge cannot be undone.

The Datastreams Pane

On the Datastreams Pane, the state of each datastream in the object can be modified, along with the datastream label and location. The MIME type of the datastream is shown, along with the control group, info type, create date, and the Fedora URL of the object.

- **Datastream States**

- *Active*: the datastream is freely available to all users.
- *Inactive*: the datastream is only available to repository administrators.
- *Deleted*: the datastream has been marked for permanent removal from the repository, pending review by repository administrators.

From this pane, users may additionally request to view a datastream, add a new datastream to the object, export a datastream's contents, or purge the datastream from the object.

Editing Datastream Content

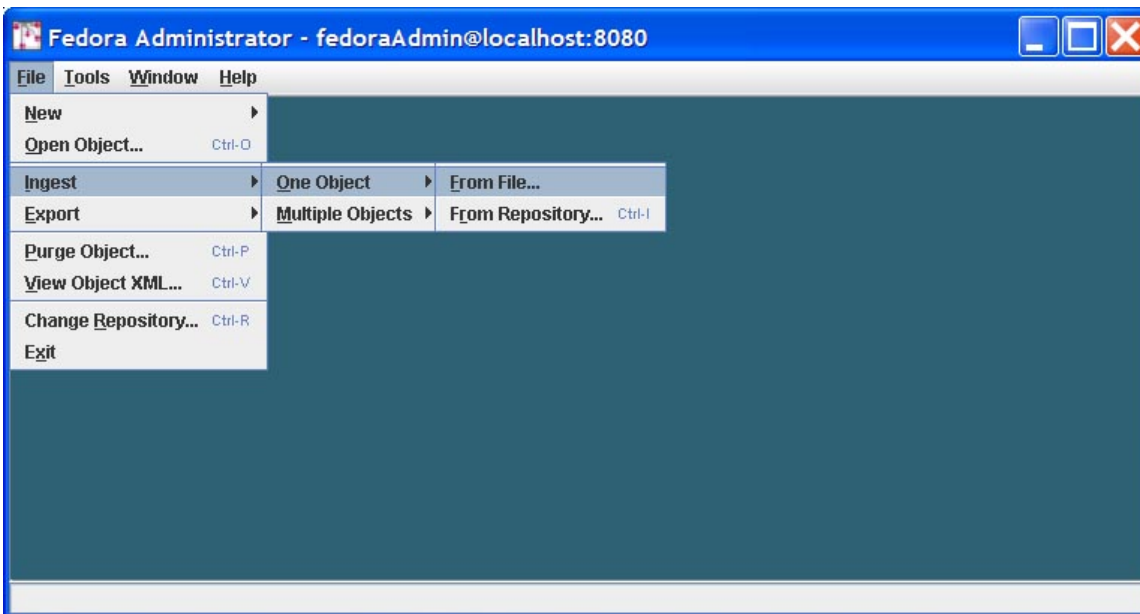
If a datastream has a *text* MIME type (e.g., *text/xml*), it may be edited in place by clicking the *Edit* button and making the desired changes in the editing window. Export the data contained in the datastream by clicking the *Export* button. Datastreams with non-text MIME types (e.g., *image* or *application*, e.g., *image/jpeg* or *application/pdf*) may only be viewed, exported, or purged from the object.

- **Import**: Choose this button to import new data by clicking the *Import* button. You will be prompted for a file name or url where the the import file is located.
- **Export**: Choose this button to export the content of a datastream. You will be prompted for a file name and location to which to write the XML file of the datastream content.
- **Purge**: Choose this button to remove a datastream from the object. You will be warned that the operation is permanent and must click "Yes" to continue. If "Yes" is selected, the datastream is immediately purged from the object. Purge cannot be performed on the DC metadata datastream.

Ingesting Objects

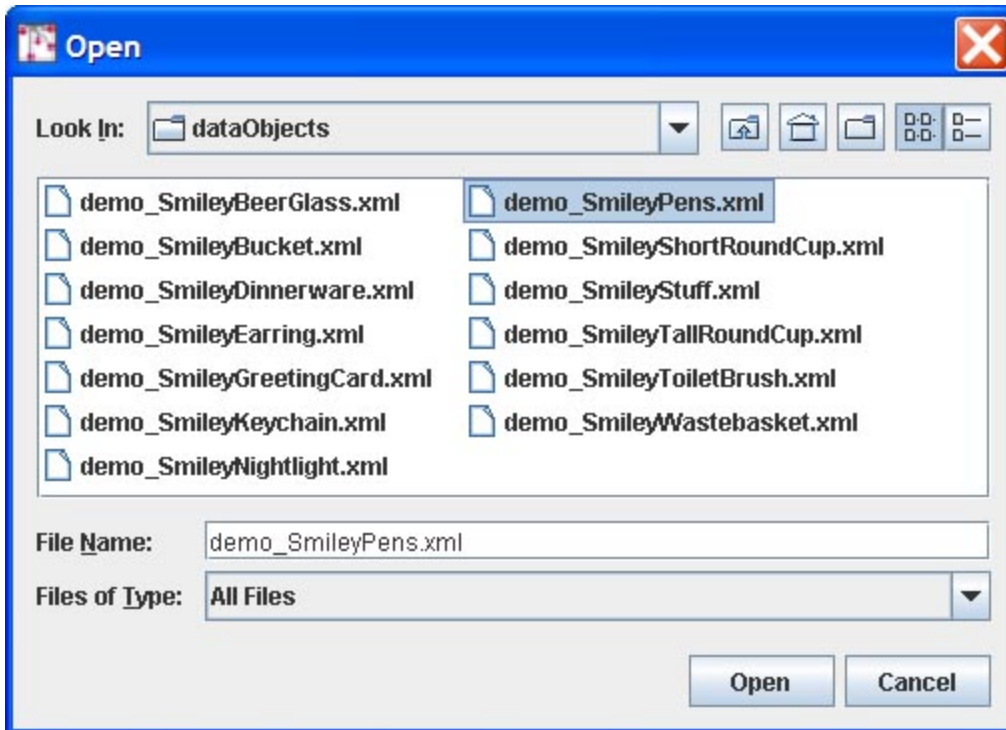
When selecting to ingest objects from the File menu, users have the option of ingesting a single object or multiple objects. Objects may be ingested from a file, directory, or from another repository.

Ingest One Object



Ingest From File

Choosing Ingest One Object from File, the user is prompted to select the file name from a dialog box or browse to the location of the file on the local drive(s) for the file to be ingested. Clicking *Open* will cause the file to be ingested. If the repository has been set to retain PIDs on ingest in *fedora.fcfig*, the PID in the object XML will be maintained. Otherwise, the PID will be overwritten.



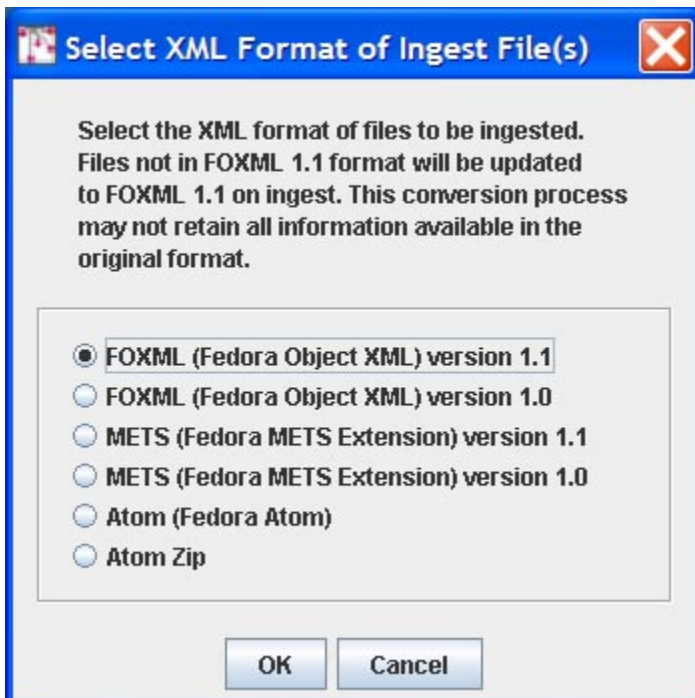
Ingest From Repository

Choosing Ingest One Object from Repository causes the Source Repository dialog box to appear. The user must fill in the hostname: port of the source repository, the protocol (http or https) and enter a username and password. Clicking *OK* initiates the Input dialog, where the user is prompted for a PID value. Clicking *OK* on the Input dialog completes the object ingestion.

Ingest Multiple Objects

Ingest From Directory

Upon selecting Ingest Multiple Objects From Directory, a dialog box prompts the user to select or browse to the directory containing the objects to be selected. Once the directory has been identified, clicking *Open* will activate a second dialog box which prompts the user to select the format of the objects to be ingested. All objects in the directory must be in the same format.

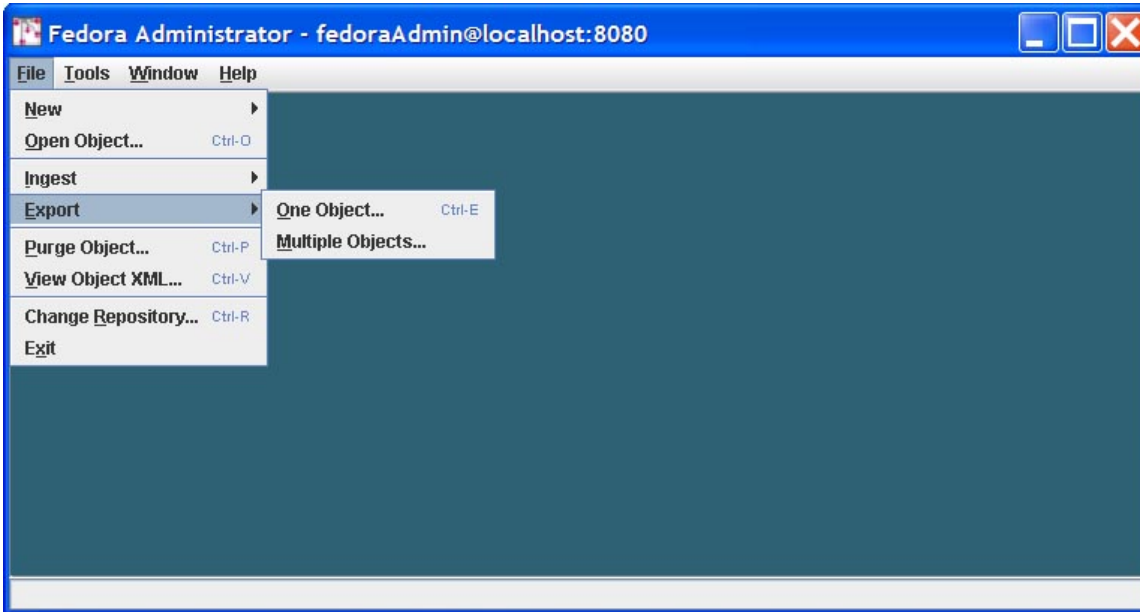


If the repository has been configured to "retain PIDs" on ingest (see the Fedora repository configuration documentation), then whatever PID is found in the ingest XML file (also known as the Submission Information Package or SIP) is what the repository accepts as the PID for the new ingested digital object. If the "retain PIDs" option is not enabled for the repository, the ingest function will automatically assign a new PID to the digital object, and ignore whatever PID was in the ingest XML file.

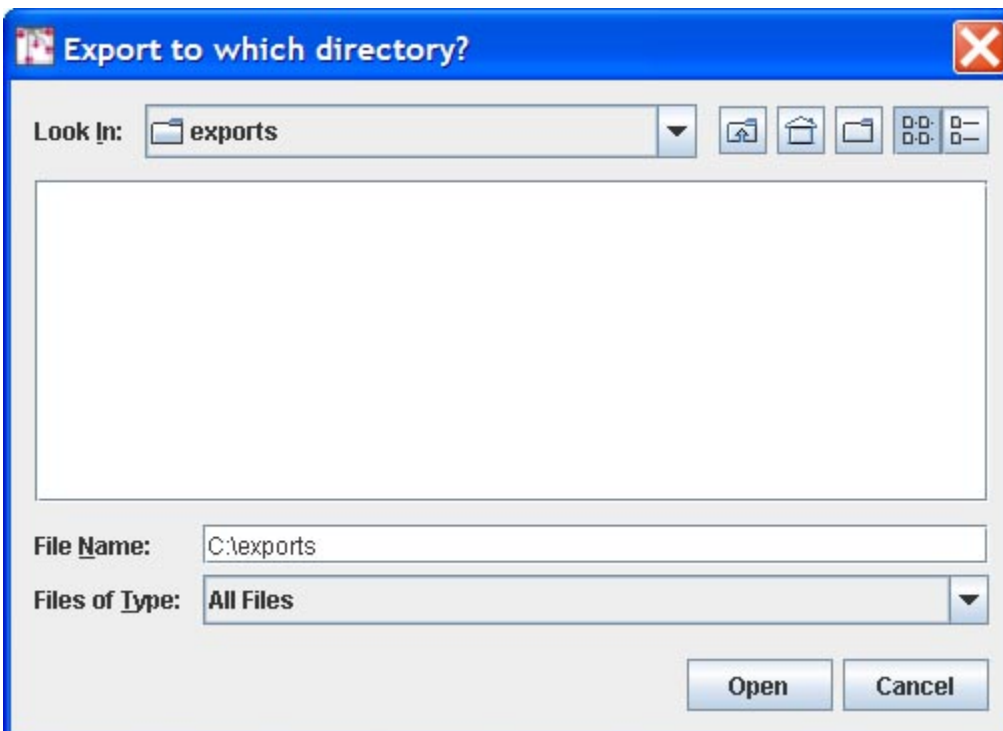
During ingest, the status bar at the bottom of the Fedora Administrator window shows the activities of the server. Once ingest is complete, a summary pane will appear giving counts of objects successfully ingested, objects failed, and time elapsed. Click *OK* to clear this message. The View Ingest Log dialog will then open. The user may click *Yes* and view the detailed log file or *No* to view the file at a later time. The log file is created in the `$FEDORA_HOME/client/logs/` directory.

Exporting Objects

Users have the option of exporting a single object or exporting multiple objects. This functionality can be accessed through the File/Export menu.



Export One Object



When the *Export One Object* option is selected, the user is prompted to select a directory to which the export file will be written. Clicking *Open*

causes the user to be prompted for an object PID. The PID will be used as the basis for the export file (named like *pid.xml*). Before the export is done, you must first provide a few more pieces of information. First, you must choose the export serialization format:

- FOXML 1.1 (the most current FOXML format)
- FOXML 1.0 (the FOXML format used with pre-3.0 Fedora repositories)
- METS 1.1 (the most current Fedora extension of METS)
- METS 1.0 (the METS format used with pre-3.0 Fedora repositories)
- ATOM (the Fedora extension of Atom)
- ATOM ZIP (an ATOM based format which packages all datastreams along with the object XML in a ZIP file)

Next you must choose the "export context" which will create an appropriate export file for the context in which you plan to use it. There are three export context types to choose from:

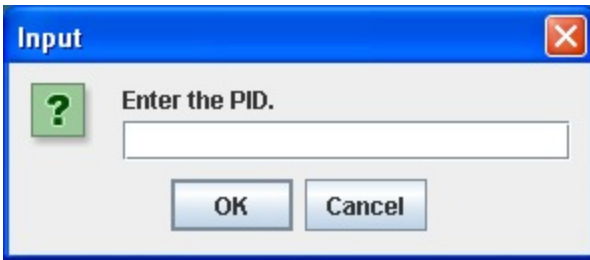
- **Migrate:** an export file will be create that is appropriate for migrating into another Fedora repository. This means that any references to the repository's base URL (with its host and port) will be virtualized, so that if the export object is subsequently ingested into another Fedora repository, the object will inherit the base URL of the new repository. The result is that repository-referential assertions in the exported digital object will naturally adapt to the new repository environment.
- **Public Access:** an export file will be created that is appropriate for facilitating public access and re-use of the digital object. All datastream locations will be resolvable URLs, most notably, the Managed Content datastreams will contain public callback URLs to the repository from which the object was exported. The assumption is that the repository will still exist, and that the original object will continue to be served up by the repository (so the callback URLs continue to work).
- **Archive:** an export file will be created with all managed content contained within it. This means that all Managed Content datastreams will have the datastream content Base64-encoded within the digital object export file. For Referenced and Redirected datastreams, their by-reference URLs will be in the export file (meaning that the repository will not pull the external content into the export file, but keep it by-reference). For datastreams that were of type "Inlined XML" the XML will remain in-lined as it normally is the the digital object XML file.




Export Multiple Objects

When the *Export Multiple Objects* option is selected, the user is prompted to select a directory to which the export files will be written. Clicking *Open* causes the user to be prompted for the export serialization format, as described above. When the format has been selected and *OK* is clicked, all objects are exported into the selected directory in the selected format type. The files are named based on their pid values.

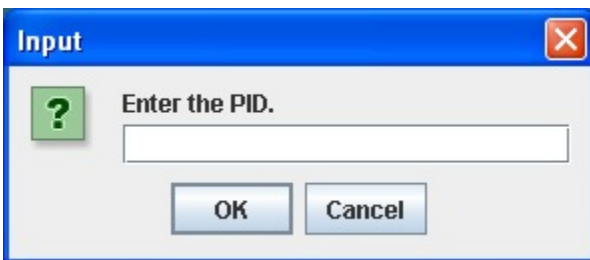
Purging Objects



Purging an object completely and permanently removes it from the repository. Upon selecting the Purge option, the user will be prompted to enter an object PID and a reason for the object's removal.

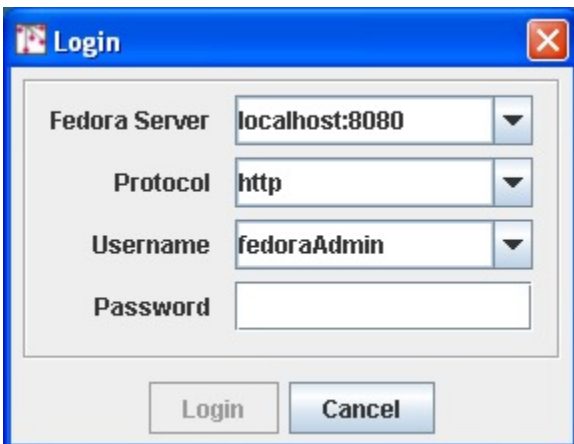
 **Warning**
There is no "Are you sure" dialog with this option.

Viewing Object XML



This menu option allows a user to view the xml, but not edit. XML may be cut and pasted into another application using standard keyboard commands of the host operating system, (e.g. CTL-C, CTL-V in Windows) if desired.

Changing Repository

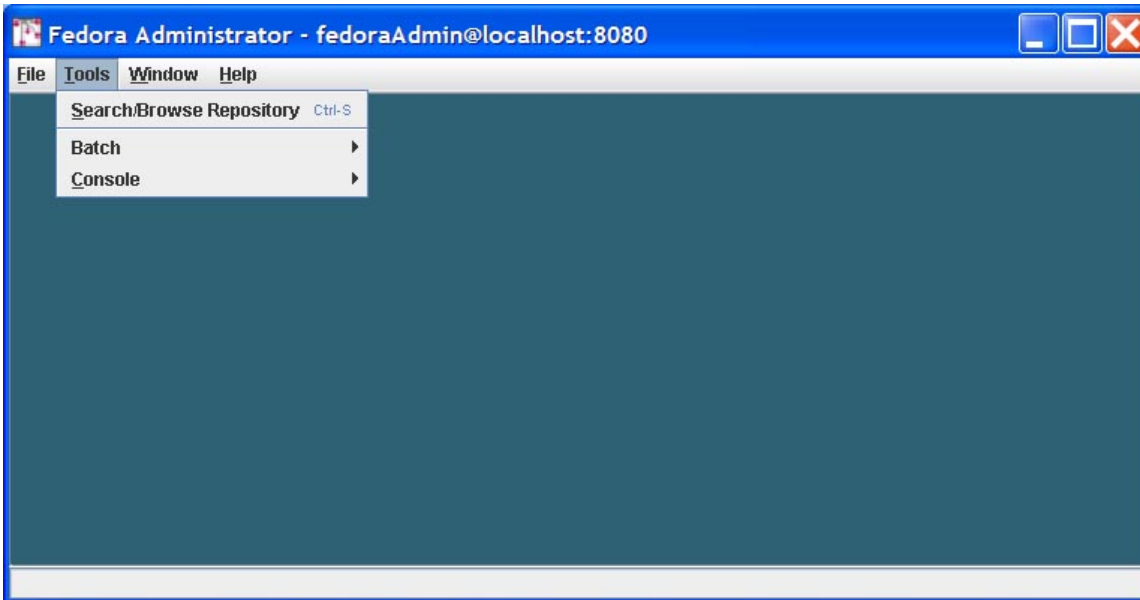


The Change Repository option allows a repository administrator to login to a different Fedora repository. When selected, this menu option causes the Login dialog to be displayed. The repository administrator may then select a different Fedora server with which to connect, entering the appropriate login name and password.

Exiting the Fedora Administrator

The Exit menu option closes all connections with the Fedora server instance and logs the user out of the repository.

Tools Menu

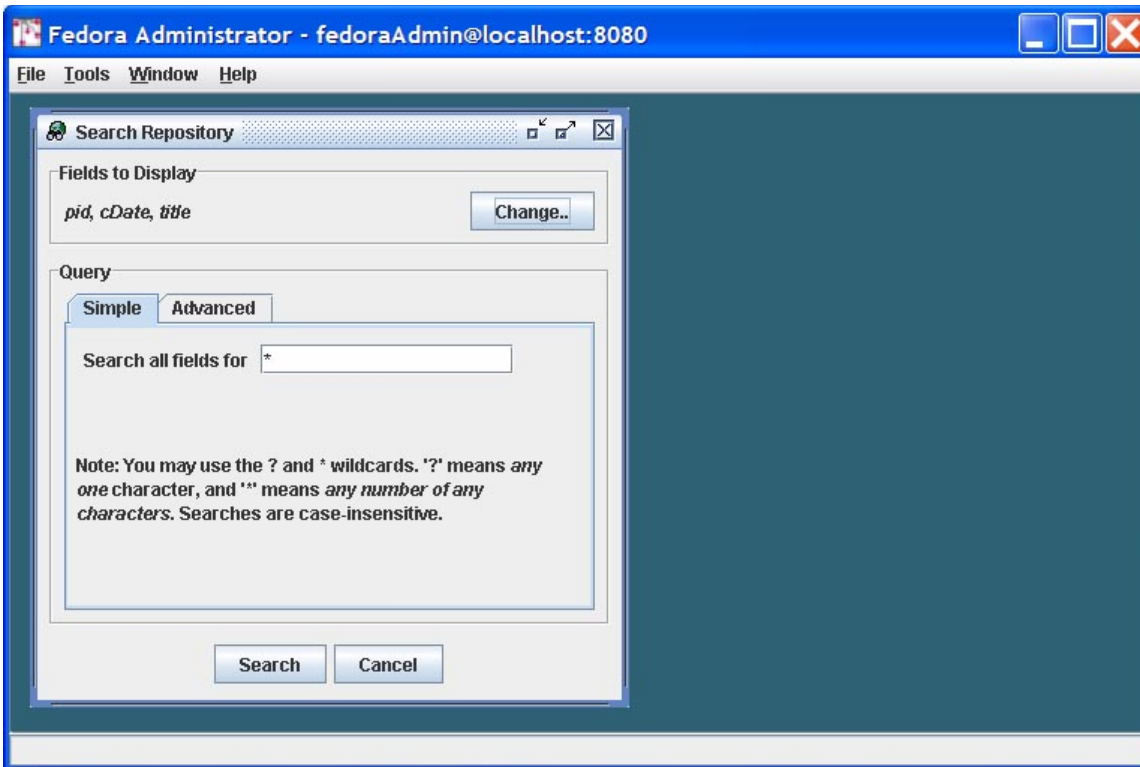


Commands on the Tools menu provide the user with the ability to search and retrieve objects from the repository, build and ingest batches of digital objects, and under the console submenu, gain access directly to API-M and API-A methods for testing purposes.

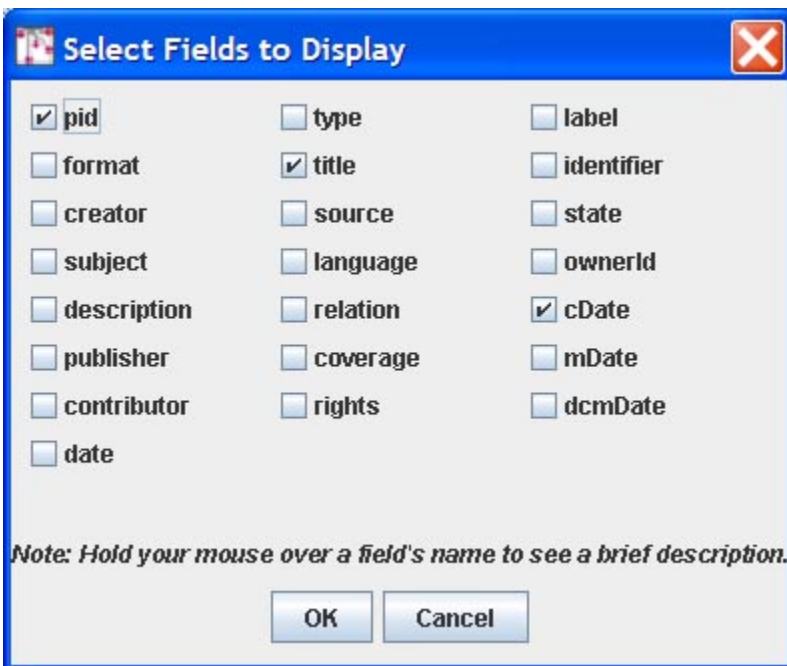
Searching and Browsing the Repository

The Search/Browse Repository menu option provides a mechanism for searching and retrieving objects from the Fedora repository. Upon ingestion, metadata from the Fedora System Metadata section and the Dublin Core (DC) Metadata section of the object are indexed in a relational database, and may be searched using this menu option. The DC Metadata section is an optional Implementer-Defined XML Metadata datastream in the object, where the *Datastream ID* is DC, and the XML conforms to the schema at http://www.openarchives.org/OAI/2.0/oai_dc.xsd. If a Dublin Core metadata datastream is not provided, Fedora will construct a minimal DC datastream consisting of the elements dc:title and dc:identifier. The value for dc:title will be obtained from the object's label (if present in the object) and the value for dc:identifier will be assigned to the object's persistent identifier or PID. The search interface provides both simple and advanced searching. All queries are case insensitive. Simple Search enables queries of words and phrases occurring anywhere in an object's indexed metadata fields. Advanced Search enables fielded searching across any combination of metadata elements using string comparison operators (= and ~) for string fields, and value comparison operators (=, >, >=, <, <=) for date fields (dc:date fields may be treated as both). The wildcards, * and ? may be used in any string-based query.

Simple Search Tab

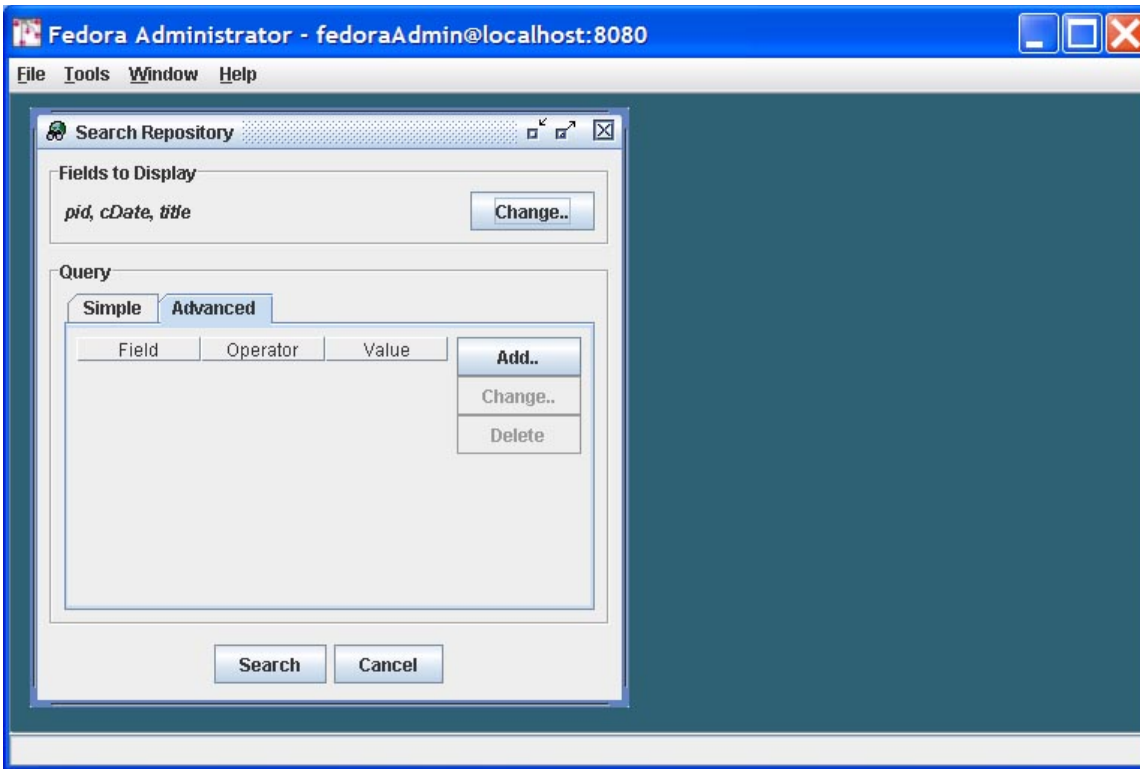


The Simple Search tab is the default selection in the Search Repository window. The Simple Search query searches both the Dublin Core metadata and the Fedora System Metadata fields. At the top of the Search window, the user may select fields to be displayed by clicking the *Change* button and selecting/deselecting field names from the dialog.

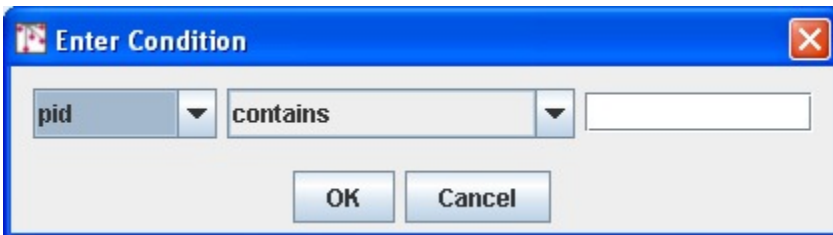


The Simple Search searches all indexed metadata fields for the text entered into the text box. All searches are case insensitive. The wildcard character '*'; can be substituted for any string of characters. The wildcard character '?'; can be substituted for any single character. Clicking *Search* will retrieve a list of objects where the entered text string appears in an indexed metadata field.

Advanced Search Tab



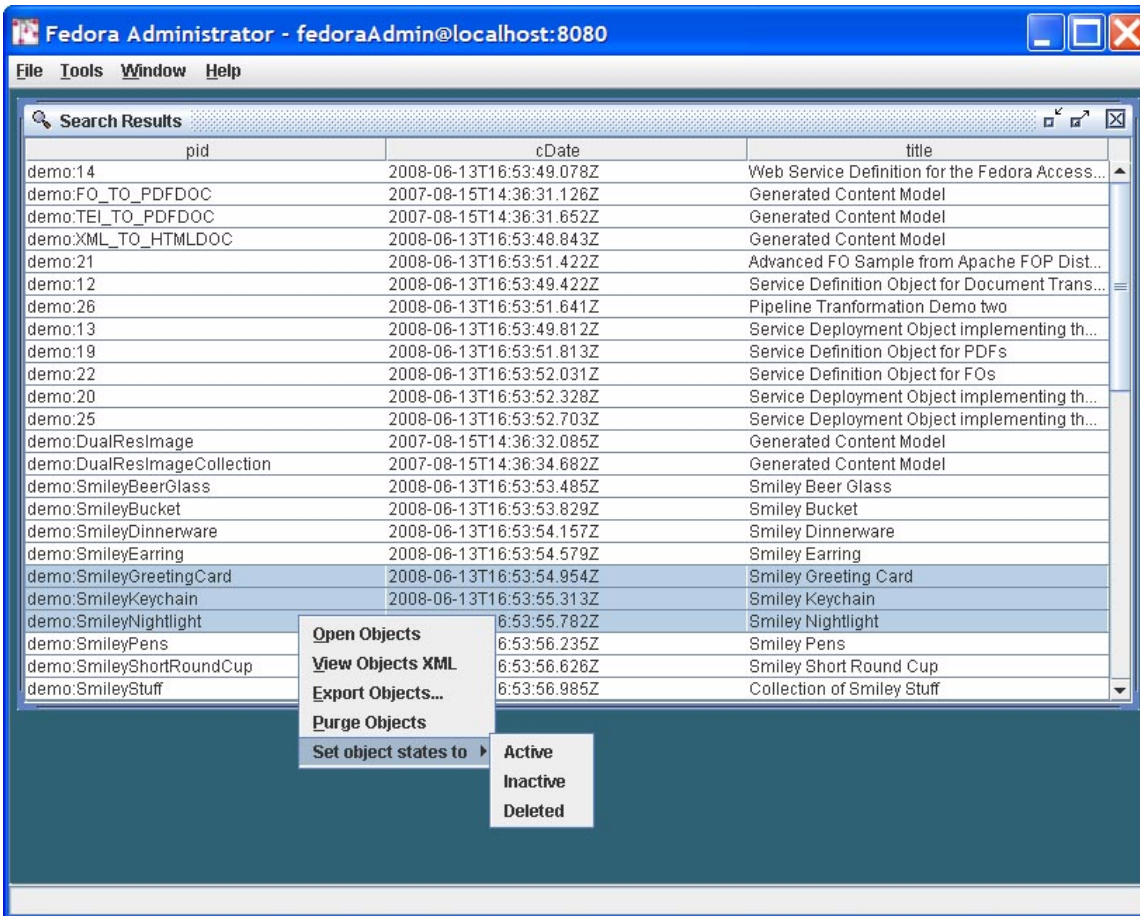
The Advanced Search query enables users to refine their repository search by searching specific fields for specific values provided in the query.



The search conditions can be modified by clicking the *Add* button, which opens the Enter Condition dialog. The user selects the field to be defined from the drop down menu, selects the condition to be matched, and enters the text to be matched, if appropriate. Clicking *OK* saves the condition. Once all conditions are entered, clicking *Search* will retrieve a list of objects in which all conditions are met.

Search Results Window

The Search Results Window displays the results of a successful search in a table format. Across the top of the table are a row of labels of the fields that have been returned from the objects meeting the search criteria. Double clicking anywhere on a row opens that object. Right clicking anywhere on a row opens a pop-up menu that contains object level tasks from which the user may select. These tasks include *Open Object*, *View Object XML*, *Export ...*, *Purge*, and *Set Object State To*. If *Set Object State To* is selected, a submenu will provide the user with valid states from which to select.

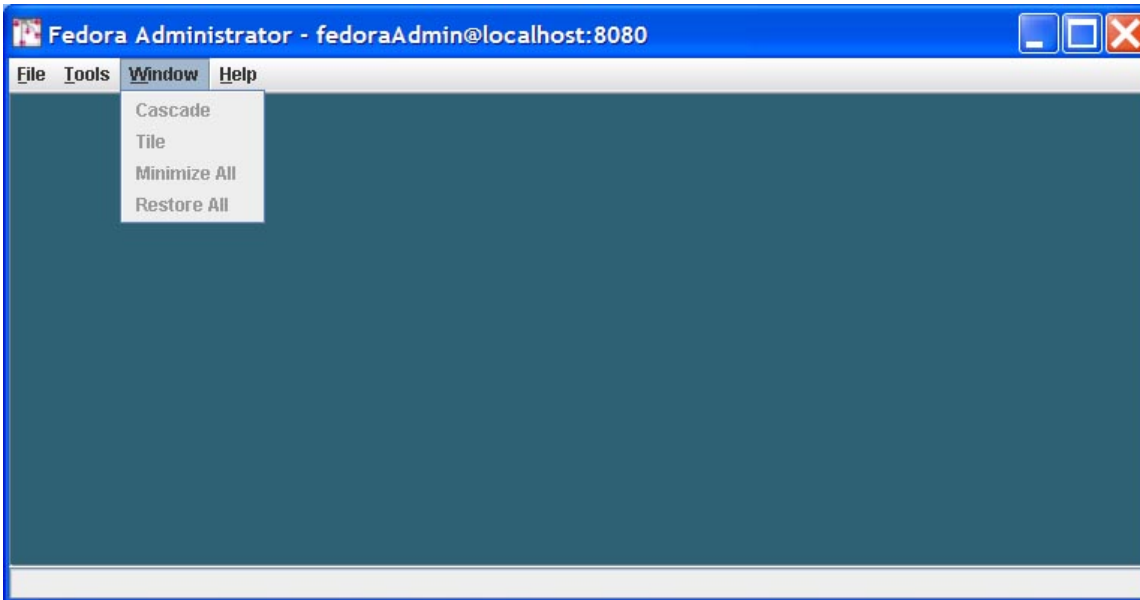


The *Purge* and *Set Object State To* submenu options can be used on multiple objects by using mouse clicks or the equivalent keyboard commands to multiselect rows in the Search Results Window. In this way, groups of objects can be purged from the system, or have their states changed by means of one search and retrieval operation.

Batch Processing

The Batch menu item includes tools to create and update multiple Fedora objects. For more information on batch processing, see [Batch Processing](#).

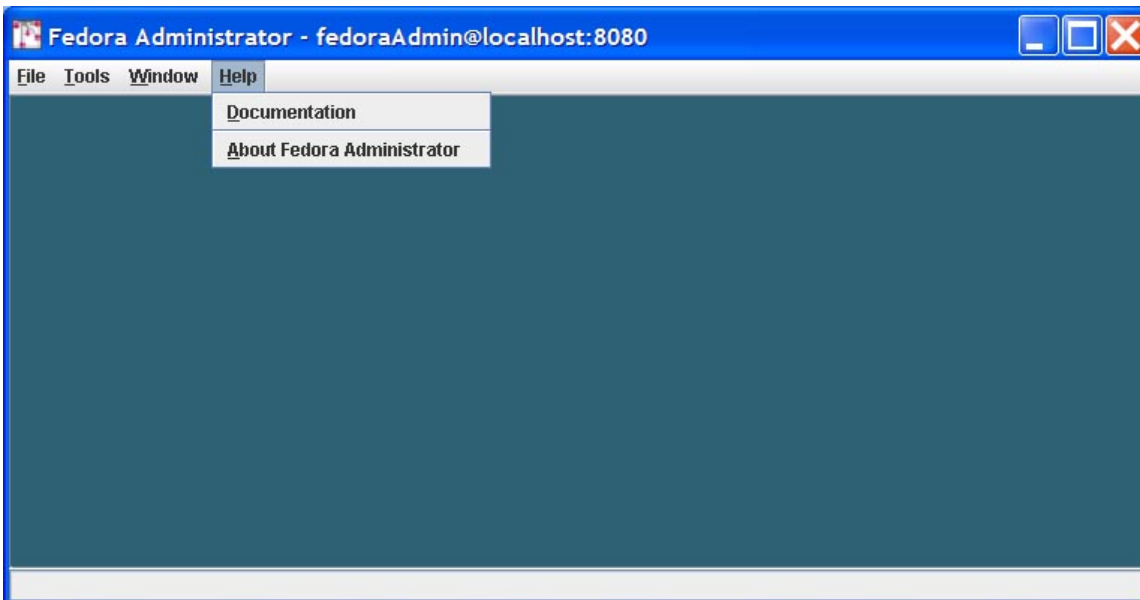
Window Menu



The Window menu contains standard commands for managing multiple panes open in the Fedora Administrator window. These include:

- Cascade
- Tile
- Minimize All
- Restore All

Help Menu



The help menu has two options:

- **Documentation** - which gives users the URL to the online documentation for Fedora.
- **About Fedora Administrator** - which provides version information, and copyright and licensing notices.

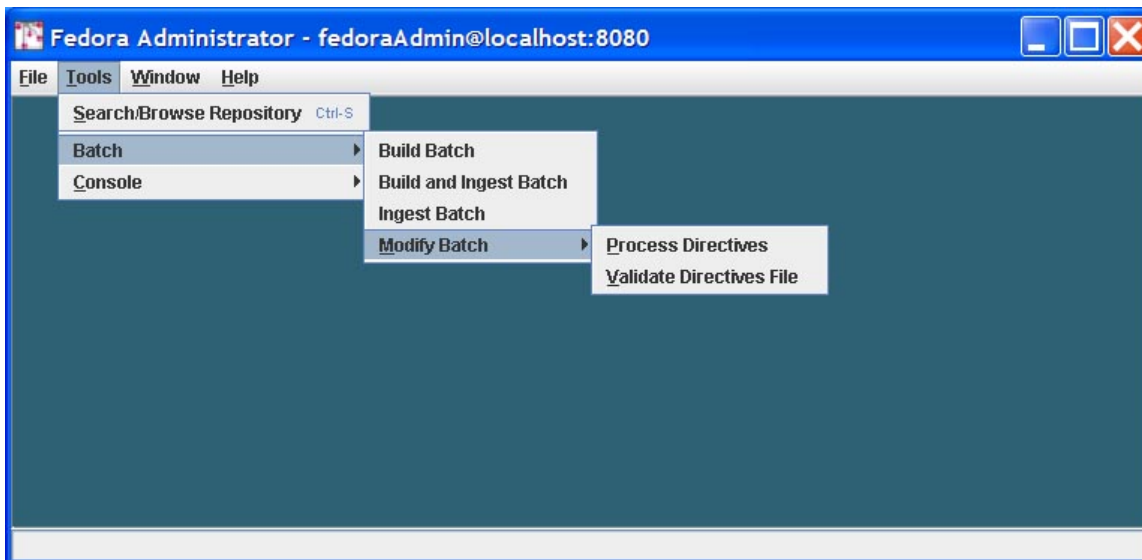
Appendix A: Digital Object Construction

This manual's focus is on practical use of the Fedora Administrator Tool. For further discussion on the theory behind digital object construction, please see the [Fedora Digital Objects](#).

Batch Processing

- Batch Processing
- Batch Ingest
- Building Fedora objects in Batch
- Fedora Administrator Instructions
- To Demo
- Ingesting Fedora Objects in Batch
- Fedora Administrator Instructions
- To Demo
- Building and Ingesting Fedora objects in Batch
- Fedora Administrator Instructions
- To Demo
- Object Processing Map
- XML Format
- Text Format
- Object-Specifics
- Progress Report File
- Batch Modify
- Process Directives
- Validate Directives
- To Demo
- Sample Modify Directives File
- Command Line Alternatives

Batch Processing

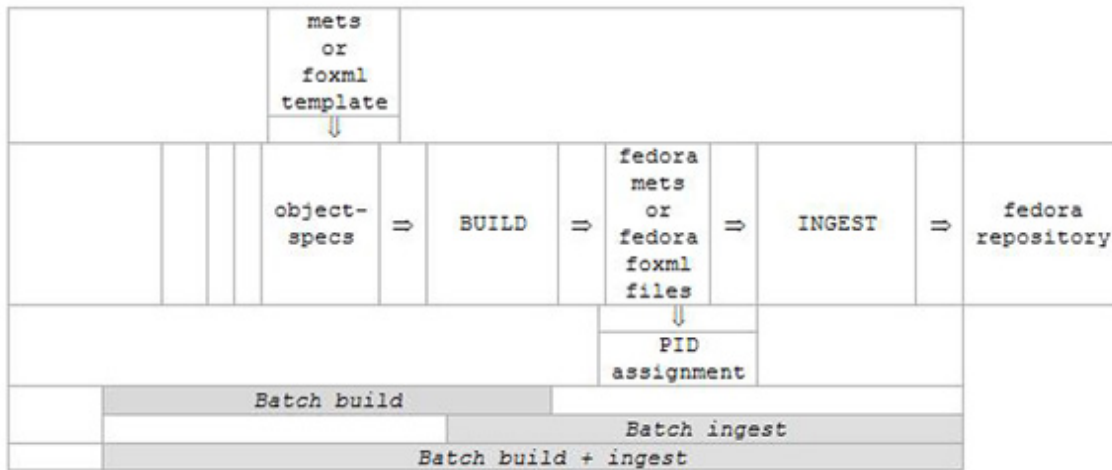


The Batch menu item includes tools to create and ingest multiple Fedora objects, which are FOXML documents or Fedora-specific METS documents contained in files outside the repository. The Batch Modify tool is also included here and allows users to modify batches of already-ingested Fedora objects.

Batch Ingest

It's simple to ingest objects created by one-up edit or by custom scripting. The Batch menu also supports building objects. This takes a general template common to all objects in a batch and makes object-specific substitutions into the template. The template can be either a Fedora METS XML document or a Fedora FOXML document and contains data common to the objects of the batch. Separate XML documents hold the per-object substitution values. The format of the template (FOXML or METS) determines the format of the built objects. The relatedness of objects in a batch is defined by what Fedora Administrator allows to be substituted and by which substitutions you choose to make. Data from the template are retained, unless replaced per individual object, including XML comments.

Fedora Administrator provides for three modes of object batch processing: **batch build**, **batch ingest**, and a combined **batch build and ingest**. This phased processing is shown in the following diagram:



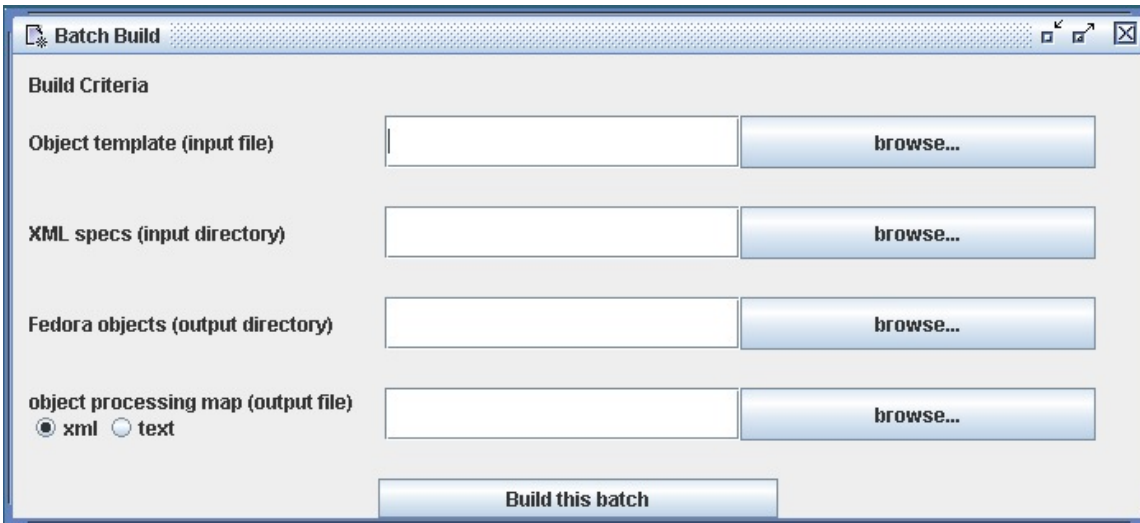
Building Fedora objects in Batch

Build a set of Fedora METS XML or Fedora FOXML XML files from a common Fedora METS or Fedora FOXML template and simple (non-METS) XML *object-specs*. The resulting objects are then ready for ingesting into Fedora.

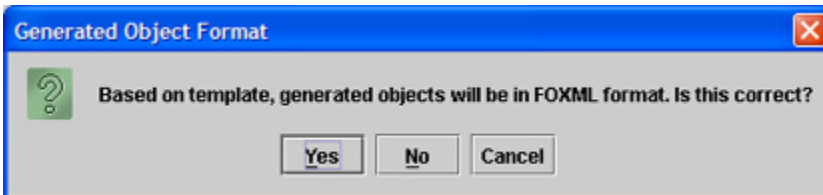
Fedora Administrator Instructions

Select *Tools* on the Fedora Administrator menu bar, select item *Batch* and then *Build Batch*. This will open a *Batch Build* window. You may need to adjust this window's size to see its controls. Use the *browse* buttons to enter the four required settings. Clicking on a *browse* button opens a standard directory/file selection dialog.

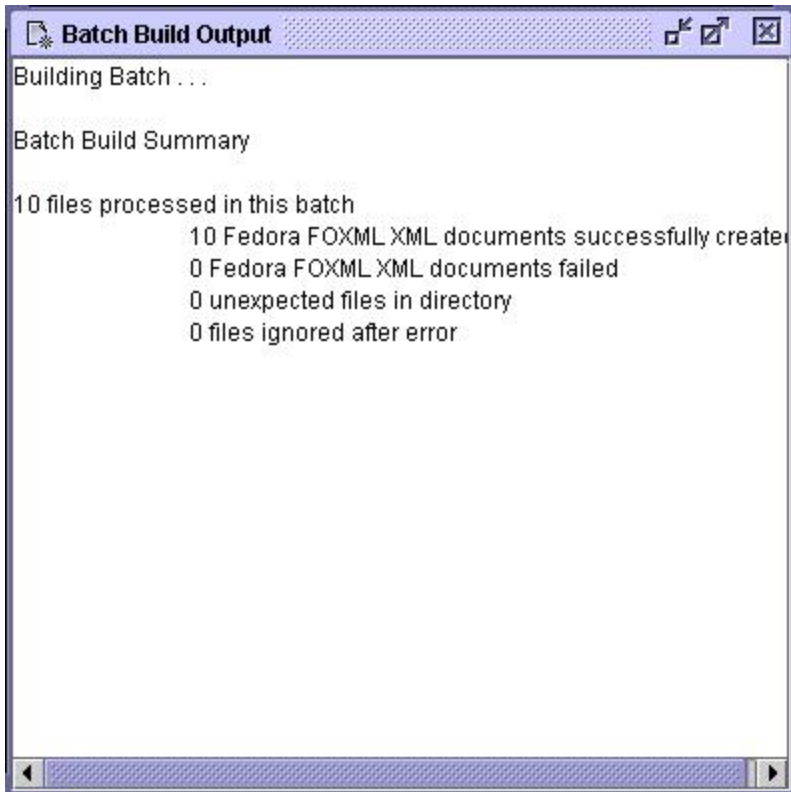
Then click the *Build this batch* button to build the batch of Fedora *METS* or Fedora *FOXML* XML documents.



A confirmation dialog will open requesting confirmation of the object template selected. Clicking *Yes* will continue the batch build. Clicking *No* or *Cancel* will return the user to the *Batch Build* window.



A second (output-only) window will open to show progress. You can build multiple different batches before closing the *Batch Build* window.



You can then ingest the created batch as described elsewhere in this document. No subdirectories or files are deleted by Fedora Administrator. Setup and cleanup of the files in the batch must be done by you using standard operating systems facilities.

To Demo

You can use files and subdirectories of directory **client/demo/batch-demo**, relative to your **FEDORA_HOME** environment variable. (When you create your own batches, the needed directories and files can be anywhere in the file space of the system on which you are running Fedora Administrator or command-line *BatchTool*.)

- Use file **metes-template.xml** for *METS template (input file)* if you want to build *METS* objects.
- Use file **foxml-template.xml** for *FOXML template (input file)* if you want to build *FOXML* objects. Use subdirectory **object-specifics** for *XML specs (input directory)*; this is a directory holding (all and only) per-object data.
- Use subdirectory **objects** for the built objects (*output directory*); this is a directory to hold (all and only) Fedora object files built by Fedora Administrator. The format of the built files will be determined by the type of template used (*METS* or *FOXML*).

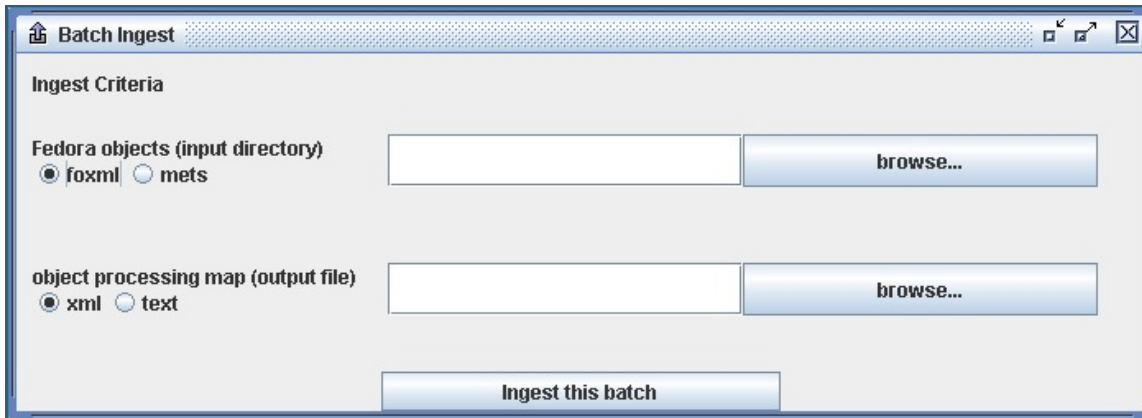
Specify a file path of your choice for *object processing map (output file)*; this is a file which maps *object-specs* to objects built. See the section on *object processing maps*, elsewhere in this documentation. Note that PIDs cannot be reported in this (*Batch Build*) mode, as they have not yet been assigned. Optionally select the output format for *object processing map*, either **xml** or **text** (**xml** is the default format).

Ingesting Fedora Objects in Batch

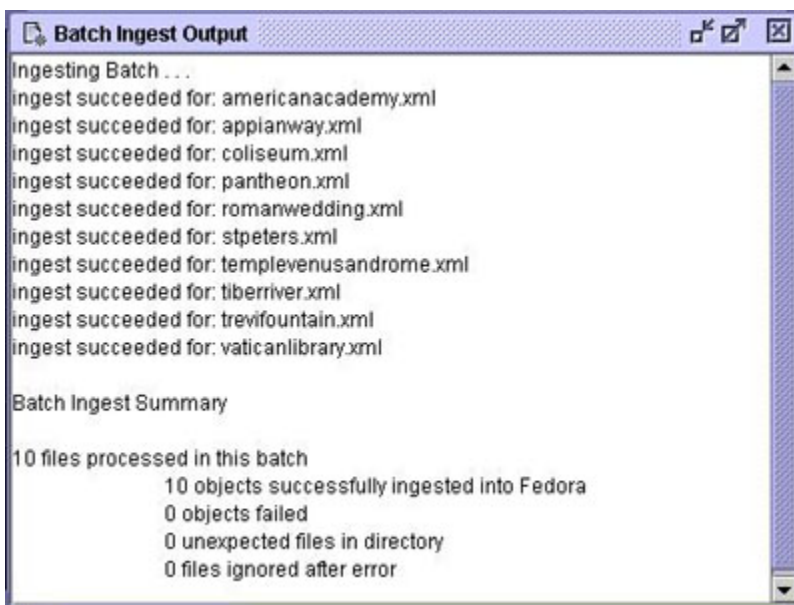
Create a set of Fedora objects in your repository from a corresponding set of Fedora *METS* XML or Fedora *FOXML* XML files.

Fedora Administrator Instructions

Select *Tools* on the Fedora Administrator menu bar, and select item *Ingest Batch*. This will open a *Batch Ingest* window. You may need to adjust this window's size to see its controls. Use the *browse* buttons to enter the two required settings. Clicking on a *browse* button opens a standard directory/file selection dialog.



You must also indicate the format of the objects to be ingested by selecting either the foxml or mets radio button. Then click the *Ingest this batch* button to ingest the batch into your Fedora repository. A second (output-only) window will open to show progress. You can ingest multiple different batches before closing the *Batch Ingest* window.



No subdirectories or files are deleted by Fedora Administrator. Setup and cleanup is by using standard operating systems facilities. Fedora Administrator does not itself validate on *Batch Build*, but batch ingest into Fedora does. The batch fails on the first individual object ingest failure. Fedora will not ingest a *METS* file whose **METS:xmldata** elements are empty or contain non-tagged character data.

To Demo

You can use files and subdirectories of directory **client/demo/batch-demo**, relative to your **FEDORA_HOME** environment variable. (When you create your own batches, the needed directories and files can be anywhere in the file space of the system on which you are running Fedora Administrator or command-line *BatchTool*.) You will need to have already done a *Build Batch* demo, explained elsewhere in this document, to populate the **objects** directory needed in this current demo. If you have ingested these objects before, either in this *Ingest Batch* mode following a separate *Build Batch* mode, or in a *Build and Ingest Batch* mode, you will first need to edit **OBJIDs** in the *object-spec* files, or to remove the corresponding objects from your Fedora repository.

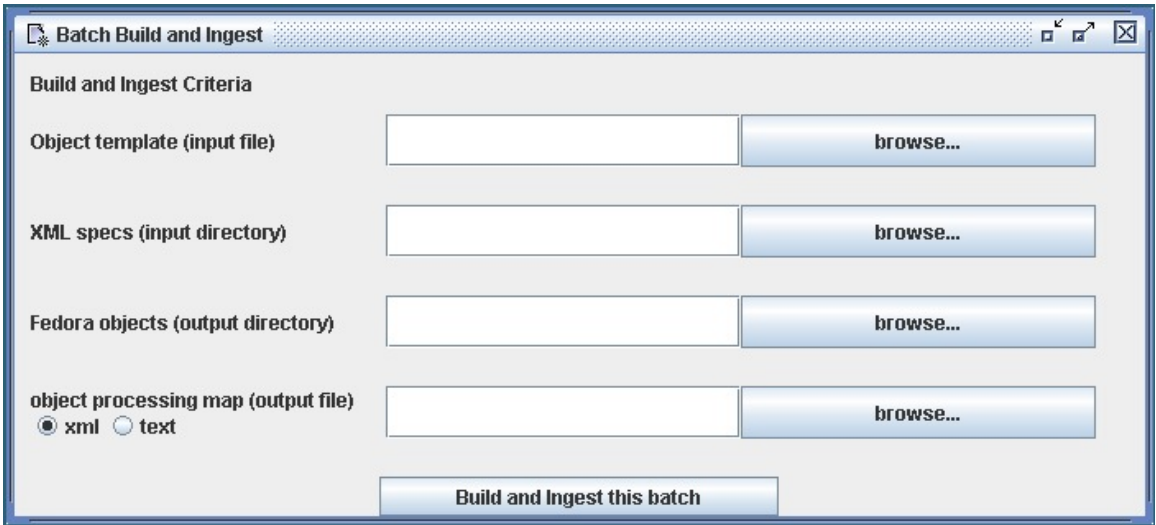
Use subdirectory **objects** for *built objects (input directory)*; this is a directory holding (all and only) Fedora object files to ingest. Specify a file path of your choice for *object processing map (output file)*; this is a file which maps objects to their assigned PIDs. See the section on *object processing maps*, elsewhere in this documentation. Note that *object-specs* of objects previously built by Fedora Administrator cannot be reported in this (*Batch Ingest*) mode, as they (as source documents) are no longer known. Optionally select the output format for object processing map, either **xml** or **text** (**xml** is the default format).

Building and Ingesting Fedora objects in Batch

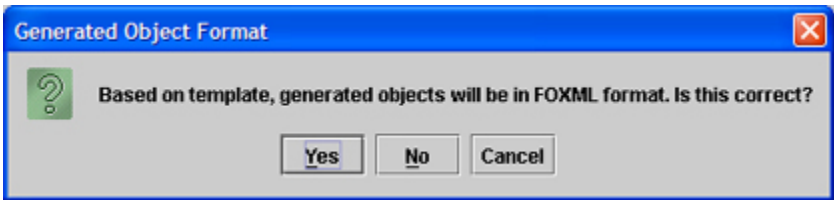
This process builds a set of Fedora *METS* XML or Fedora *FOXML* XML files from a common Fedora *METS* or Fedora *FOXML* template and simple *batchMerge* XML *object-specs*, then ingests the resulting batch into Fedora.

Fedora Administrator Instructions

Select *Tools* on the Fedora Administrator menu bar, and select item *Build and Ingest Batch*. This will open a *Batch Build and Ingest* window. You may need to adjust this window's size to see its controls. Use the *browse* buttons to enter the four required settings. Clicking on a *browse* button opens a standard directory/file selection dialog.



Then click the *Build and Ingest this batch* button to build the batch of Fedora METS XML documents and then ingest them into Fedora. The format of the built objects is determined by the format of the template. A confirmation dialog will open requesting confirmation of the object template selected. Clicking *Yes* will continue the batch build. Clicking *No* or *Cancel* will return the user to the *Batch Build and Ingest* window.



A second (*output-only*) window will open to show progress. You can build and ingest multiple different batches before closing the *Batch Build and Ingest* window.



There is then no need to separately ingest the created batch; no directories or files are deleted by Fedora Administrator. Setup and cleanup of the files in the batch must be done by you using standard operating systems facilities. Fedora Administrator does not itself validate on *Batch Build*, but batch ingest into Fedora does. The batch fails on the first individual object ingest failure. Fedora will not ingest a *METS* file whose **METS:xmldata** elements are empty or contain non-tagged character data.

To Demo

You can use files and subdirectories of directory **client/demo/batch-demo**, relative to your **FEDORA_HOME** environment variable. (When you create your own batches, the needed directories and files can be anywhere in the file space of the system on which you are running Fedora Administrator or command-line *BatchTool*.) If you have ingested these objects before, either in this *Build and Ingest Batch* mode or in separate sequential *Build Batch* and *Ingest Batch* modes, you will first need to edit **OBJIDs** in the *object-spec* files, or to remove the corresponding objects from your Fedora repository. Use file **metstemplate.xml** for *METS template (input file)* if you want to create Fedora *METS* objects. Use file **foxml-template.xml** for *FOXML template (input file)* if you want to create Fedora *FOXML* objects. Use subdirectory **object-specifics** for *XML specs (input directory)*; this is a directory holding (all and only) per-object data. Use subdirectory **objects** for *built objects (output directory)*; this is a directory to hold (all and only) Fedora object files built by Fedora Administrator.

Specify a file path of your choice for *object processing map (output file)*; this is a file which maps *object-specs* through objects built and on to PIDs assigned. See the section on *object processing maps*, elsewhere in this documentation. Unlike separate *Batch Build* and *Batch Ingest* modes, the complete triple is reported in this *Batch Build and Ingest* mode. Optionally select the output format for *object processing map*, either **xml** or **text** (**xml** is the default format).

Object Processing Map

The *object-processing-map* file has one of the following formats, depending on the choice of *xml* or *text* in Fedora Administrator. *Batch Build* processing results in an *object processing map* whose individual maps have only **path2spec** and **path2object** attributes or fields. *Batch ingest* processing results in an *object processing map* whose individual maps have only **path2object** and **pid** attributes or fields. *Batch build and Ingest* processing results in an *object processing map* whose individual maps have all three **path2spec**, **path2object** and **pid** attributes or fields.

XML Format

```

<object-processing-map>
  <map
    path2spec="/mellon/dist/client/demo/batch-demo/object-specifics/americanacademy.xml"
    path2object="/mellon/dist/client/demo/batch-demo/objects/americanacademy.xml"
    pid="demo:3010" />
    . . .
  <map
    path2spec="/mellon/dist/client/demo/batch-demo/object-specifics/vaticanlibrary.xml"
    path2object="/mellon/dist/client/demo/batch-demo/objects/vaticanlibrary.xml"
    pid="demo:3019" />
</object-processing-map>

```

Text Format

The field separator is tab; relative paths are used for practical illustration.

```

object-specifics/americanacademy.xml objects/americanacademy.xml demo:3010
. . .
object-specifics/vaticanlibrary.xml objects/vaticanlibrary.xml demo:3019

```

Object-Specifics

Object-specifics are coded in XML files. These data include: object ID, label, and comment; datastream and object metadata and accompanying label; datastream URLs, titles, and labels. Prior to Fedora 2.1, there was no formal *batchMerge* schema for object-specific files and the schema was implied based on the example object-specific files. The new *batchMerge.xsd* schema provides a more formal definition that conforms to the pre-Fedora 2.1 object-specific examples and also provides some new extensions for attributes introduced with *FOXML*. The new schema can be found at: <http://www.fedora.info/definitions/1/0/api/batchMerge.xsd>.

Where possible, attribute names on elements are the same as in the Fedora *METS* or *FOXML* schema, and so correspond to like-named attributes in the Fedora *METS* or *FOXML* template. How these map is described below and by running the demo and viewing the results for one of the objects. Any individual substitution is optional. When absent as a substitution, the value in the template will be used for the resulting Fedora *METS* or *FOXML* object. (Demo template and *object-specific* contents are chosen instructively to highlight substitutions made.) Datastream URLs will generally be specific to an object; practice will show which other substitutions are generally made. All *non-METS* and *non-FOXML* namespaces used in your own metadata must be declared, as in `xmlns:uvalibadmin` in the demo. The *metadata* element is for backward compatibility with pre-Fedora 2.1 *object-specific* file formats where inline XML metadata was treated separately from other types of datastreams. It is recommended that users use the new extended *datastream* element for all types of datastreams. The *metadata* element may be deprecated in a future release of the *batchMerge* schema.

When working with a *METS* template object:

Datastream IDs here map to those found in the Fedora **METS:fileGrp** element (the nested, not the nesting, one). The associated **xlink:href** and **xlink:title** attributes are substituted into the Fedora **METS:Flocat** element, which is nested within that Fedora **METS:fileGrp** element. Datastream labels substitute instead into **METS:structMap**.

When working with a *FOXML* template object:

Datastream IDs here map to the **ID** attribute found in the Fedora **foxml:datastream *element**. The associated ***xlink:href** attribute maps to the **REF** attribute found in the Fedora **foxml:contentLocation** element. The **xlink:title** attribute is for backward compatibility. Both datastream **LABEL** and **xlink:title** attributes map to the **LABEL** attribute found in the Fedora **foxml:datastreamVersion** element. It is recommended that you use the **LABEL** attribute instead of the **xlink:title** attribute when referring to the label of a datastream.

Case matters in attribute and element names. Fedora will retain as PIDs only **OBJIDs** whose prefixes are included in the `fedora.fcfg` file `retainPids` parameter (e.g., `test`, `demo`, etc.). Other **OBJIDs** will be replaced by Fedora-generated PIDs. *object-specs* in a given batch should meet the structural requirements of that batch's template: same number and tagging of datastreams, same number and tagging of metadata elements. Since substitutions are optional, individual *object-specs* cannot have "missing" data: the resulting object simply retains the template's value. Neither can *object-specs* have "extra" data: the resulting object simply lacks the *object-spec's* data - because the template isn't designed to use it. In either case, the batch goes on.

The following *object-spec* fragment from *americanacademy.xml* illustrates some of this.

```

<?xml version="1.0" encoding="utf-8"?>
<input xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xlink="http://www.w3.org/TR/xlink"
      xsi:schemaLocation="http://www.fedora.info/definitions/
http://www.fedora.info/definitions/1/0/api/batchMerge.xsd"
      xmlns="http://www.fedora.info/definitions/"
      OBJID="demo:3010" LABEL="American Academy">
  <datastreams>
    . . .
    <datastream ID="RIGHTS1">
      <!-- *** TESTING: SUBSTITUTING METADATA FOR RIGHTS1 *** -->
      <xmlContent>
        <uvalibadmin:admin xmlns:uvalibadmin="http://dl.lib.virginia.edu/bin/dtd/admin/admin.dtd">
          <uvalibadmin:adminrights>
            <uvalibadmin:policy>
              <uvalibadmin:access>unrestricted</uvalibadmin:access>
              <uvalibadmin:use>educational</uvalibadmin:use>
            </uvalibadmin:policy>
          </uvalibadmin:adminrights>
        </uvalibadmin:admin>
      </xmlContent>
    </datastream>
    . . .
    other metadata datastreams
    . . .
    <datastream ID="DS1"
      xlink:href="http://www.fedora.info/demo/batch-demo/thumb/americanacademy.jpg"
      LABEL="*** TESTING: SUBSTITUTING LABEL FOR DS1 ***"/>
    <datastream ID="DS2"
      xlink:href="http://www.fedora.info/demo/batch-demo/medium/americanacademy.jpg"
      xlink:title="*** TESTING: SUBSTITUTING XLINK:TITLE FOR DS2 ***"/>
    <datastream ID="DS3"
      xlink:href="http://www.fedora.info/demo/batch-demo/high/americanacademy.jpg"/>
    <datastream ID="DS4"
      xlink:href="http://www.fedora.info/demo/batch-demo/very-high/americanacademy.jpg"/>
  </datastreams>
</input>

```

Progress Report File

Fedora Administrator already provides a progress report of each use of a batch tool, written to a GUI window, to provide user feedback. Additionally, this progress report is now written to a text file, to provide a permanent record. Note that this progress report is not especially suited and is not intended for further processing by another computer program. Use the "object processing map", a different output file already provided, for such machine processing. The progress report file is written to the same directory as the object processing map. The name of any instance of these new files includes the time when it is written, e.g., 20031203-123201-365.txt. The final group of numerals would serve to differentiate report files written, oddly but possibly, at the same second, by 2 instances of the GUI client running on the same machine.

The following description tells what is recorded in this new file and how its directory is chosen in giving the location of the object processing map. The batch tools are available through the Tools menu, under Batch, and serve to provide: Build Batch, Build and Ingest Batch, or Ingest Batch. After selecting one of these tools, a dialog box opens for user input of tool parameters. This dialog box is titled "Batch Build", "Batch Build and Ingest", or "Batch Ingest", depending on which tool is chosen. For each tool, one of the required parameters is the path to the "object processing map" (an output file), which records the tool's processing in a form amendable to later input to another program. This path is specified in a usual file dialog, including its parent directory. This is the parent directory, also, into which the new processing report file is written.

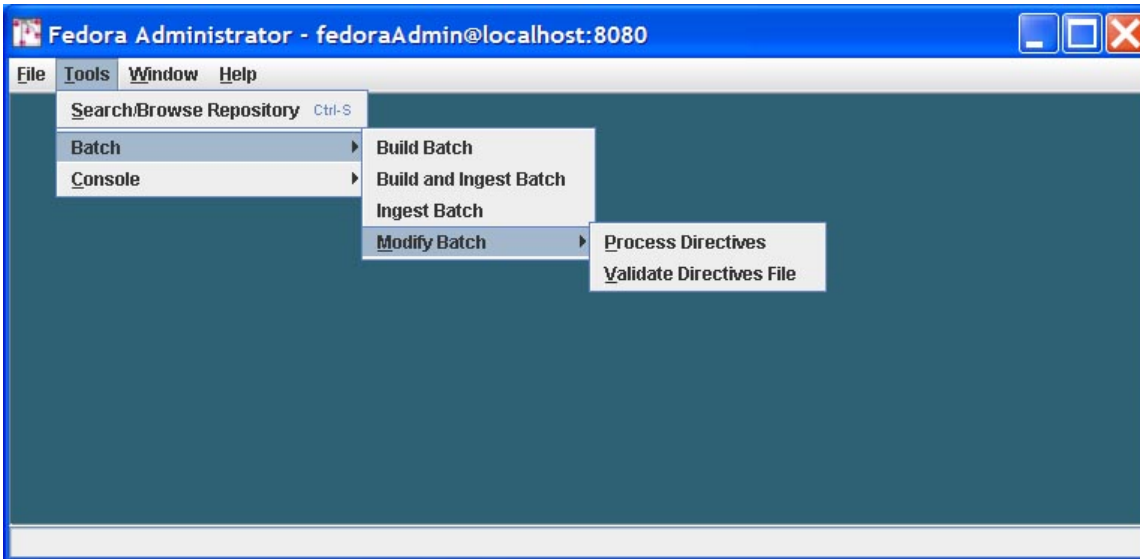
The contents of this new file is simply the contents of the respective output window of the GUI client, one of: "Batch Build Output", "Batch Build and Ingest Output", or "Batch Ingest Output".

Batch Modify

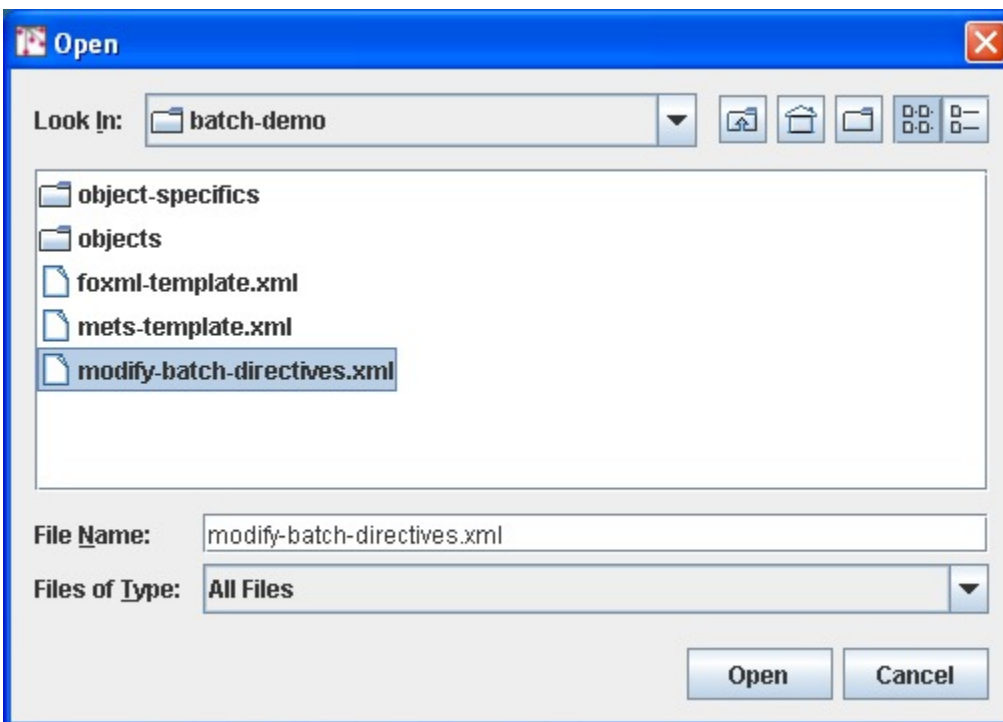
The Batch Modify Utility is an application that enables the modification of objects in batch mode. It is designed for use by repository administrators and is available under the Tools menu of the Administrator GUI client. It can also be invoked as a command-line utility using either the fedora-modify.bat (Windows) or fedora-modify.sh (Unix) scripts located in the distribution client/bin directory. The basic design of the utility is to process an xml input file containing modify directives and then process each directive in sequence using the methods of API-M. The format of the directives file is specified by an xml schema name batchModify.xsd. The schema is available in the local distribution in the tomcat ROOT webapp and is also available from the Fedora website at <http://www.fedora.info/definitions/1/0/api/batchModify.xsd>. Each modify directive mirrors the capability of the corresponding API-M directives.

Process Directives

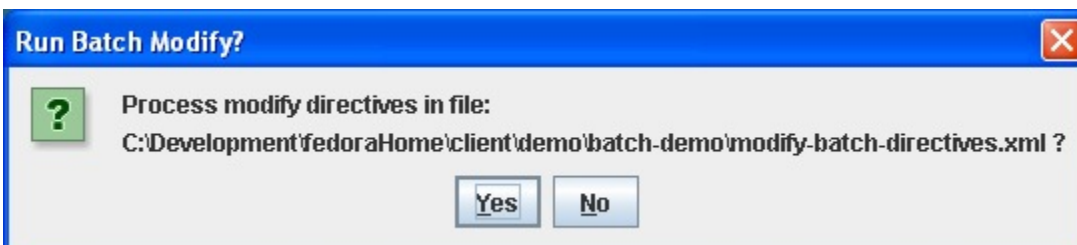
The Batch Modify Utility is accessed through the Tools menu of the Administrator GUI client under the submenu heading of Batch/Modify Batch.



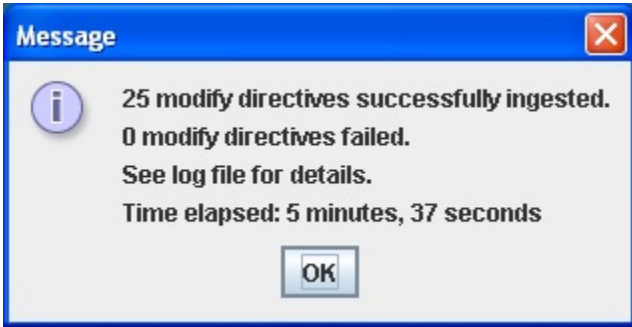
The Modify Batch menu contains two sub-items named Process Directives and Validate Directives File. The Process Directives menu item is used to begin processing a valid set of xml directives in a modify directives file. The Validate Directives File item is used to parse an existing batch modify directives file to insure that it conforms to the Fedora Batch Modify XML schema. Selecting the Process Directives item will prompt for the location of the directives file.



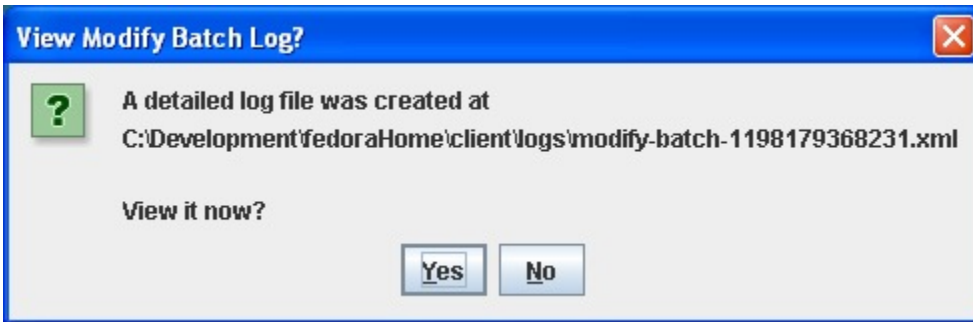
After selecting a directives file, you will be asked to confirm the choice.



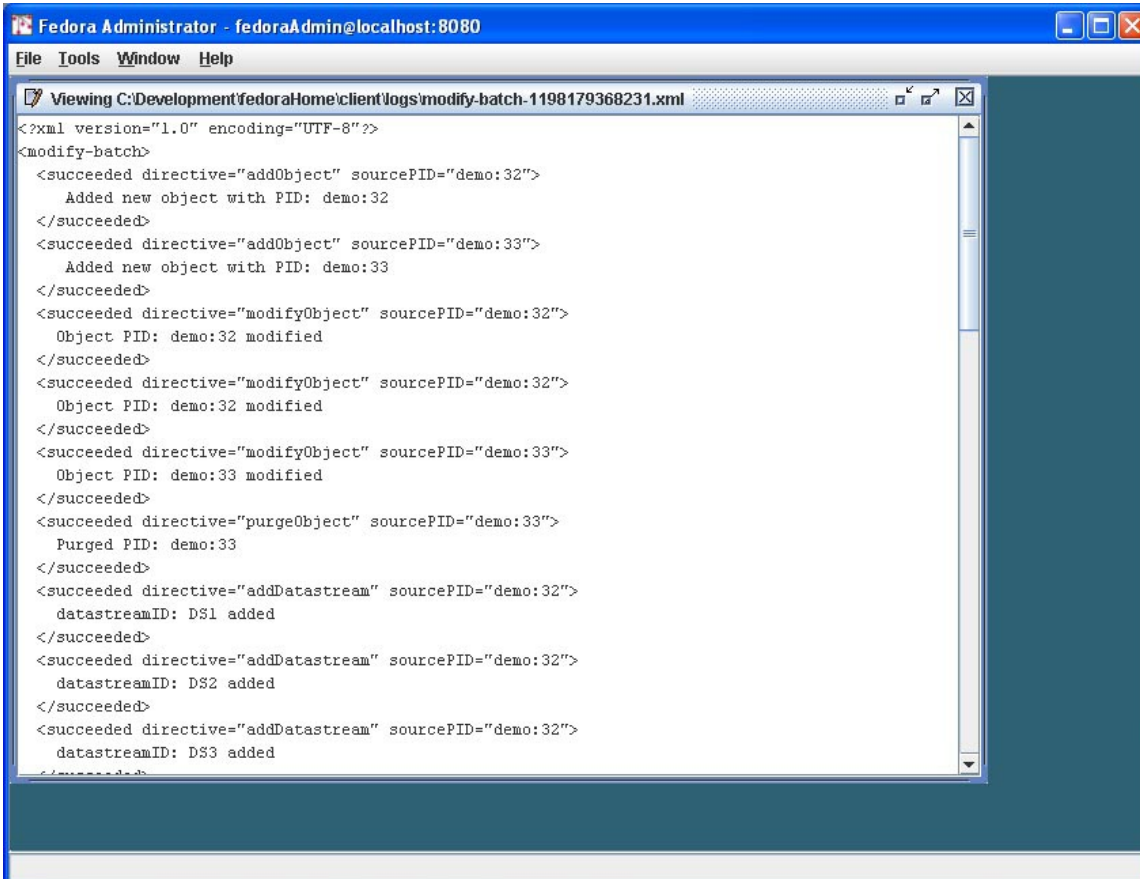
Upon clicking the yes button, the directives file will be processed and a log file generated that details the number of directives processed and any errors that may have occurred.



If there were any errors, you can peruse the log file to ascertain more information about what caused the directive to fail.

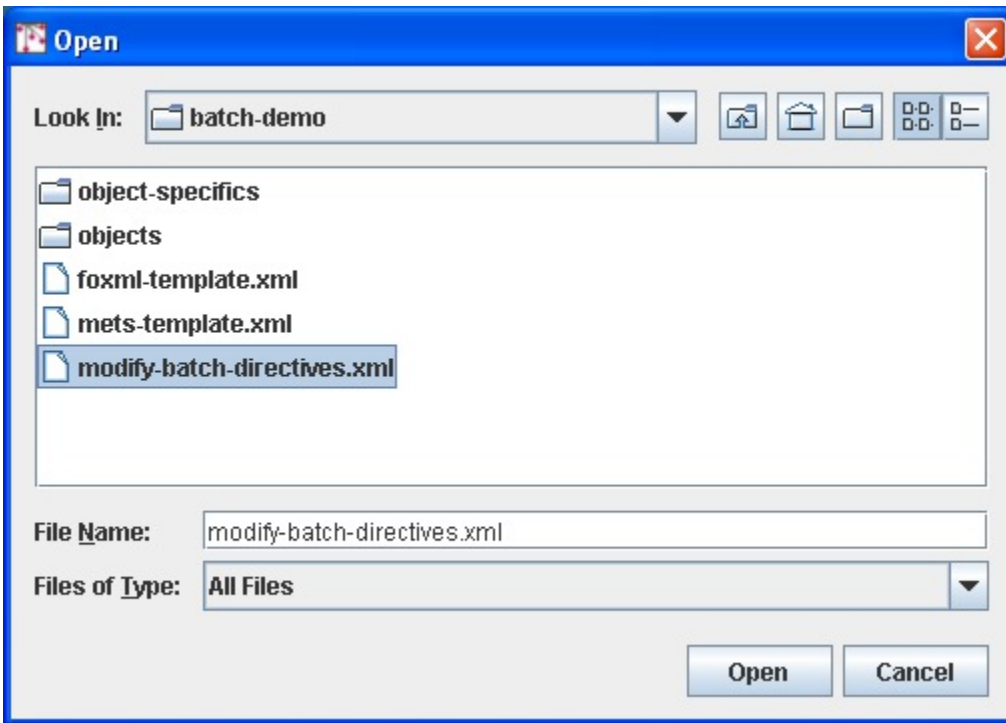


The log file consists of an xml file containing entries for each directive that was processed.

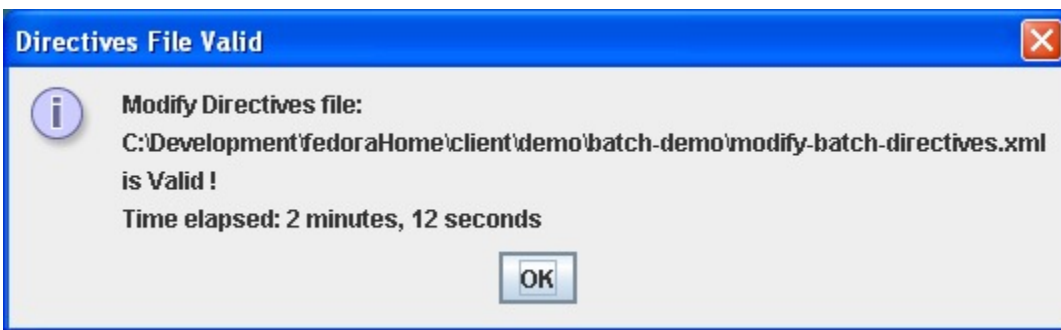
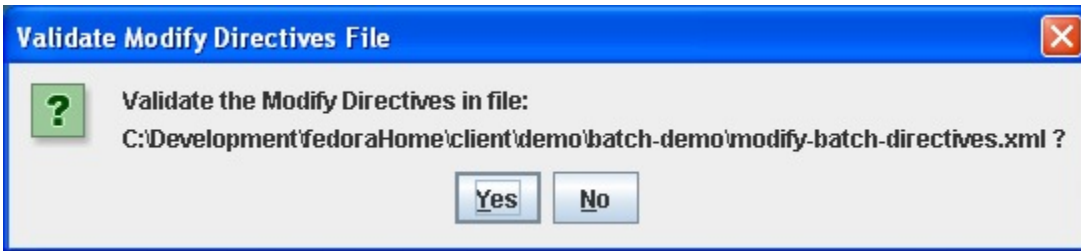


Validate Directives

The Validate Directives menu item is used to verify that an existing directives file is valid before attempting to process the file. This can be useful when the directive files were created using a non-xml editor that was unable to validate the file as it was created. The steps for validating a directive file are similar to that for processing. First you are prompted for the location of the directives file to be processed.



Confirming the file location initiates the parsing process and the results are displayed when parsing is completed.



To Demo

There is a sample batch modify directives file included in the distribution located in the dist/client/demo/batch-demo directory named modify-batch-directives.xml. This sample file creates a new object and then performs various additions, deletions, and modifications to the object's components through a series of directives. For reference the sample directives file is included below.

Sample Modify Directives File

```
<?xml version="1.0" encoding="utf-8"?>
<!-- ***** -->
```

```

<!-- This is a sample modify directives file that performs a variety of -->
<!-- modifications on the datastreams of two new demo -->
<!-- objects (demo:32 and demo:33) that the script will create. In general, -->
<!-- one should always validate a modify directives file against the -->
<!-- modifyBatch.xsd schema prior to processing to catch any syntax errors. -->
<!-- Processing will halt if the directives file is invalid and a log file -->
<!-- is generated summarizing the results of the batch. Refer to the xml -->
<!-- schema file (modifyBatch.xsd) for details on the syntax of the modify -->
<!-- file. -->
<!-- -->
<!-- ***** -->
< fbm:batchModify xmlns:fbm="http://www.fedora.info/definitions/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.fedora.info/definitions/
    http://www.fedora.info/definitions/1/0/api/batchModify.xsd">
<!-- ***** -->
<!-- Add a new empty object to the repository that has no datastreams -->
<!-- with PID of demo:32. -->
<!-- -->
<!-- NOTE: By default each Fedora repository specifies several PID -->
<!-- namespaces (e.g. "demo")that will be retained at ingest by the -->
<!-- repository allowing ingested objects to retain the PID specified -->
<!-- at ingest time. This list of PID namespaces is configurable in -->
<!-- fedora.fcfg using the retainPids config parameter. In this example, -->
<!-- demo:32 is specified PID so the generated object will retain this -->
<!-- PID value after ingestion. Leaving the pid attribute empty -->
<!-- (i.e., "") will force the repository to assign a PID to the new -->
<!-- object using the namespace specified by the PidNamespace parameter -->
<!-- in the fedora.fcf config file. -->

<!-- -->
<!-- Required attributes on directive: -->
<!-- pid - PID of the object; leave blank if you want repository to -->
<!-- assign the pid. -->
<!-- label - label for the object -->
<!-- contentModel - content model of the object -->
<!-- logMessage - message to be written in audit trail record -->
<!-- ***** -->
<fbm:addObject pid="demo:32" label="Sample Object Used With Batch Modify Utility"
  contentModel="Object" logMessage="BatchModify - addObject"/>
<!-- ***** -->
<!-- Add a second new empty object to the repository that has no -->
<!-- datastreams with a PID of demo:33. -->
<!-- ***** -->
<fbm:addObject pid="demo:33" label="2nd Sample Object Used With Batch Modify Utility"
  contentModel="Object" logMessage="BatchModify - addObject"/>
<!-- ***** -->
<!-- Modify object demo:32 by changing the label on the object. -->
<!-- -->
<!-- Required attributes on directive: -->
<!-- pid - PID of the object -->
<!-- logMessage - message to be written in audit trail record -->
<!-- -->
<!-- Optional attributes on directive: -->
<!-- label - label for the object -->
<!-- state - state of the object -->
<!-- ***** -->
<fbm:modifyObject pid="demo:32" label="Object Label was changed" logMessage="BatchModify -
modifyObject"/>
<!-- ***** -->
<!-- Modify object demo:33 by changing its state to deleted. -->
<!-- ***** -->
<fbm:modifyObject pid="demo:33" state="D" logMessage="BatchModify - modifyObject "/>
<!-- ***** -->
<!-- Purge object demo:33 that was just created and modified. -->
<!-- -->
<!-- Required attributes on directive: -->
<!-- pid - PID of the object to be purged -->
<!-- logMessage - message to be written in audit trail record -->
<!-- -->

```

```

<!-- Optional attributes on directive: -->
<!-- force - boolean indicating whether to purge the object if any -->
<!-- critical dependencies exist. A value of true will purge the object -->
<!-- regardless of any dependencies. A value of false will allow the -->
<!-- purge only if no dependencies exist. This parameter is currently -->
<!-- not implemented in the server. -->
<!-- ***** -->
<fbm:purgeObject pid="demo:33" logMessage="BatchModify - purgeObject"/>
<!-- ***** -->
<!-- Add a new datastream to demo:32 object with ID of DS1. -->
<!-- -->
<!-- Note: A dsID of DS1 is assigned by the server since no dsID is -->
<!-- specified on the directive and this is the first datastream in -->
<!-- this object. -->
<!-- -->
<!-- -->
<!-- Required attributes on directive: -->
<!-- pid - PID of the object -->
<!-- dsLabel - label fo the datastream -->
<!-- dsMIME - MIME type of the datastream -->
<!-- logMessage - message to be written in audit trail record -->
<!-- dsState - state of the object -->
<!-- dsControlGroupType - control group type of the datastream -->
<!-- dsLocation - location of the datastream; for XMLMetadataDatastreams -->
<!-- dsLocation omit dsLocation or leave as blank. For all -->
<!-- other datastream types, it is required and must -->
<!-- denote the remote location of the datastream. -->
<!-- -->
<!-- Optional attributes on directive: -->
<!-- dsID - ID of the datastream; omit or leave as blank if you want -->
<!-- repository to assign datastream ID. -->
<!-- formatURI - URI identifying the format of the datastream -->
<!-- versionable - boolean indicating whether datastream is versionable -->
<!-- altIDs - space delimited string of alternated identifiers for the -->
<!-- datastream. This info is currently maintained in the -->
<!-- object, but not acted on. -->
<!-- ***** -->
<fbm:addDatastream pid="demo:32" dsLabel="Thorny's Coliseum thumbnail jpg image" dsMIME="image/jpeg"
    dsLocation="http://localhost:8080/demo/simple-image-demo/coliseum-thumb.jpg"
    dsControlGroupType="E" dsState="A" logMessage="BatchModify - addDatastream"/>
<!-- ***** -->
<!-- Add a 2nd datastream to demo:32 object. The new datastream will be -->
<!-- a medium resolution image and will be assigned a dsID of DS2 since -->
<!-- no dsID is specified and this is the second datastream for this -->
<!-- object. -->
<!-- ***** -->
<fbm:addDatastream pid="demo:32" dsLabel="Thorny's Coliseum medium jpg image" dsMIME="image/jpeg"
    dsLocation="http://localhost:8080/demo/simple-image-demo/coliseum-medium.jpg"
    dsControlGroupType="E" dsState="A" logMessage="BatchModify - addDatastream "/>
<!-- ***** -->
<!-- Add a 3rd datastream to demo:32 object. The new datastream will be -->
<!-- a high resolution image and will be assigned a dsID of DS3 since -->
<!-- no dsID is specified and this is the third datastream for this -->
<!-- object. -->
<!-- ***** -->
<fbm:addDatastream pid="demo:32" dsLabel="Thorny's Coliseum high jpg image" dsMIME="image/jpeg"
    dsLocation ="http://localhost:8080/demo/simple-image-demo/coliseum-high.jpg"
    dsControlGroupType="M" dsState="A" logMessage="BatchModify - addDatastream"/>
<!-- ***** -->
<!-- Add a 4th datastream to demo:32 object. The new datastream will be -->
<!-- a very high resolution image and will be assigned a dsID of DS4 -->
<!-- since no dsID is specified and this is the fourth datastream for -->
<!-- this object. -->
<!-- ***** -->
<fbm:addDatastream pid="demo:32" dsLabel="Thorny's Coliseum veryhigh jpg image" dsMIME="image/jpeg"
    dsLocation="http://localhost:8080/demo/simple-image-demo/coliseum-veryhigh.jpg"
    dsControlGroupType="M" dsState="A" logMessage="BatchModify - addDatastream"/>
<!-- ***** -->
<!-- Add a 5th datastream to demo:32 object. The new datastream will be -->
<!-- a screen size image and will have dsID of SCREEN. This datastream -->

```

```

<!-- will also have values assigned for alternate IDs, formatURI, and -->
<!-- is declared not to be versionable. -->
<!-- ***** -->
<fbm:addDatastream pid="demo:32" dsID="SCREEN" altIDs="AlternateID1 AlternateID2"
    formatURI="info:fedora/demo/content/JPEG#" versionable="false"
    dsLabel="Thorny's Coliseum screen size jpg image" dsMIME="image/jpeg"
    dsLocation="http://localhost:8080/demo/simple-image-demo/coliseum-high.jpg"
    dsControlGroupType="E" dsState="A" logMessage="BatchModify - addDatastream"/>
<!-- ***** -->
<!-- Add a 6th datastream to demo:32 object. The new datastream will be -->
<!-- a user-defined inline descriptive metadata datastream and will -->
<!-- have dsID of DESC. -->
<!-- ***** -->
<fbm:addDatastream pid="demo:32" dsID="DESC" dsLabel="Descriptive metadata for Thorny's Coliseum
screen size jpg image"
    dsMIME="text/xml" dsControlGroupType="X" dsState="A" logMessage="BatchModify -
addDatastream">
<fbm:xmlData>
<uvalibdesc:desc xmlns:uvalibdesc="http://dl.lib.virginia.edu/bin/dtd/descmeta/descmeta.dtd">
<uvalibdesc:time>
    <uvalibdesc:date type="created" certainty="ca." era="bc">1st century</uvalibdesc:date>
</uvalibdesc:time>
<uvalibdesc:identifier scheme="URN">uva-lib:2</uvalibdesc:identifier>
<uvalibdesc:rights type="use">unrestricted</uvalibdesc:rights>
<uvalibdesc:subject scheme="other" othertype="keyword">Roman Empire</uvalibdesc:subject>
<uvalibdesc:subject scheme="other" othertype="keyword">Roman</uvalibdesc:subject>
<uvalibdesc:subject scheme="other" othertype="keyword">Vespaciano</uvalibdesc:subject>
<uvalibdesc:subject scheme="other" othertype="keyword">Amphitheatrum Flavium</uvalibdesc:subject>
<uvalibdesc:title type="main">Coliseum -- Rome, Italy</uvalibdesc:title>
<uvalibdesc:form>architecture</uvalibdesc:form>
<uvalibdesc:mediatype type="image"><uvalibdesc:form>digital</uvalibdesc:form></uvalibdesc:mediatype>
<uvalibdesc:agent role="publisher">Alderman Library</uvalibdesc:agent>
<uvalibdesc:covspace>
    <uvalibdesc:geometry>
        <uvalibdesc:point>
            <uvalibdesc:lat>41.54N</uvalibdesc:lat><uvalibdesc:long>12.27E</uvalibdesc:long>
        </uvalibdesc:point>
    </uvalibdesc:geometry>
</uvalibdesc:covspace>
<uvalibdesc:covtime>
    <uvalibdesc:date era="bc">72</uvalibdesc:date><uvalibdesc:date era="bc">80</uvalibdesc:date>
</uvalibdesc:covtime>
<uvalibdesc:culture>Roman</uvalibdesc:culture>
<uvalibdesc:place type="original"><uvalibdesc:geogname>Italy,
Rome</uvalibdesc:geogname></uvalibdesc:place>
</uvalibdesc:desc>
</fbm:xmlData>
</fbm:addDatastream>
<!-- ***** -->
<!-- Add a 7th datastream to demo:32 object. The new datastream will be -->
<!-- duplicate of the screen size image that will be deleted later in -->
<!-- this set of directives. It will have a dsID of SCREENDUP. -->
<!-- ***** -->
<fbm:addDatastream pid="demo:32" dsID="SCREENDUP" dsLabel="Thorny's Coliseum screen size jpg image"
dsMIME="image/jpeg"
    dsLocation="http://localhost:8080/demo/simple-image-demo/coliseum-high.jpg"
    dsControlGroupType="R" dsState="A" logMessage="BatchModify - addDatastream"/>
<!-- ***** -->
<!-- Modify datastream DS1 of demo:5 object by changing its label. -->
<!-- -->
<!-- NOTE: Optional attributes that are omitted indicate that that -->
<!-- attribute is to remain uncahnged and the original value of of the -->
<!-- datastream attribute in the object will be preserved. Optional -->
<!-- attributes that are set to the empty string indicate that the -->
<!-- attribute value will be blanked out in the datastream. -->
<!-- In this example the datastream location and state attributes are -->
<!-- not modified. -->
<!-- -->
<!-- Required attributes on directive: -->
<!-- pid - PID of the object -->

```

```

<!-- dsID - ID of the datastream -->
<!-- dsControlGroupType - control group type of the datastream -->
<!-- logMessage - message to be written in audit trail record -->
<!-- -->
<!-- Optional attributes on directive: -->
<!-- dsLabel - label fo the datastream -->
<!-- dsState - state of the object -->
<!-- dsLocation - location of the datastream -->
<!-- dsMIME - MIME type of the datastream -->
<!-- formatURI - URI identifying the format of the datastream -->
<!-- versionable - boolean indicating whether datastream is versionable -->
<!-- altIDs - space delimited string of alternated identifiers for the -->
<!-- datastream. This info is currently maintained in the -->
<!-- object, but not acted on. -->
<!-- force - boolean indicating whether to purge the object if any -->
<!-- critical dependencies exist. A value of true will purge -->
<!-- the object regardless of any dependencies. A value of -->
<!-- false will allow the purge only if no dependencies exist. -->
<!-- ***** -->
<fbm:modifyDatastream pid="demo:32" dsID="DS1" dsControlGroupType="E"
    dsLabel="New label for datastream DS1"
    logMessage="BatchModify - modifyDatastream"/>
<!-- ***** -->
<!-- Modify datastream SCREEN of demo:32 object by changing its label -->
<!-- and datastream location so that it points to the same content as -->
<!-- that of datastream DS3 which is a high-res image. -->
<!-- -->
<!-- NOTE: Optional attributes that are omitted indicate that that -->
<!-- attribute is to remain uncahnged and the original value of of the -->
<!-- datastream attribute in the object will be preserved. Optional -->
<!-- attributes that are set to the empty string indicate that the -->
<!-- attribute value will be blanked out in the datastream. -->
<!-- ***** -->
<fbm:modifyDatastream pid="demo:32" dsID="SCREEN" dsControlGroupType="E"
    dsLabel="Changed label for datastream SCREEN"
    dsLocation="http://localhost:8080/fedora/get/demo:5/DS4"
    logMessage="BatchModify - modifyDatastream"/>
<!-- ***** -->
<!-- Modify datastream DC of demo:32 object by changing its label and -->
<!-- its xml content so that its content is replaced by the block of -->
<!-- xml below. -->
<!-- ***** -->
<fbm:modifyDatastream pid="demo:32" dsID="DC" dsControlGroupType="X"
    dsLabel="New label for DC datastream"
    logMessage="BatchModify - modifyDatastream">
<fbm:xmlData>
  <oai_dc:dc xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/"
    xmlns:dc="http://purl.org/dc/elements/1.1/"
    xmlns:my="http://foo.bar.none">
    <dc:title>Coliseum in Rome</dc:title>
    <dc:creator>Thornton Staples</dc:creator><dc:subject>Architecture, Roman</dc:subject>
    <dc:description>Image of Coliseum in Rome</dc:description>
    <dc:publisher>University of Virginia Library</dc:publisher>
    <dc:format>image/jpeg</dc:format>
    <dc:identifier>demo:5</dc:identifier>
    <my:this>some text</my:this>
    <my:that>some more text</my:that>
    <my:other>even more text</my:other>
  </oai_dc:dc>
</fbm:xmlData>
</fbm:modifyDatastream>
<!-- ***** -->
<!-- Modify datastream DESC of demo:32 object by changing its label. -->
<!-- Also change its formatURI and set versionable to false. -->
<!-- ***** -->
<fbm:modifyDatastream pid="demo:32" dsID="DESC" dsControlGroupType="X"
    dsLabel="New label for DC datastream" formatURI="info:fedora/new/formatURI"
    versionable="false" logMessage="BatchModify - modifyDatastream"/>
<!-- ***** -->
<!-- Purge datastream SCREENDUP from object demo:32 -->

```

```
<!-- -->
<!-- Required attributes on directive: -->
<!-- pid - PID of the object to be purged -->
<!-- logMessage - message to be written in audit trail record -->
<!-- -->
<!-- Optional attributes on directive: -->
<!-- force - boolean indicating whether to purge the object if any -->
<!-- critical dependencies exist. A value of true will purge -->
<!-- the object regardless of any dependencies. A value of -->
<!-- false will allow the purge only if no dependencies exist. -->
<!-- purge only if no dependencies exist. This parameter is -->
<!-- currently not implemented in the server. -->
<!-- ***** -->
<fbm:purgeDatastream pid="demo:32" dsID="SCREENDUP" logMessage="BatchModify - purgeDatastream"/>
<!-- ***** -->
<!-- Change the state of datastream DS1 to deleted -->
<!-- -->
<!-- Required attributes on directive: -->
<!-- pid - PID of the object -->
<!-- dsID - ID of the datastream -->
<!-- dsState - new value for state of the datastream -->
<!-- logMessage - message to be written in audit trail record -->
<!-- ***** -->
<fbm:setDatastreamState pid="demo:32" dsID="DS1" dsState="D" logMessage="BatchModify -
setDatastreamState"/>
<!-- ***** -->
<!-- Change the state of datastream DESC to inactive -->
<!-- ***** -->
```

```
<fbm:setDatastreamState pid="demo:32" dsID="DESC" dsState="I" logMessage="BatchModify -
setDatastreamState"/>
</fbm:batchModify>
```

Command Line Alternatives

Rather than using the Fedora Administrator UI, you can build, ingest, and modify batches of Fedora object with the command-line utilities included with Fedora.

Please see relevant usage instructions at [Client Command Line Utilities](#) for the following scripts:

- fedora-batch-build
- fedora-batch-ingest
- fedora-batch-buildingest
- fedora-modify

Fedora Web Administrator

- [Introduction](#)
- [Starting the Fedora Web Administrator](#)
- [Object Menu](#)
- [Search Menu](#)
- [Working with Objects](#)
- [Fedora Security](#)
- [Flash Security](#)

Introduction

The Fedora Web Administrator provides a web-based user interface to perform repository administration. The application, which was built using Adobe Flex, uses a Flash plugin to run within a web browser. All coordination with the Fedora repository is performed using the Fedora REST API. The Fedora Web Administrator can be used to discover, display, ingest, modify, and purge objects and datastreams within a Fedora Repository.



Fedora 3.2+ Required

The Fedora Web Administrator depends on the Fedora REST API, and specifically on modifications made to this API for Fedora 3.2. For this reason the Fedora Web Administrator will not function correctly if you attempt to use it with a pre-3.2 Fedora Repository.

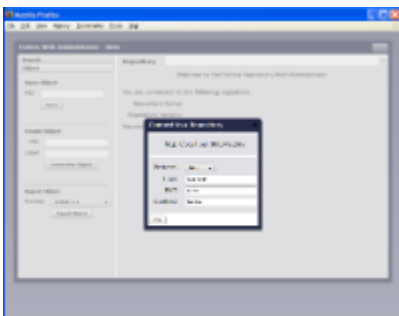


Beta Status

In Fedora 3.2, the Fedora Web Administrator is considered to be in Beta. If you find a bug, please report it on the [Fedora Repository tracker](#) or on the Fedora Users mailing list. Please be sure to indicate the browser being used when you encountered the issue.

Starting the Fedora Web Administrator

Ensure that the Fedora Repository (version 3.2+) has been started. Start a web browser (preferably Mozilla Firefox) and enter the URL <http://localhost:8080/fedora/admin>, replacing the protocol, host, port, and context to be appropriate to your installation.



The first task in using the Fedora Web Administrator, is connecting to a Fedora Repository. You will need to select the protocol (http or https) and enter the host, port, and context of your running Fedora. (If you plan to connect to Fedora using a protocol, host, or port which is different from the

protocol, host, and port on which the Web Administrator was loaded, see the [Flash Security](#) section below.)



After you enter your connection information and click Ok, the Web Administrator will connect to the Fedora Repository. Note that you can select the button on the top right corner (which displays the connection options you selected) at any time to connect to a different repository.

The interface of the Web Administrator is divided into two parts. The section on the left provides a slider with Search and Object menus. These menus provide tools that help you either retrieve a digital object which is already in the repository, or add a new object to the repository. The section on the right of the Web Administrator interface is where tabs are made available with search results and object information.

Object Menu

The Object Menu provides three options for working with digital objects. If you know the PID of the object you want to view, simply type that PID into the space provided under the Open Object heading and select Open. This will open the object in a new tab, or if the object has already been opened, it will be selected.



If you want to create a new object, you can do so using the Create Object option. Under the Create Object heading, enter the PID and/or Label for your new object and click Create New Object. This will create a new empty object for you, which you can then update to include all of the necessary information.

If you already have an object defined in XML format, you can use the Ingest Object option. First select the format of the object you plan to ingest, then choose Ingest Object and a dialog will be provided for you to select your object file.



Pop-up

In most browsers (not IE) the file upload dialog is produced as a pop-up window, so if your browser blocks pop-ups, you may need to add an exception or turn off pop-up blocking temporarily in order to use this feature.

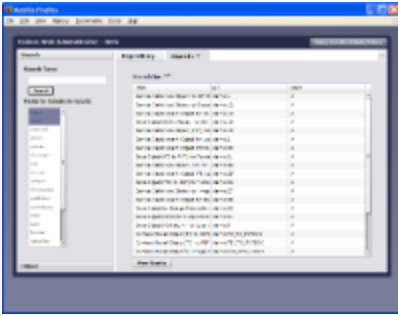


The internal format for all objects in Fedora is FOXML 1.1, so any object ingested in another format will be converted to FOXML 1.1 upon ingest. You can always export the object again using any of the available export formats, but the conversion process may produce XML which is slightly different from your original.

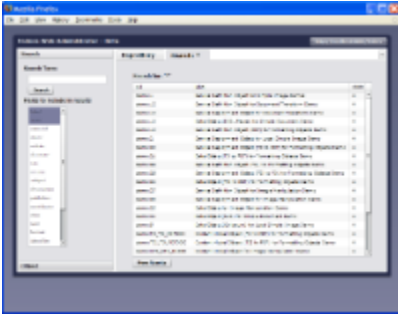
Search Menu

The Search Menu provides a way to perform simple searches over object properties and Dublin Core fields (contained in the DC datastream) in order to discover and locate objects in your repository.

To perform a search, select the Search heading to open the Search menu. Then enter your search term (or leave the box blank to search for all objects) and click Search. You also have the option of selecting the fields which you would like to see as part of the result listing.



The results of your search will be opened in a new tab with the results in a table. Selecting any of the table headers will sort the results (on that page). The columns of the table can be reordered and re-sized to make reading easier. To see the next page of results, click on the More Results button.

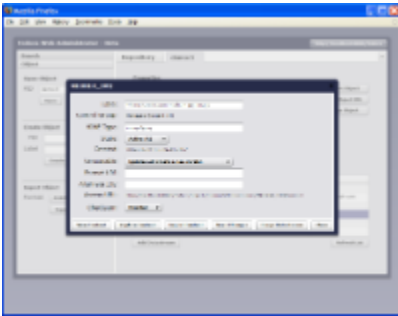


Selecting any of the rows in the results table will open the corresponding object.

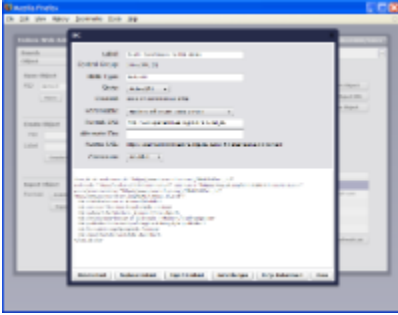
Working with Objects

When an object has been selected, through either the Object menu or search results, it will be opened in a new tab. If a tab for that object is already open, it will be selected. On the object tab you can view and change the properties of the object. You also have the option to export the object (in any of the various formats available), view the XML of an object (in FOXML 1.1 format), or purge the object from the repository. Also on the object tab is the list of datastreams stored as part of the object.

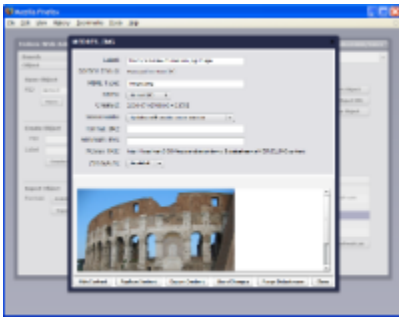
Selecting a datastream from the list will open a dialog to display the properties of the datastream.



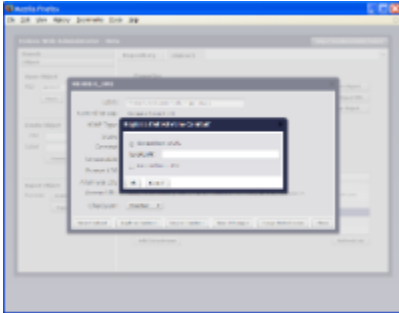
From the datastream dialog you have the option to update any of the properties. If the datastream is text or XML, you have the option to edit that content directly by selecting Edit Content.



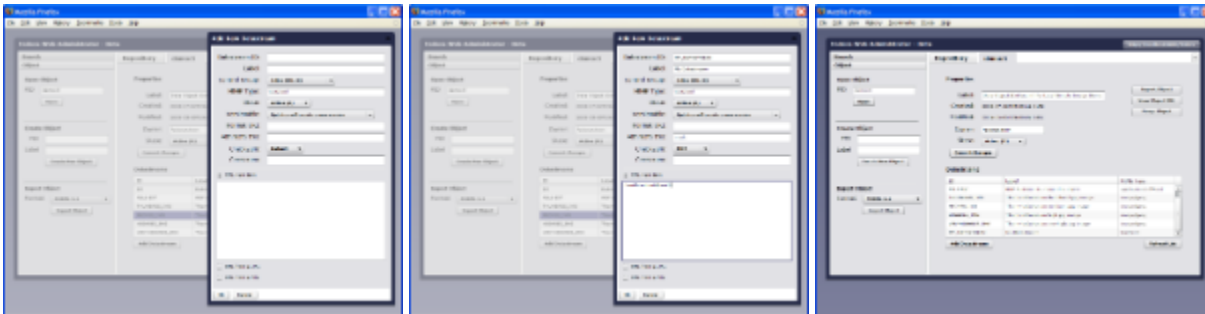
If the object is an image, you have the option to view the image by selecting View Content.



Select Export Content to download the datastream content. To replace the contents of the datastream, select Replace Content which, depending on the control group of the datastream, will give you different options for replacement.

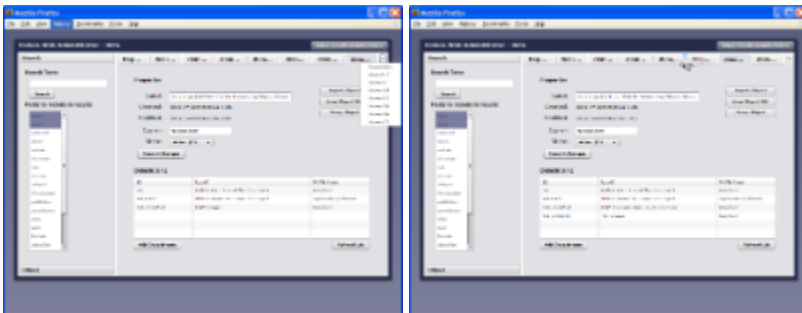


To add a new datastream, select Add Datastream on the object tab. This opens a new dialog which allows you to specify the details of the new datastream. Selecting the control group will change the available options for including content with the datastream.



After a datastream is added it will be included on the object's datastream list.

If the number of object tabs exceeds the space available for tabs, you can use the drop-down list on the top right to select tabs by name. You also have the option of closing tabs or reorganizing them by dragging them to a new position.



Fedora Security

The Web Administrator uses the Fedora REST API to perform all requests, so all actions pass through Fedora security, including user authorization and XACML policy enforcement, prior to being permitted.

Authentication for the Web Administrator is handled by the browser, so the dialog used to prompt for your username and password will differ from browser to browser. Note that since the browser is storing the authentication credentials you may be able to run multiple instances of the Web

Administrator across browser windows or tabs without having to re-authenticate. In order to "log out" you will need to end your browser session which, in most cases, requires either closing your browser or using a feature of the browser to clear authenticated sessions.

The timing of when you are prompted for credentials in the Web Administrator depends on how authentication is configured within your Fedora Repository. If API-A auth is enabled, you will be prompted on your first connection. However, if API-A auth is disabled, you will not be prompted to authenticate until you perform an API-M function. (Note that `getDatastream`, which is used to retrieve datastream information, is considered to be an API-M function.)

Flash Security

The Web Administrator runs inside of a Flash player in your browser and is thus subject to the security constraints imposed by this player. A few of these restrictions are listed here:

- HTTP connections made using SSL can only be made if the flash application was originally loaded via SSL. What this means is that if you want to connect to Fedora using the https protocol option, you need to connect to the Web Administrator using https. Simply put, before selecting the https protocol option in the connect dialog, make sure that the URL in the browser window already begins with https.
- Flash clients are not allowed to make HTTP requests to external services without explicit permission being given by those services. What this means is that if you are running a Web Administrator on a server which uses a different host or port than your Fedora Repository server, you will need to set up a permissions file, called `crossdomain.xml`, for your Fedora Repository. This file gives applications running in a Flash player the permission to make use of services provided by that server. This file will need to be available at the root of the domain, such as at <http://localhost:8080/crossdomain.xml>. An example `crossdomain.xml` file can be found on your Fedora server at <http://host:port/fedora/admin/crossdomain.xml>. While the example allows access to any Flash player, you have the option of placing limits on who will have access to your services. More information about `crossdomain.xml` files can be found on [Adobe's website](#).

Messaging Client

The Messaging Client is a Java client for connecting to and receiving messages from the Fedora Messaging System. Please see the [Messaging Client](#) section of the [Messaging documentation](#) for more information.

REST API Java Client

MediaShelf Java Client Library

We recommend the use of the [MediaShelf Java Client library](#) for Java applications using the REST API.

Fedora Local Services

[FOP Service](#)

[Image Manipulation Service](#)

[SAXON XSLT Processor Service](#)

FOP Service

Welcome to the FOP Service

This local service provides transformations from [XSL-FO](#) format to PDF. The FOP Service ^[1] functions as a Java servlet and can be invoked using the following syntax:

```
[hostname]:[port-number]/fop/FOPServlet?source=[xml-fo-source]
```

Where:

- `[hostname]` – The hostname of the Fedora server; default is localhost.
- `[port-number]` – The port number on which the Fedora server is running; default is 8080.
- `[xml-fo-source]` – The URL of the XML source file.

Example:

```
http://localhost:8080/fop/FOPServlet?source=http://www.fedora.info/services/fop/sample_fo.xml
```

[1] This service uses the FOP formatter from Apache which is licensed under the Apache Software License v1.1 (ASL). For additional information regarding FOP formatter, please refer to the FOP project web site at: <http://xml.apache.org/fop/>.

Image Manipulation Service

Welcome to the Image Manipulation Service

ImageManipulation [1] is a Java servlet that takes a URL of an image as a parameter and, based on other given parameters, can perform a variety of image related manipulations on the object. After the image is manipulated, it is then sent back as an image/type object to the calling parent, most often a browser or an HTML img tag.

```
[hostname]:[port-number]/imagemanip/ImageManipulation?url=[url-of-image][parameters]
```

Where:

- [hostname] – The hostname of the Fedora server.
- [port-number] – The port number on which the Fedora server is running.
- [url-of-image] – The URL of the image to open. Supported formats currently are gif, jpg, tiff, png, and bmp.
- [parameters] – Optional parameters that can be used to manipulate the image:
 - **Resize:** `op=resize&newWidth=[width-in-pixels]` – This resizes an image to a specified width in pixels, automatically scaling the height in proportion to the width.
 - **Zoom:** `op=zoom&zoomAmt=[zoom-value]` – This zooms an image, depending on the zoom-value. $0 < \text{zoom-value} < 1$: zoom out, $\text{zoom-value}=1$: original, $1 < \text{zoom-value}$: zoom in.
 - **Adjust Brightness:** `op=brightness&brightAmt=[bright-value]` – This adjusts the brightness of an image, depending on the bright-value. $0 < \text{bright-value} < 1$: darkens, $\text{bright-value}=1$: original, $1 < \text{bright-value}$: lightens.
 - **Watermark:** `op=watermark&wmText=\watermark-text]` – Watermarks an image, with the supplied text.
 - **Grayscale:** `op=grayscale` – Converts an image to grayscale.
 - **Crop:** `op=crop&cropX=[crop-x]&cropY=[crop-y]&cropWidth=[crop-width]&cropHeight=[crop-height]` – Crops an image, starting from (crop-x,crop-y) coordinate, and crops to the optional (crop-width,crop-height) coordinate if supplied, or just to the lower-right corner of the image. All values are measured in pixels.
 - **Convert:** `op=convert&convertTo=[format]` – Converts an image to another format. Currently supported formats are: gif, jpg, tiff, png, and bmp.

Examples:

```
/imagemanip/ImageManipulation?url=http://www.explore.cornell.edu/tour_landmarks/img/frosty_tower02.jpg&op=zoom&zoomAmt=&equ  
/imagemanip/ImageManipulation?url=http://www.explore.cornell.edu/tour_landmarks/img/frosty_tower02.jpg&op=convert&convertTo=g
```

[1] This service uses the ImageJ API from Wayne Rasband which is free to use in the public domain. For additional information regarding the ImageJ API, please refer to the ImageJ project web site at: <http://rsb.info.nih.gov/ij/>.

SAXON XSLT Processor Service

Welcome to the Saxon XSLT Processor Service

This local service provides an XSLT Transformation Engine [1] for transforming XML-encoded source documents using a supplied XSLT stylesheet. The service functions as a Java servlet and can be invoked using the following syntax:

```
hostname:[port-number]/saxon/SaxonServlet?source=[xml-source]&style=[xsl-stylesheet]&clear-stylesheet-cache=yes
```

Where:

- [hostname] – The hostname of the Fedora server; default is localhost.

- [port-number] – The port number on which the Fedora server is running; default is 8080.
- [xml-source] – The URL of the XML source file.
- [xsl-stylesheet] – The URL of the XSLT stylesheet file.
- clear-stylesheet-cache – An optional parameter indicating if the stylesheet cache is to be cleared. A value of "yes" signals to clear the stylesheet cache. If the parameter is omitted or has any other value, the stylesheet cache remains untouched.

Example:

```
http://localhost:8080/saxon/SaxonServlet?source=http://dl.lib.virginia.edu/data/xmltext/ead/viu03270&style=http://dl.lib.virginia.edu/bin/...
```

Configuring Authentication

If the Saxon Servlet needs to retrieve a stylesheet or source xml that is protected by basic authentication, it can be configured to do so by setting the parameters in the servlet's `web.xml` file. In the Fedora server distribution, this file can be found under the tomcat directory in `webapps/saxon/WEB-INF`.

The example below is what needs to be added for each host:port/path combination that the servlet needs to provide credentials for. The `param-value` is a colon-delimited username and password pair. When the Saxon Servlet needs to access a URL starting with the `param-name`, it will provide these credentials. Multiple `init-param` elements may be provided.

```
<init-param>
  <param-name>credentials for localhost:8080/fedora/getDS</param-name>
  <param-value>backendUser:backendPass</param-value>
</init-param>
```

Note that the text `credentials for` must be present at the start of the `param-name`, and the URL should be supplied without the protocol part (ie `localhost:8080/fedora/path` rather than `http://localhost:8080/fedora/path`).

The Saxon Servlet will need to be restarted in order for new values to take effect.

Passing parameters

Any URL query parameters passed to the Saxon servlet are available to your stylesheet (so long as you don't use one of the standard parameter names such as `source`, `style`, `clear-stylesheet-cache`). You need to declare a parameter using `xsl:param` in your stylesheet, which you can then reference as eg `$myParam`.

Example

URL
<pre>http://localhost:8080/saxon/SaxonServlet?source=http://dl.lib.virginia.edu/data/xmltext/ead/viu03270&style=http://dl.lib.virginia.edu/bin/...&param1=SomeText</pre>

Stylesheet
<pre><?xml version="1.0" encoding="ISO-8859-1"?> <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <xsl:param name="param1"/> <xsl:template match="/"> <output> <xsl:value-of select="\$param1"/> </output> </xsl:template> </xsl:stylesheet></pre>

Passing additional XML documents

If a URL query parameter specifies the URL of an XML document, the contents of the document can be processed using the `xslt:document(...)` function.

For example, if you construct an `SDef` object that binds to a datastream and passes this to the Saxon servlet as a URL query parameter, you can access this additional datastream content (in addition to the `source` parameter as the main `xslt` input) using this method.

Example

URL

`http://localhost:8080/saxon/SaxonServlet?source=http://dl.lib.virginia.edu/data/xmltext/ead/viu03270&style=http://dl.lib.virginia.edu/bin/e
¶m1=http://dl.lib.virginia.edu/data/xmltext/another-input`

Stylesheet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="param1"/>
  <xsl:template match="/">
    <output>
      <xsl:value-of select="document($param1)/path/to/element"/>
    </output>
  </xsl:template>
</xsl:stylesheet>
```

Setting the Internet Media Type (MIMEType)

By default the Saxon servlet will return a media type of `text/html` if none has been defined in your stylesheet. If your stylesheet is generating XML you may find it useful to set the media type so the correct type will be returned - for instance to a browser via an object method (disseminator). To do this include an `xsl:output` element as a top-level element in your stylesheet, with a `media-type` attribute defining the media type.

Internet Media Type example

```
<xsl:output indent="yes" method="xml" media-type="text/xml"/>
```

^[1] This service uses the SAXON XSLT Processor from Michael Kay which is licensed under the Mozilla Public License (MPL). For additional information regarding the SAXON XSLT Processor, please refer to the Saxon project web site on SourceForge at: <http://saxon.sourceforge.net/>.

Frequently Asked Questions

Administration FAQ

- How do I configure Fedora to allow API-M access from remote hosts?
- How do I turn off API-M access?
- How do I undo SSL-only access to API-M functions?

Backup And Rebuild FAQ

- How can I backup my Fedora Repository without having to rebuild ?
- Is it possible to rebuild without taking my repository offline?
- What do I need to backup in order to be able to rebuild my Fedora Repository ?

Development FAQ

- How do you configure Eclipse for Fedora with Maven?
- Where are Fedora's bugs reported?
- Where is the source code ? How can it be accessed ?

Installation FAQ

- How do I get the get Fedora running under the JBoss Application Server?
- How do I uninstall Fedora Commons ?
- Which Linux Distribution is best for Fedora Commons?

Usage FAQ

- "Policy blocked datastream resolution" error while adding datastream
- How can I implement collections in Fedora Commons?
- How can I retrieve the audit datastream ?
- How do I customize the HTML views produced by the REST API?
- How do I set my own PIDs (at runtime) for Fedora objects?
- Mime type for SDep Output
- Must I provide DC, RELS-EXT and RELS-INT as Inline XML Metadata?
- Resource Index date comparison with ITQL
- Where is the list of Dublin Core and FOXML properties?

Administration FAQ

- How do I configure Fedora to allow API-M access from remote hosts?
- How do I turn off API-M access?
- How do I undo SSL-only access to API-M functions?

How do I configure Fedora to allow API-M access from remote hosts?

How do I configure Fedora to allow API-M access from remote hosts?

Fedora's default server security settings disallow remote administrative access. This restriction can be relaxed by either:

1. Modifying (or removing) the `deny-apim-if-not-localhost.xml` policy file.

See the default policy documentation in the user guide:

<http://www.fedora-commons.org/confluence/display/FCR30/Fedora+Authorization+with+XACML+Policy+Enforcement>

See the "Activating and Loading" instructions in the user guide:

<http://www.fedora-commons.org/confluence/display/FCR30/Fedora+Authorization+with+XACML+Policy+Enforcement>

2. Disabling policy enforcement altogether

See the "Enabling/Disabling XACML Policy Enforcement" section in the user guide:

<http://www.fedora-commons.org/confluence/display/FCR30/Fedora+Authorization+with+XACML+Policy+Enforcement>

How do I turn off API-M access?

Question: I would like to prevent changes to my repository for some period of time. Is there a way to go "read-only" and disable API-M?

Answer: Yes; if you have XACML policy enforcement enabled, you can disable all API-M requests to your repository via policy. While disabled, all API-M requests will result in an "Authorization Denied" message for the requesting user or application. As with all XACML policy changes, it is not necessary to restart your repository to put the new rules into effect.

Instructions:

1. Ensure you have XACML policy enforcement enabled. This is the default option with Fedora 3.x, so it is likely already enabled for you. You can verify by opening your \$FEDORA_HOME/server/config/fedora.fcfg, and checking the value of the ENFORCE_MODE parameter. The value should be "enforce-policies". If it is not, you will need to change it, then restart Fedora.
2. Create a new file at \$FEDORA_HOME/data/fedora-xacml-policies/repository-policies/default/read-only.xml with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        PolicyId="disable-writes"

RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
  <Description>disable writes</Description>
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">urn:fedora:names:fedora:2.1:action:api-m</Attri
<ActionAttributeDesignator
DataType="http://www.w3.org/2001/XMLSchema#string"
          AttributeId="urn:fedora:names:fedora:2.1:action:api"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Rule RuleId="1" Effect="Deny"/>
</Policy>
```

3. Run \$FEDORA_HOME/server/bin/fedora-reload-policies.sh http username password (in Windows, the path to the script is %FEDORA_HOME%\server\bin\fedora-reload-policies.bat)
4. When you want to re-enable API-M access, simply delete the file and run fedora-reload-policies again.

How do I undo SSL-only access to API-M functions?

Question: I inadvertently chose SSL for API-M access when I installed Fedora, and now I want plain HTTP access to API-M functions without reinstalling Fedora. How can I reconfigure Fedora to allow non-SSL access to API-M?

Answer: In the installed webapp, the WEB-INF/web.xml file has a <security-constraint> block with a description of APIM. There's a <user-data-constraint> block with a transport-guarantee value of **CONFIDENTIAL**. If you comment out or remove that <user-data-constraint> block, Tomcat won't require SSL for those resources (API-M).

Instructions:

1. Edit the WEB-INF/web.xml under the fedora webapp directory in your webapp server (Tomcat, in most cases). Comment out the relevant <user-data-constraint> block:


```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Fedora Repository Server</web-resource-name>
    <description>Fedora-generated security-constraint</description>
    <description>APIM</description>
    <url-pattern>/index.html</url-pattern>
    <url-pattern>/getDSAAuthenticated</url-pattern>
    <url-pattern>/management/getNextPID</url-pattern>
    <url-pattern>/management/upload</url-pattern>
    <url-pattern>/services/management</url-pattern>
    <url-pattern>*.jws</url-pattern>
  </web-resource-collection>
  <!-- Commented out to disable SSL-only access to API-M functions -->
  <!-- <user-data-constraint> -->
  <!--   <transport-guarantee>CONFIDENTIAL</transport-guarantee> -->
  <!-- </user-data-constraint> -->
</security-constraint>
```

2. Restart the Fedora web application.



Note that if you only change the `web.xml` file under `fedora/webapps`, your change will be lost the next time `fedora.war` is redployed. To persist your changes across deployments, unpack the `fedora.war` file, edit the `WEB-INF/web.xml` file as described above, repack the WAR and drop it back in place in your `webapps` directory.

Backup And Rebuild FAQ

- How can I backup my Fedora Repository without having to rebuild ?
- Is it possible to rebuild without taking my repository offline?
- What do I need to backup in order to be able to rebuild my Fedora Repository ?

How can I backup my Fedora Repository without having to rebuild ?

Question: My Fedora Repository is very big and a complete rebuild would take too long. How can I backup my repository so that I don't have to rebuild it ?

Answer: When performing a complete backup, several steps are involved:

1. Copy `disable-writes.xml` (see below) into your `$FEDORA_HOME/data/fedora-xacml-policies/repository-policies/default/directory`
2. Run `$FEDORA_HOME/server/bin/fedora-reload-policies.sh http admin-user admin-pass` (this immediately makes any repo-wide policy changes active without requiring a restart)
3. Wait a few minutes to let any in-progress writes complete.
4. Do a database+filesystem backup (see [this page](#) for the files involved)
5. Remove `disable-writes.xml`
6. Run `fedora-reload-policies` again

This could be scripted as part of a regular backup process. Step 3 is not perfect, however: if you wait 5 minutes but someone is in the middle of upload a multi-GB file to the repository, you might still get an inconsistent backup. You could also watch the low level system activity (thread dump, `strace`, `lsof`, etc) to see if there are any ongoing writes.

```

<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:1.0:policy"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  PolicyId="disable-writes"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
  <Description>disable writes</Description>
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <AnyResource/>
    </Resources>
    <Actions>
      <Action>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">urn:fedora:names:fedora:2.1:action:api-m</AttributeValue>
          <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="urn:fedora:names:fedora:2.1:action:api"/>
        </ActionMatch>
      </Action>
    </Actions>
  </Target>
  <Rule RuleId="1" Effect="Deny"/>
</Policy>

```

Is it possible to rebuild without taking my repository offline?

Question: My repository is very large and I cannot afford to take it offline for a long rebuild. Is there a way to do it while keeping the repository running?

Answer: It is possible to reconstitute the database and/or resource index while the repository is running in read-only mode. This is called a *warm rebuild*. People will still be able to access (but not change) items in your repository while the rebuild is taking place, but a quick restart will be required at the end of the process.

The `fedora-rebuild` utility was originally designed to run *cold rebuilds* against offline repositories, but with some extra work, you can also use it to rebuild into a new store (resource index or database) while your repository is still handling requests using the original store. At the end of this process, you will need to tell Fedora to use the new store, which requires a restart.



These instructions have been tested in the following configurations:

- Fedora 3.3 with embedded Derby and local Mulgara (Database and Resource Index rebuild)
- Fedora 2.2.3 with embedded McKoi (Database rebuild only)

Instructions:

1. Prepare
2. Disable writes
3. Run the rebuild
4. Copy temporary configuration to live repository
5. Restart and verify
6. Re-enable writes
7. Clean up

1. Prepare

Create a temporary Fedora Home directory

This directory will contain a subset of what is currently in your live repository's `$FEDORA_HOME` directory in order to support running the rebuild with an alternate configuration. This can be anywhere you like. Below, we'll assume it's `/tmp/temp-home`, or `C:\temp-home` if you're running Windows.

Unix-based:

```
export ORIG_HOME=$FEDORA_HOME
export TEMP_HOME=/tmp/temp-home
mkdir $TEMP_HOME
mkdir $TEMP_HOME/server
mkdir $TEMP_HOME/server/bin
mkdir $TEMP_HOME/server/config
cp $ORIG_HOME/server/bin/* $TEMP_HOME/server/bin
cp $ORIG_HOME/server/config/* $TEMP_HOME/server/config
```

Windows:

```
set ORIG_HOME=%FEDORA_HOME%
set TEMP_HOME=C:\temp-home
mkdir %TEMP_HOME%
mkdir %TEMP_HOME%\server
mkdir %TEMP_HOME%\server\bin
mkdir %TEMP_HOME%\server\config
copy %ORIG_HOME%\server\bin\*. * %TEMP_HOME%\server\bin
copy %ORIG_HOME%\server\config\*. * %TEMP_HOME%\server\config
```

Configure the new store

1. Ensure that the temporary Fedora is configured to point at the original low level storage locations.
 - a. Open \$TEMP_HOME/server/config/fedora.fcfg
 - b. Search for ILowlevelStorage
 - c. If the "class" attribute ends with ".DefaultLowlevelStorageModule":
 - i. Ensure that the object_store_base param specifies an absolute directory (by default, it is a relative directory)
 - ii. Do the same for datastream_store_base
 - d. If the "class" attribute ends with ".AkubraLowlevelStorageModule", no change is needed; Akubra's configuration already specifies an absolute directory.
 - e. If the "class" attribute is anything else, ensure that all path-oriented params are specifying the path in absolute form.
2. If you will be doing a Resource index rebuild, initialize a new triplestore:
 - a. Open \$TEMP_HOME/server/config/fedora.fcfg
 - b. Locate the datastore element (near the end of the file) corresponding to the triplestore you're using.
 - c. If you're using Mulgara/Kowari:
 - i. If the remote param value is "false", change the path param to point to the new, preferred location of your rebuilt resource index data directory. Specify this as an absolute path. For example, "/opt/fedora/data/resourceIndexRebuilt". Do not create this directory yourself; it will be automatically created at the beginning of the resource index rebuild.
 - ii. If the remote param value is "true":
 1. Bring up a new instance of Mulgara/Kowari, on a different host and/or port than your existing instance.
 2. Change/add the "host" and "port" params accordingly.
 - d. If you're using MPTStore:
 - i. Modify the jdbcURL param to use a different database name. For example, if it was "jdbc:postgresql:riTriples", change it to "jdbc:postgresql:riTriplesRebuilt"
 - ii. Create this new, empty database using the method recommended by your database vendor and ensure that the database user (username) specified in this section of fedora.fcfg has full permission on it.
3. If you will be doing a database rebuild, initialize a new database:
 - a. If you're using a non-bundled database (MySQL, Oracle, PostgreSQL):
 - i. Open \$TEMP_HOME/server/config/fedora.fcfg
 - ii. Locate the datastore element (near the end of the file) corresponding to the database you're using.
 - iii. Modify the jdbcURL param to use a different database name. For example, if it was "jdbc:postgresql:fedora3", change it to "jdbc:postgresql:fedora3rebuilt"
 - iv. Create this new, empty database using the method recommended by your database vendor and ensure that the database user (dbUsername) specified in this section of fedora.fcfg has full permission on it.
 - b. If you're using the bundled Derby database:
 - i. Open \$TEMP_HOME/server/config/fedora.fcfg
 - ii. Search for jdbc:derby
 - iii. Change the jdbcURL param to point to the new, preferred location of your rebuilt database. For example, if the value is currently "jdbc:derby:/opt/fedora/derby/fedora3;create=true", change it to "jdbc:derby:/opt/fedora/derby/fedora3-rebuilt". Do not create this directory yourself; it will be automatically created at the beginning of the database rebuild.
 - c. If you're using the bundled McKoi database:
 - i. Open \$TEMP_HOME/server/config/fedora.fcfg
 - ii. Search for jdbc:mckoi
 - iii. Change the jdbcURL param to point to the new, preferred location of your rebuilt database. For example, if the value is currently "jdbc:mckoi:local:///opt/fedora/mckoi1.0.3/db.conf?create_or_boot=true", change it to "jdbc:mckoi:local:///opt/fedora/mckoi1.0.3-rebuilt/db.conf?create_or_boot=true"
 - iv. Create this new directory. For example, "mkdir /opt/fedora/mckoi1.0.3-rebuilt"
 - v. Copy all files (but not subdirectories) from the original mckoi directory to this new one. For example, "cp

/opt/fedora/mckoi1.0.3/* /opt/fedora/mckoi1.0.3-rebuilt".

2. Disable writes

See [How do I turn off API-M access?](#)

3. Run the rebuilder

First, make sure your FEDORA_HOME and CATALINA_HOME environment variables are set correctly:

Unix-based:

```
export FEDORA_HOME=$TEMP_HOME
echo $FEDORA_HOME
echo $CATALINA_HOME
```

Windows:

```
set FEDORA_HOME=%TEMP_HOME%
echo %FEDORA_HOME%
echo %CATALINA_HOME%
```

The echoed value of FEDORA_HOME should be the path to your temporary Fedora Home directory (e.g. **/tmp/temp-home**) and the echoed value of CATALINA_HOME should be the path to your live repository's tomcat directory (e.g. **/opt/fedora/tomcat**).

Next, run the rebuilder [as described here], but don't stop the server.

NOTE: If you need to rebuild *both* the database and resource index, make sure you have configured both, and *rebuild the database first*, followed by the resource index.

4. Copy temporary configuration to live repository

- Make a backup of \$ORIG_HOME/server/config/fedora.fcfg
- Copy \$TEMP_HOME/server/config/fedora.fcfg to \$ORIG_HOME/server/config/fedora.fcfg (replacing the existing file)

5. Restart and verify

- Set the FEDORA_HOME environment variable back to the original value (\$ORIG_HOME)
- Run \$CATALINA_HOME/bin/shutdown.sh (%CATALINA_HOME%\bin\shutdown.bat if running Windows)
- Run \$CATALINA_HOME/bin/startup.sh (%CATALINA_HOME%\bin\startup.bat if running Windows)
- Check \$FEDORA_HOME/server/logs/fedora.log for any sign of errors at startup.
- Try some queries on the /fedora/search and /fedora/risearch web interfaces and make sure they behave as expected.

6. Re-enable writes

See [How do I turn off API-M access?](#)

7. Clean up

You can now safely remove the original database and/or resource index data to recover disk space. You can also remove the \$TEMP_HOME directory.

What do I need to backup in order to be able to rebuild my Fedora Repository ?

All you absolutely need are the datastream and object directories, wherever you configured them to be.

The locations are the values of the "object_store_base" and "datastream_store_base" params in \$FEDORA_HOME/sever/config/fedora.fcfg.

As per default they point to the following directories:

\$FEDORA_HOME/data/objects (contains all FOXML files)

\$FEDORA_HOME/data/datastreams (contains all binary files from the managed datastreams)

Anything within these directories must be backed up.

NOTE: This assumes you're using the default standard filesystem storage, and haven't configured another storage layer implementation. Optionally, you can also backup your custom XACML policies, should you have any. That location is configured in the param "REPOSITORY-POLICIES-DIRECTORY". The default points to \$FEDORA_HOME/data/fedora-xacml-policies/repository-policies.

If you have an active FeSL installation, the backup should also include `$FEDORA_HOME/pdp/conf` and `$FEDORA_HOME/pdp/policies`. Also, you should find a `$FEDORA_HOME/install` directory containing an `install.properties` file. It's useful to also back up that file, as when you re-install you can use that file during the installation to reinstall with the same options (without having to enter them manually) with:

```
java -jar fedora-installer.jar install.properties
```

(substituting the actual name of the fedora installer jar which will depend on which release you're using)

Development FAQ

- [How do you configure Eclipse for Fedora with Maven?](#)
- [Where are Fedora's bugs reported?](#)
- [Where is the source code ? How can it be accessed ?](#)

How do you configure Eclipse for Fedora with Maven?



Eclipse Helios

Eclipse Helios is now available. This document is in the process of being updated as the new release is tested. There is a known [problem](#) which causes lockup with the Sun JDK 1.6.0_21 which is resolved in the current Helios download and also has a manual workaround. Note that this problem still exists in Galileo and Ganymede. We have performed a successful build from Helios on WinXP and Windows 7 with Subversion and M2Eclipse installed. Help from the community would be appreciated.

How do I configure Eclipse to work with Fedora with Maven?

Eclipse is a popular integrated development environment (IDE) which is used by many developers for the Fedora Repository service, clients and other related services. You do not need to build the Fedora Repository from source in Eclipse in order to build your own projects though it may be convenient. You may build the Fedora Repository from the command line and just use Eclipse for editing and debugging. The options are too numerous for one article so we will describe just a few popular configurations to get you started.

The current Fedora Repository trunk (Fedora 3.3 and greater) uses [Apache Maven](#) for building the source. Maven helps manage dependencies and makes project information readily available. You can build Fedora using Maven from the command line or within Eclipse. Like Eclipse, Maven is based on the idea of using "convention over configuration." While Maven is very flexible, it promotes good modularization practices which should make using Eclipse for Fedora easier.

This article uses the Fedora Repository service as an example. It is important that you read the [Installation and Configuration Guide](#) first. It contains a significant amount of information about the Fedora Repository and building it from source which is not repeated here.

Steps

1. Ensure you have the latest version of the Sun 1.6 JDK installed
2. Install any Eclipse package (Galileo is recommended though Ganymede should work)
3. Configure the JDK in Eclipse
4. Configure Maven
5. Install a Subversion Client or Subversion Eclipse plug-in (recommended)
6. Install the **M2Eclipse** plug-in
7. Create an Eclipse project (using a Subversion plug-in and M2Eclipse)
8. Customize Eclipse workspace settings (required for committers, optional for others)
9. Add other interesting plug-ins

1. Installing the JDK

Fedora is only tested with the Sun JDK though others may work well or even be required for a particular operational environments such as specific application servers. Eclipse has a very sophisticated structure which supports the installation and use of multiple Java compilers and run time environments. Initially, we recommend installing the Sun JDK to get success and then you may explore other options.

The Oracle/Sun JDK may be obtained from [here](#) for common platforms. Fedora is developed using Java 1.6. It is not required that Eclipse be running in the same JDK/JRE version as is being used for development but it is convenient if you use the same JDK/JRE version on the command line as you are using for Eclipse. Note that a JDK is required for some operations of the M2Eclipse plug-in described below.

It is also convenient for you install the JDK outside "Program Files" if you are developing in Microsoft Windows, using a directory with no spaces in the name. Eclipse and the Fedora command-line Maven-based build tools will work with proper set-up but you will have less trouble if you avoid directories with spaces in their names. Also, very long directory structures may be found in Eclipse projects and generally don't cause problems with Eclipse, but may cause problems with other products in Windows XP. So having a top-level `workspaces` directory is helpful.

2. Install Eclipse

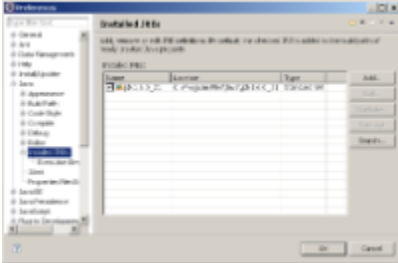
Download and install [Eclipse](#). You may install any package since you can add plug-ins later but it is convenient to install the "Eclipse IDE for Java

EE Developers" since it already contains most of the plug-ins you may want to use. It is recommended that you install the JDK separately since you may wind up using several versions.

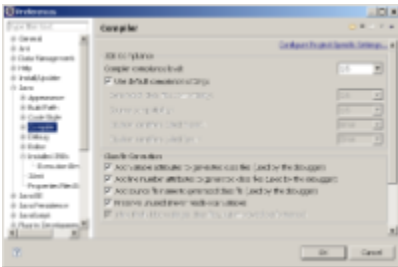
3. Select the JDK and Set the Compiler Compliance

Make sure Java 1.6 appears as an installed JRE and that it is the default JRE for the project.

A) Use the Eclipse Menu: Window -> Preferences -> Java -> Installed JREs. Despite the name of the menu item it is highly recommended that you install JDKs. Use of a JDK is required for the M2Eclipse Maven plug-in described below. Use the Add button to add the Java 1.6 JDK if it is not already installed. Check the Java 1.6 JDK and press OK.



B) Use the Eclipse Menu: Window -> Preferences -> Java -> Compiler. Select 1.6 in the Compiler compliance level: combo box and press Apply.



4. Configure Maven

The M2Eclipse plug-in includes an embedded Maven. Optionally you can also install Maven on the command-line. Regardless, you must have a local Maven repository to cache dependencies and work products from the builds. A command-line Maven can be obtained from <http://maven.apache.org/>. **If Eclipse cannot find the local Maven repository you will have ugly errors and Eclipse may lock-up.**

Duraspace provides a Maven repository for Fedora dependencies which cannot be obtained from Maven Central or common public Maven repositories. You do not have to do anything to use this repository, everything is handled by Maven from specifications in the POMs.

Maven will look for the repository in several standard locations. However, you can make things easier if you create a setting file. Settings may be obtained globally from the Maven installation or in a settings file associated with your user. The user settings file varies depending on the operating system (in Windows - "Documents and Settings\username\.m2\settings.xml", in Linux - "home/username/.m2/settings.xml"). The global setting file is usually in <maven-install>/conf. If you install a command-line Maven you may accidentally wind up with two local Maven repositories. Just make sure everything points to the same local repository. If you have problems may help to set the environment variable M2_HOME to <maven-install> and put <M2_HOME>/bin on your path.

Example user settings.xml

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
    http://maven.apache.org/xsd/settings-1.0.0.xsd">
  <localRepository>d:/m2/repository</localRepository>
</settings>
```

Example global settings.xml

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

<!--
Licensed to the Apache Software Foundation (ASF) under one
```

or more contributor license agreements. See the NOTICE file distributed with this work for additional information regarding copyright ownership. The ASF licenses this file to you under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

-->

<!--

| This is the configuration file for Maven. It can be specified at two levels:

1. User Level. This settings.xml file provides configuration for a single user, and is normally provided in `${user.home}/.m2/settings.xml`.

| NOTE: This location can be overridden with the CLI option:

| `-s /path/to/user/settings.xml`

2. Global Level. This settings.xml file provides configuration for all Maven users on a machine (assuming they're all using the same Maven installation). It's normally provided in `${maven.home}/conf/settings.xml`.

| NOTE: This location can be overridden with the CLI option:

| `-gs /path/to/global/settings.xml`

| The sections in this sample file are intended to give you a running start at getting the most out of your Maven installation. Where appropriate, the default values (values used when the setting is not specified) are provided.

|-->

```
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
http://maven.apache.org/xsd/settings-1.0.0.xsd">
```

```
  <!-- localRepository
```

```
  | The path to the local repository maven will use to store artifacts.
```

```
  | Default: ~/.m2/repository
```

```
<localRepository>/path/to/local/repo</localRepository>
```

```
-->
```

```
<localRepository>d:/m2/repository</localRepository>
```

```
  <!-- interactiveMode
```

```
  | This will determine whether maven prompts you when it needs input. If set to false,
  | maven will use a sensible default value, perhaps based on some other setting, for
  | the parameter in question.
```

```
  | Default: true
```

```
<interactiveMode>>true</interactiveMode>
```

```
-->
```

```
  <!-- offline
```

```
  | Determines whether maven should attempt to connect to the network when executing a build.
  | This will have an effect on artifact downloads, artifact deployment, and others.
```

```
  | Default: false
```

```
<offline>>false</offline>
```

```
-->
```

```

<!-- pluginGroups
| This is a list of additional group identifiers that will be searched when resolving plugins by
their prefix, i.e.
| when invoking a command line like "mvn prefix:goal". Maven will automatically add the group
identifiers
| "org.apache.maven.plugins" and "org.codehaus.mojo" if these are not already contained in the
list.
|-->
<pluginGroups>
  <!-- pluginGroup
  | Specifies a further group identifier to use for plugin lookup.
  <pluginGroup>com.your.plugins</pluginGroup>
  -->
</pluginGroups>

<!-- proxies
| This is a list of proxies which can be used on this machine to connect to the network.
| Unless otherwise specified (by system property or command-line switch), the first proxy
| specification in this list marked as active will be used.
|-->
<proxies>
  <!-- proxy
  | Specification for one proxy, to be used in connecting to the network.
  |
  <proxy>
    <id>optional</id>
    <active>true</active>
    <protocol>http</protocol>
    <username>proxyuser</username>
    <password>proxypass</password>
    <host>proxy.host.net</host>
    <port>80</port>
    <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
  </proxy>
  -->
</proxies>

<!-- servers
| This is a list of authentication profiles, keyed by the server-id used within the system.
| Authentication profiles can be used whenever maven must make a connection to a remote server.
|-->
<servers>
  <!-- server
  | Specifies the authentication information to use when connecting to a particular server,
identified by
  | a unique name within the system (referred to by the 'id' attribute below).
  |
  | NOTE: You should either specify username/password OR privateKey/passphrase, since these
pairings are
  |         used together.
  |
  <server>
    <id>deploymentRepo</id>
    <username>repouser</username>
    <password>repopwd</password>
  </server>
  -->

  <!-- Another sample, using keys to authenticate.
  <server>
    <id>siteServer</id>
    <privateKey>/path/to/private/key</privateKey>
    <passphrase>optional; leave empty if not used.</passphrase>
  </server>
  -->
</servers>

<!-- mirrors
| This is a list of mirrors to be used in downloading artifacts from remote repositories.
|

```



```

| It works like this: a POM may declare a repository to use in resolving certain artifacts.
| However, this repository may have problems with heavy traffic at times, so people have mirrored
| it to several places.
|
| That repository definition will have a unique id, so we can create a mirror reference for that
| repository, to be used as an alternate download site. The mirror site will be the preferred
| server for that repository.
|-->
<mirrors>
  <!-- mirror
  | Specifies a repository mirror site to use instead of a given repository. The repository that
  | this mirror serves has an ID that matches the mirrorOf element of this mirror. IDs are used
  | for inheritance and direct lookup purposes, and must be unique across the set of mirrors.
  |
  <mirror>
    <id>mirrorId</id>
    <mirrorOf>repositoryId</mirrorOf>
    <name>Human Readable Name for this Mirror.</name>
    <url>http://my.repository.com/repo/path</url>
  </mirror>
  -->
</mirrors>

<!-- profiles
| This is a list of profiles which can be activated in a variety of ways, and which can modify
| the build process. Profiles provided in the settings.xml are intended to provide local machine-
| specific paths and repository locations which allow the build to work in the local environment.
|
| For example, if you have an integration testing plugin - like cactus - that needs to know where
| your Tomcat instance is installed, you can provide a variable here such that the variable is
| dereferenced during the build process to configure the cactus plugin.
|
| As noted above, profiles can be activated in a variety of ways. One way - the activeProfiles
| section of this document (settings.xml) - will be discussed later. Another way essentially
| relies on the detection of a system property, either matching a particular value for the
property,
| or merely testing its existence. Profiles can also be activated by JDK version prefix, where a
| value of '1.4' might activate a profile when the build is executed on a JDK version of
'1.4.2_07'.
| Finally, the list of active profiles can be specified directly from the command line.
|
| NOTE: For profiles defined in the settings.xml, you are restricted to specifying only artifact
| repositories, plugin repositories, and free-form properties to be used as configuration
| variables for plugins in the POM.
|-->
<profiles>
  <!-- profile
  | Specifies a set of introductions to the build process, to be activated using one or more of the
  | mechanisms described above. For inheritance purposes, and to activate profiles via
<activatedProfiles/>
  | or the command line, profiles have to have an ID that is unique.
  |
  | An encouraged best practice for profile identification is to use a consistent naming convention
  | for profiles, such as 'env-dev', 'env-test', 'env-production', 'user-jdcasey', 'user-brett',
etc.
  | This will make it more intuitive to understand what the set of introduced profiles is
attempting
  | to accomplish, particularly when you only have a list of profile id's for debug.
  |
  | This profile example uses the JDK version to trigger activation, and provides a JDK-specific
repo.
  <profile>
    <id>jdk-1.4</id>

    <activation>
      <jdk>1.4</jdk>
    </activation>

    <repositories>

```

```

    <repository>
      <id>jdk14</id>
      <name>Repository for JDK 1.4 builds</name>
      <url>http://www.myhost.com/maven/jdk14</url>
      <layout>default</layout>
      <snapshotPolicy>always</snapshotPolicy>
    </repository>
  </repositories>
</profile>
-->

<!--
| Here is another profile, activated by the system property 'target-env' with a value of 'dev',
| which provides a specific path to the Tomcat instance. To use this, your plugin configuration
| might hypothetically look like:
|
| ...
| <plugin>
|   <groupId>org.myco.myplugins</groupId>
|   <artifactId>myplugin</artifactId>
|
|   <configuration>
|     <tomcatLocation>${tomcatPath}</tomcatLocation>
|   </configuration>
| </plugin>
| ...
|
| NOTE: If you just wanted to inject this configuration whenever someone set 'target-env' to
|       anything, you could just leave off the <value/> inside the activation-property.
|
</profile>
  <id>env-dev</id>

  <activation>
    <property>
      <name>target-env</name>
      <value>dev</value>
    </property>
  </activation>

  <properties>
    <tomcatPath>/path/to/tomcat/instance</tomcatPath>
  </properties>
</profile>
-->
</profiles>

<!-- activeProfiles
| List of profiles that are active for all builds.
|
<activeProfiles>
  <activeProfile>alwaysActiveProfile</activeProfile>
  <activeProfile>anotherAlwaysActiveProfile</activeProfile>

```

```
</activeProfiles>
-->
</settings>
```

5. Install a Subversion Client

It is recommended that you install a Subversion plug-in, either [Subversive](#) or [Subclipse](#) (use only one or bad things will happen), even if you only intend to build from the command-line. Installing a Subversion plug-in enables you to see the history of files and often is required for advanced functions of other plug-ins. Unfortunately, some plug-ins may work only with Subversive or Subclipse but not both. Please provide comments on this article regarding your experience.

A Subversion plug-in is required for a number of functions in the Maven plug-in M2Eclipse [described below](#). Subclipse is provided by the same organization as M2Eclipse however Subversive integration with M2Eclipse is available and in our testing seems to support all common operations. Also, Subversive is the incubated Eclipse plug-in.

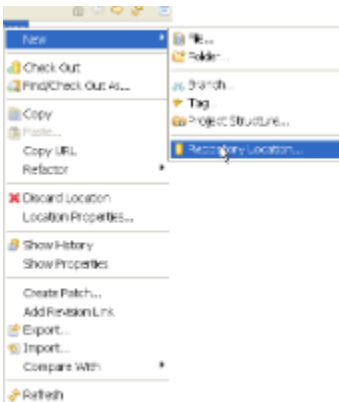
Many developers use a command-line [svn](#) or graphical Subversion client such as [Tortoise](#) as their primary SCM client. Building from the command-line and using Eclipse primarily for editing, debugging, and other functions is a common approach for developing Fedora Commons projects (see [Building the Fedora Repository](#)). However, it is still very helpful to have a Subversion plug-in installed. One Maven plug-in we use to automatically create build numbers, `buildnumber-maven-plugin`, and depends on having a command-line Subversion available, but will only generate a warning which can be ignored if one is not present.

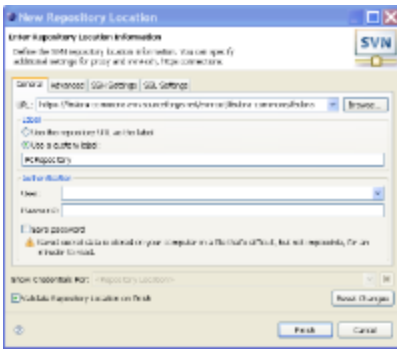
In the following example we show the steps for using Subversive. Installing Subversive is a bit complicated and is changing as it is moving through Eclipse incubation. Once you have installed a Subversion client the following steps may be used to customize your Eclipse settings and to link to the Fedora Subversion repository. Note that some of the steps can only be accomplished after you have downloaded the Fedora sources preferably using Subversion. Don't worry, just get them done before you start editing or building the code.

A) Use the Eclipse Menu: Window -> Open Perspective -> Other. Using the Open Perspective dialog select `SVN Repository Exploring` and press Apply.

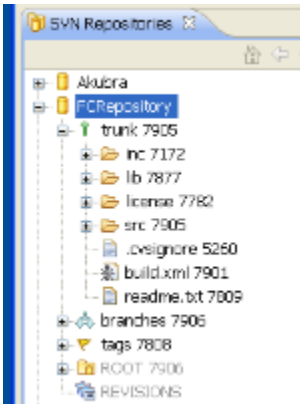


B) Use the right-mouse click menu or the Eclipse Menu: File -> New -> Repository Location. Enter the URL of the Subversion repository (no authentication is needed to download), optionally add a custom label and press OK.

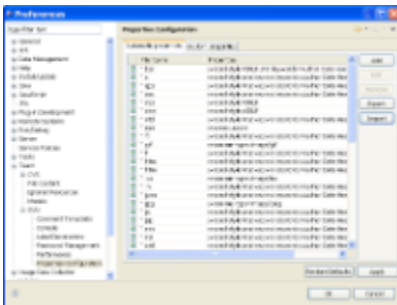




C) Currently the Fedora Repository Project Subversion repository is located at: <https://fedora-commons.svn.sourceforge.net/svnroot/fedora-commons/fedora> on *Sourceforge.net*. After you have successfully define the repository location you should be able to see its *trunk*, *branches*, and *tags*.



D) Use the Eclipse Menu: Window -> Preferences -> Team -> SVN -> Properties Configuration. On the *Automatic Properties* tab import `resources/devdocs/devenv/subversion.config` from the Fedora Repository sources and press *Apply*. Please note, its best to do this step later after you have installed M2Eclipse and downloaded the project.



6. Install the M2Eclipse Plug-in

There are Eclipse targets built into Maven but the [M2Eclipse](#) from Sonatype provides a capable and well documented addition to your Eclipse IDE. It is highly recommended that you install it. The installation is well documented and there is an online book [Developing with Eclipse and Maven](#) that is highly recommended reading. Note that the installation process only documents the use of Subclipse, another Sonatype contribution. However, there is a Subversive integration available which in our testing supports common used M2Eclipse functionality (you must install the [Subversive Integration for the M2Eclipse Project plug-in](#) [here](#)).

M2Eclipse will complain if you are not running Eclipse within a JDK are instead using a stand-alone JRE. This will require you to edit the `eclipse.ini` file found in top-level director of your Eclipse installation. The formatting of this file is tricky and unforgiving. Below is an example from a Windows installation:

```

-startup
plugins/org.eclipse.equinox.launcher_1.0.200.v20090520.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_1.0.200.v20090519
-product
org.eclipse.epp.package.jee.product
--launcher.XXMaxPermSize
256M
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
-vm
c:/java/jdk1.6.0_13/bin/javaw.exe
-vmargs
-Dosgi.requiredJavaVersion=1.6
-Xms512m
-Xmx1024m

```

7. Create an Eclipse Project

You must create an Eclipse project for Fedora. The layout of the Fedora Repository source pre-dated the use of Eclipse and does not follow the most natural layout for Eclipse projects. However, Eclipse can be configured to be an effective IDE for development. The ongoing refactoring for Maven is making it more natural to build using Eclipse and enabling the use of Eclipse plug-ins. There are actually too many ways to configure Eclipse for us to document here. We will, however, illustrate one common configuration which you may use (and challenge the community to add more options to this article).



Eclipse Friendliness

Refactoring of Fedora is underway to improve modularity which will help make Fedora more Eclipse friendly. Code sections have been entirely restructured to build with Maven. However, there is still work to be done in refactoring Fedora to improve modularity. You can follow this ongoing effort now in this Wiki and JIRA. The work at [ANT to Maven2](#) is concluded.

There is an easy way to create the Fedora Repository project with M2Eclipse. First you must specify the repository location using the SVN Repository Exploring perspective. Then using the Checkout as Maven Project wizard you will create the `fedora-repository` project and import the source code. As a prerequisite, we assume you have already successfully configured the [Subversion](#) client in Eclipse, installed M2Eclipse and configured your local Maven repository. The following example uses the Subclipse plug-in.

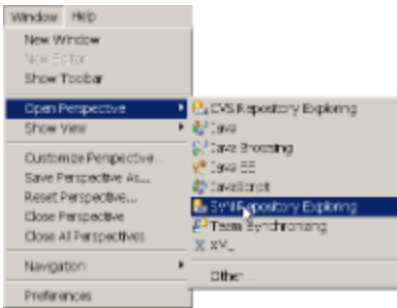
1. Set up the Subversion Repository
2. Select the *trunk* or the *branch* with which you wish to work
3. Use the Check Out as Maven Project Wizard
4. Create a Maven Run As Maven Build configuration
5. Use the Run As Maven Build configuration to build the project the first time

Set Up the Source Repository and Create the Project

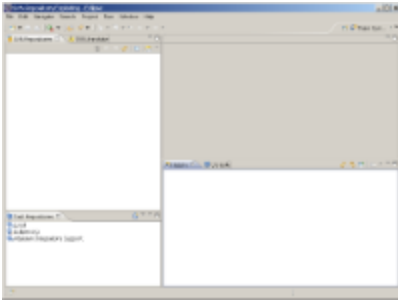
A) In a fresh install of Eclipse you will see the Welcome screen. It will help if you use the Eclipse Menu: Project -> Build Automatically to uncheck this function. We will use the Maven build for the first build of the repository.



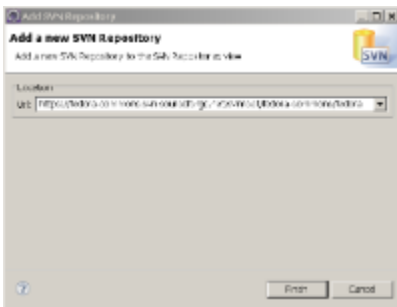
B) Using the Eclipse Menu: Select the Window -> Open Perspective -> SVN Repository Exploring open the perspective.



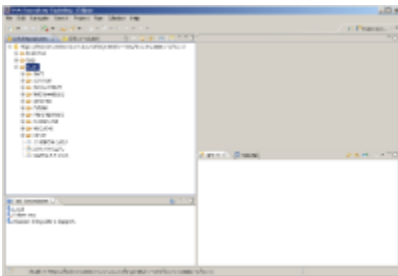
C) You should see the following perspective:



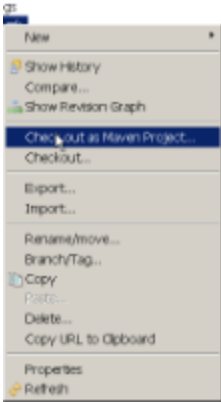
D) Right click in SVN Repositories view to display the Add SVN Repository dialog. Use the URL <https://fedora-commons.svn.sourceforge.net/svnroot/fedora-commons/fedora>.



E) Open the repository and select the trunk or a desired branch. This may take some time depending on the response of the Subversion repository.



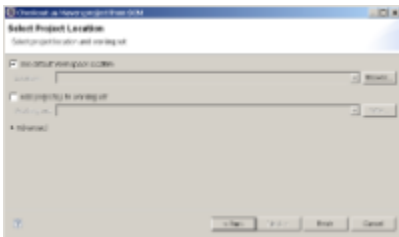
F) Right click to display the SVN menu and select Check out as Maven Project...



G) Select the target location and revision. If you do not see `svn` as a choice in the `SCM URL` combo box Subclipse is not correctly set up. Dialogs for Subversive may be different. If you have only one repository, the dialog will already be populated. Otherwise select the Fedora Subversion repository URL. Typically you will check out the `Head` Revision and `Check out All projects`. Open the `Advanced` drop down and select `Resolve Workspace Projects` and `Separate projects` for modules. This will create a separate Eclipse project for each Maven pom which you may find useful later.

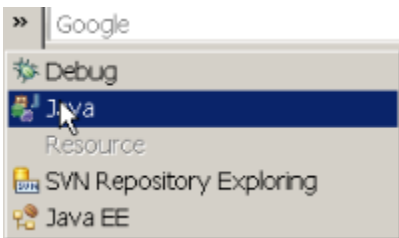


H) Select the `Next>` button to display the `Select Project Location` dialog. It is a personal preference where the workspace will be created on your workstation. If you have other project in the same workspace you may want to set up a working set just for the Fedora Repository (e.g. `fcrepo`) though you can to this later too. Select the `Finish` button and the wizard will complete its operations. This may take some time because the wizard must create the Eclipse project(s), download the sources, and download the Maven dependencies to your local repository. If this process seems to hang you may be having problems with the configuration of your local Maven repository. If this happens you may be able to kill Eclipse, correct the problem and use the `Update Maven Dependencies` feature of M2Eclipse. Otherwise, delete the project and start again using the `Check out as Maven Project...` wizard.

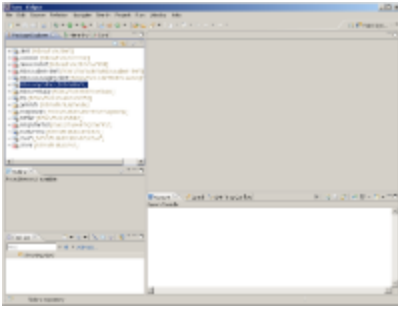


Set up the Maven Run As Configuration and Build the Project

I) Switch to the Java perspective to set up the run configurations.



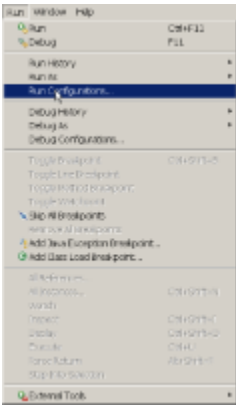
J) You will notice a number of projects, one for each Maven pom. Using this approach lets you make Eclipse settings for each module. The name of the top-level project is automatically set to `fcrepo` and M2Eclipse will have already make key Eclipse project settings for you.



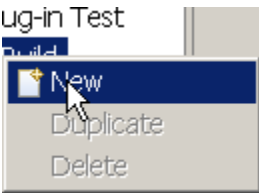
K) There are two ways to create a new Run Configuration from the Java perspective. You can right click on the `fcrepo` project to start the new Run Configuration dialog.



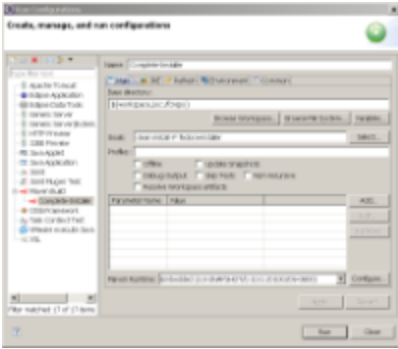
L) Alternately, you can use the Eclipse Menu: Run -> Run Configurations ...



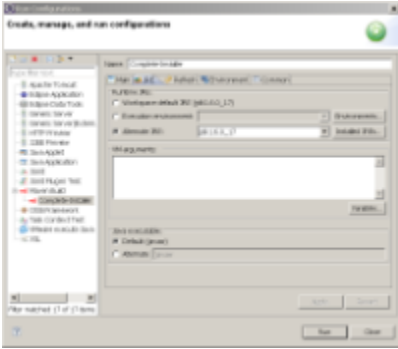
M) Start the Maven Run Configuration dialog using a right click on Maven Build and selecting the new menu item.



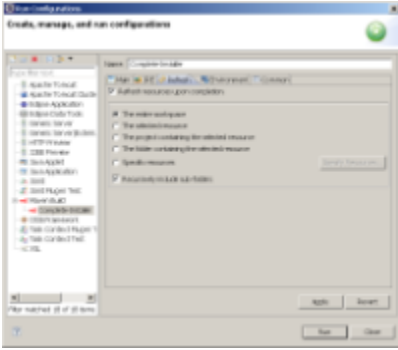
N) You will see the Run Configuration dialog. Using a Maven run configuration is exactly the same as running a Maven build from the command line. Later we will show how to use more of the Eclipse build features but for the first build lets use a basic run configuration. Give the configuration a name, set the base directory and the goals which build the entire repository. Press the `Apply` button to save the configuration (notice the name will be added to the list of Maven run configurations in the left panel of the dialog).



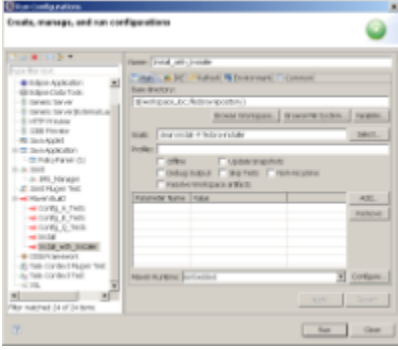
O) Check that the JRE is set to the Sun Java 1.6 initially. After you have a successful build you may also want to build and test under other JREs. Press the `Apply` button to save the configuration and press the `Run` button to start the build.



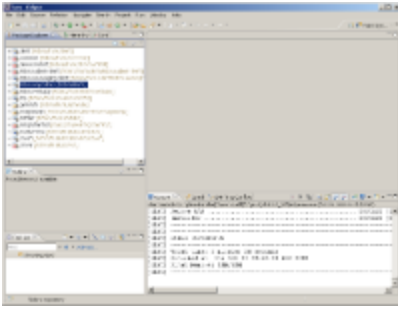
P) Since Fedora uses generated code and Maven will update the project structure it is helpful to refresh the project display after each build. This may not be fully sufficient so you also may need to update select Maven configurations after a build (see [Update Maven Projects](#))



Q) You can add additional Maven run configurations for individual modules, other goals and integration tests. Note for the integration tests you will need a compatibly configured running Fedora Repository.



R) Congratulations! You have succeeded in your first Maven build of Fedora.



S) Update Maven Configuration

Since Fedora uses generated code and Maven will update the project structure it is sometime necessary to manually update selected projects helpful. After the first build and if there are any changes to the common or generated code right click on the `common` and `fedora-admin-client` projects. Use the Maven -> Update Maven Configuration menu item to refresh the project.



M2Eclipse has a number of options for deploying projects but its handling of hierarchical projects is not complete. Since Fedora is a hierarchical project, it takes a little experience but Eclipse will work properly if permitted. Using the default M2Eclipse configuration is best. It presents subproject POMs as ordinary appropriate (usually Java) Eclipse projects in the workspace separately for each POM. If you use this configuration, the Eclipse functionality will work well. However, you will see both the hierarchical POM projects and the Java Projects in the project explorer views. If you look at the source directly you will see a normal directory hierarchy. M2Eclipse will use information from the POM to create functional Eclipse projects (setting up the `.project` and `.classpath` files). You can build equally using Eclipse Maven configuration or from the command line.

8. Customize the Workspace Settings

It is helpful if you customize the workspace settings. The coding standards settings are agreed to by the committers. Contributors are asked to use of the same settings if possible to help facilitate the acceptance process. You will have customize the workspace setting each time you create a new workspace or use the `Copy Workspace Setting` feature once you have your first workspace set up. Many of the settings can be loaded from files in the source distribution. To set up your workspace (code style settings, etc):

- See resources (Fedora post 3.3) `<fedora-repository>/resources/doc/devdocs/devenv/README.txt`

The next video shows how to prepare the Eclipse IDE for development of the Fedora Repository source code.



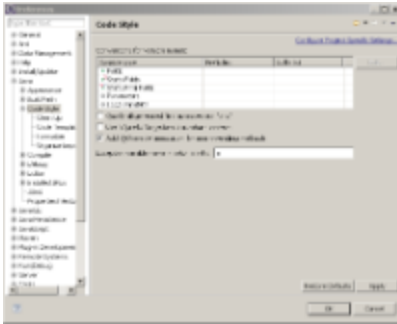
This video was prepared using Europa but generally will work for Eclipse Galileo and Ganymede also. There are small differences in the IDE versions and their dialogs. These instructions and screen images below were prepared with Ganymede and we are adding Galileo.

To set up Eclipse (Helios, Galileo and Ganymede):

It is recommended that you use to Eclipse Helios or Galileo (this section has been tested for Helios and Galileo - some small differences will be encountered). You must install Eclipse and a compatible Java JDK prior to beginning this configuration. Also you need to obtain a set of configuration files which are located in the Fedora Repository source.

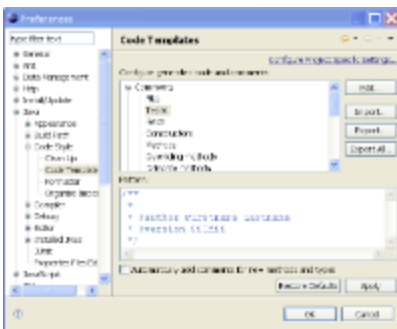
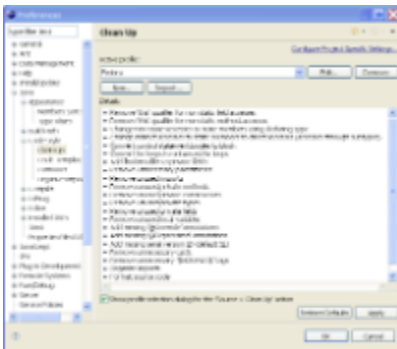
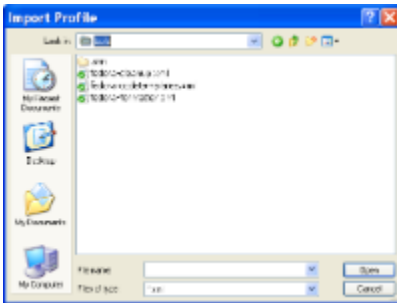
A) Use the Eclipse Menu: Window -> Preferences -> Java -> Code Style

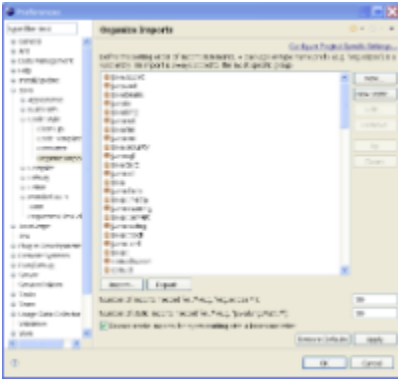
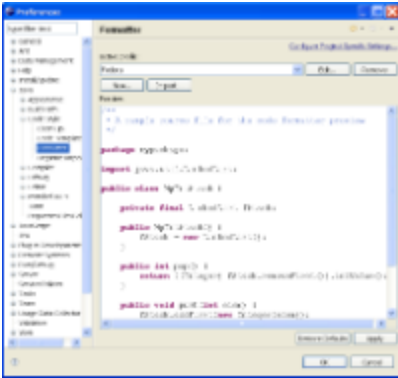
Under the top-level `Code Style` section:



1. Check "Add @Override ...", and leave other checkboxes unchecked
2. Make sure "Exception variable name" is e

B) Import the configurations files for the Clean Up, Code Templates, Formatter and Organize Imports from the build directory in the Fedora Repository source.

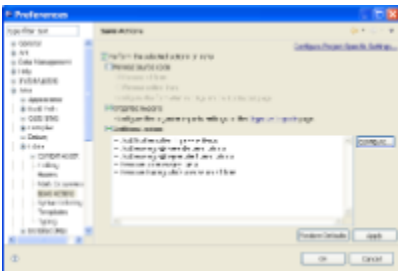




1. Under the Clean-up section import fedora-cleanup.xml and press Apply
2. Under the Code Templates section import fedora-codetemplates.xml and press Apply
3. Also under the Code Templates section in Comments -> Types edit Types to replace "Firstname Lastname" with your own name and press Apply
4. Under the Formatter section import fedora-formatter.xml and press Apply
5. Under the Organize Imports section import fedora.importorder and press Apply

C) Use the Eclipse Menu: Window -> Preferences -> Java -> Editor -> Save Actions

1. Select "Perform the selected actions on save"
2. Make sure "Additional Actions" is selected
3. Click "Configure...", and go to the "Code Organizing" tab, and make sure "Remove trailing whitespace" and "All lines" are selected



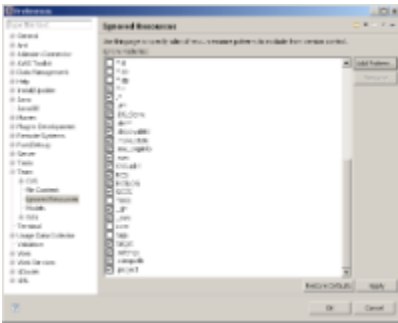
Optional Settings

Fedora makes use of WSDL generated Java classes. Without these, Eclipse will complain loudly. There are problems with refreshing in Eclipse that make generated code a source of some difficulties. Also, the Eclipse builder provides substantial feedback such as errors and warnings that

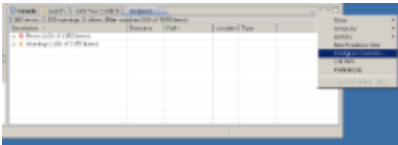
can aid in coding but can also be overwhelming.

A) Use the Eclipse Menu: Window -> Preferences -> Team -> Ignored Resources

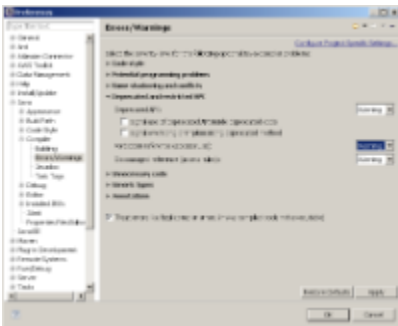
Add target, .settings, .classpath and .project and press the Apply button.



B) In the Problems view add a filter for warnings and errors. Use the Configure Contents menu item. Press the New button to create a filter configuration. Provide a name for the filter. Uncheck WSDL Problem, XML Problem, XML Schema Problem and XSL Problem in the Types subpanel. If you created a working set you may want to apply the filter to it. Otherwise select and appropriate scope. Press the Ok button to create the filter.



C) Use the Eclipse Menu: Window -> Preferences -> Java -> Compiler -> Errors/Warnings and open the Deprecated and Restricted API dropdown. Set the Forbidden reference (access rules) combo box to Warning. Press the Apply or Ok button to save the settings. This may start a full rebuild.



Installing from the Command-Line

1. Make sure you have a fresh checkout from Subversion.
2. If you have not previously built do it now.
3. After a successful build create the installer (or use the `-P installer` in the build configuration).
4. Copy the installer from the `<workspace>/fedora-repository/installer/target` to an install directory.
5. Run the installer to create an initial deployment (see the [Installation and Configuration Guide](#)). You should use a different directory from

- your Eclipse workspace for now. You may choose any configuration but use the **Quick** install until you have more experience.
- Start and stop the server once to ensure the configuration is correct and the `/path/to/tomcat/webapps/fedora` directory is created. This will permit the installer to set up the initial configuration which is difficult to do manually.

Known Issues

- Eclipse creates META-INF/MANIFEST.MF files
 - <https://issues.sonatype.org/browse/MNGECLIPSE-671>
 - https://bugs.eclipse.org/bugs/show_bug.cgi?id=285510
 - <http://www.nabble.com/New-files-added-to-src-tree-on-Maven-import-td24772549.html>
- New or renamed modules are not updated in Eclipse project layout
 - <http://www.nabble.com/New-modules-and-SCM-headaches.-td22167868.html>

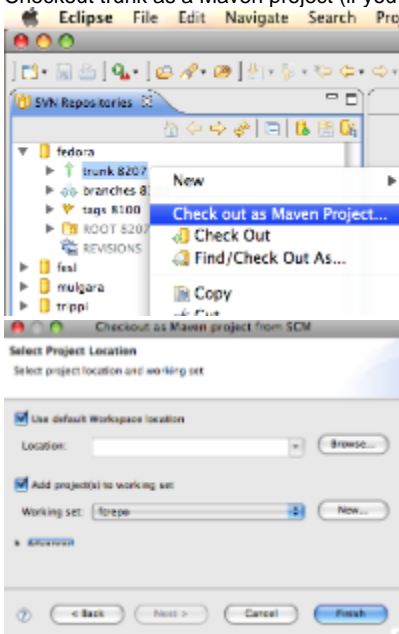
Eclipse with Maven notes

These are the steps I followed to get Fedora building correctly in Eclipse -- Eddie (6 Nov 2009)

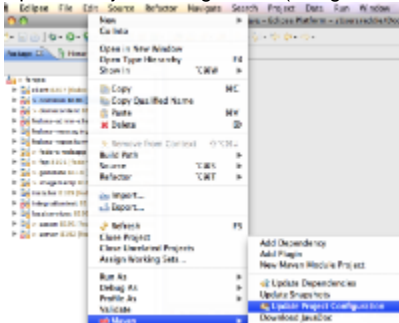
Tested with:

- Eclipse 3.4 (Ganymede), Carbon 32-bit & Eclipse 3.5 (Galileo), Cocoa 64-bit
- m2eclipse
- Subversive
 - SVNKit 1.3
 - Subversive Integration for the M2Eclipse Project (from <http://www.polarion.org/projects/subversive/download/integrations/update-site/>)
 - I also needed to update SVN Team Provider using <http://download.eclipse.org/technology/subversive/0.7/update-site/> rather than the one provided by the update site in Galileo

- Add the fedora svn repository to Eclipse (<https://fedora-commons.svn.sourceforge.net/svnroot/fedora-commons/fedora>)
- Optional: create a new working set for Fedora (e.g. fcrepo)
- Checkout trunk as a Maven project (if you created a working set, check out into the working set)

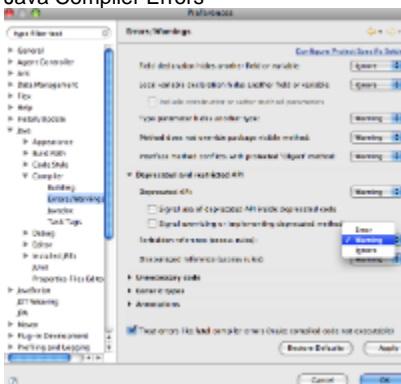


- Update Project Configuration (using the Maven menu) for common and fedora-admin-client projects

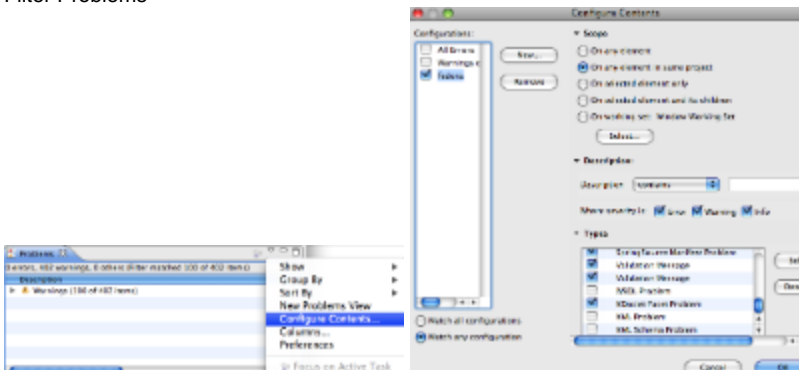


- Add target, .settings, .classpath, .project to Ignored Resources (Preferences/Team)

6. Java Compiler Errors



7. Filter Problems



Known Issues

1. Eclipse creates META-INF/MANIFEST.MF files
 - a. <https://issues.sonatype.org/browse/MNGECLIPSE-671>
 - b. https://bugs.eclipse.org/bugs/show_bug.cgi?id=285510
 - c. <http://www.nabble.com/New-files-added-to-src-tree-on-Maven-import-t24772549.html>
2. New or renamed modules are not updated in Eclipse project layout
 - a. <http://www.nabble.com/New-modules-and-SCM-headaches.-td22167868.html>

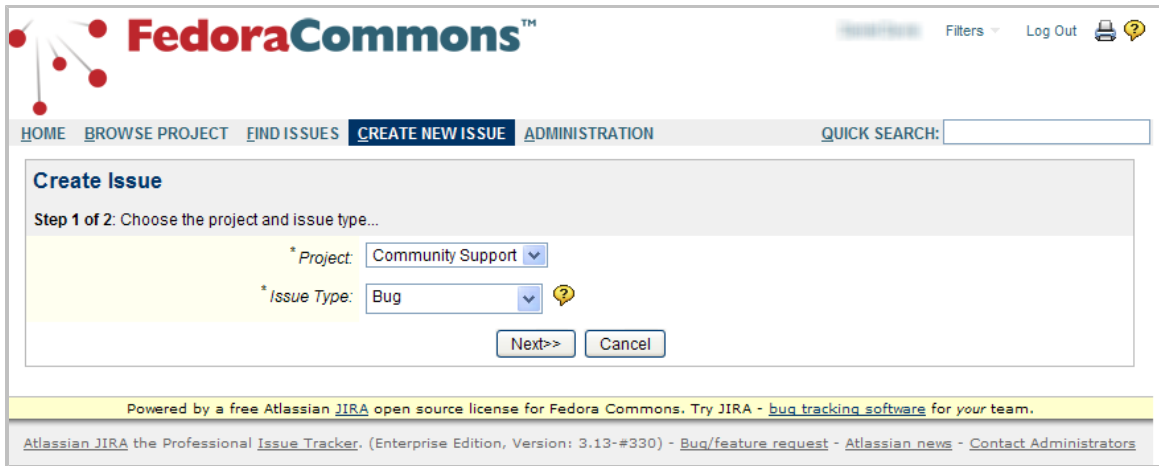
Where are Fedora's bugs reported?

Where are Fedora's bugs reported?

There are two methods to report bugs in Fedora Commons software.

First, you may report them on the support Sourceforge.net mailing lists `fedora-commons-users` or `fedora-commons-developer`. Often it is hard to determine whether the problem you are facing is a software defect (a.k.a a bug) or that you are learning how a feature works. It is a good idea to use these lists as a starting point. Usually it is best to start with the `fedora-commons-users` mailing list since your posting will receive the widest audience. If the consensus is that you are facing a defect in the software please provide clues to characterize the bug. If you are a software developer, please feel free to use either mailing list but postings on the developers list are generally expected to be better characterized for the development community.

Second, when you are fairly certain you have found a bug, please provide a Bug Report in the [Fedora Commons Jira tracker](#). You will need to create an account to be able to make the entry. It is important that you fill the form out completely since the more clues we have the more likely we are to be able to find and fix the bug. Please be sure to tell us which Fedora Commons software you are using (e.g. Fedora Repository, ProAI, GSearch) and which **version**. Even though our software uses Java it helps to know the operating environment too.



Where is the source code ? How can it be accessed ?

The Fedora Commons source code for the current version of the Fedora Repository can be downloaded using the download link under "downloads" on the [main page](#) for the current version.

The source is hosted on [github](#).

Checking out the code

If you wish to work with the source code, you will need to install [git](#).

You can clone (create your own local git repository) of the Fedora source using

```
git clone git@github.com:fcrepo/fcrepo.git
```

or

```
git clone https://github.com/fcrepo/fcrepo.git
```

The main development line of Fedora is the "master" branch

```
cd fcrepo
git checkout master
```

Downloading the source without using git

If you want to get a copy of the latest source code without using git you can download a tarball of the source code.

To download the current main development line ("master" branch):

```
curl -L -o fcrepo-source-master.tar.gz https://github.com/fcrepo/fcrepo/tarball/master
```

To download a branch (substituting in the appropriate branch-name, eg "maintenance-3.4" for the current 3.4 maintenance branch)

```
fcrepo-source-branch-name.tar.gz https://github.com/fcrepo/fcrepo/tarball/branch-name
```

Note that if you are intending to work with the source code, rather than just view the source or build the Fedora Repository from it, you will need to get a copy using git as above.

Installation FAQ

- How do I get Fedora running under the JBoss Application Server?
- How do I uninstall Fedora Commons ?
- Which Linux Distribution is best for Fedora Commons?

How do I get Fedora running under the JBoss Application Server?



Under Construction

This page is in the process of being updated.

How do I get Fedora Repository running under the JBoss Application Server?

Fedora WAR file will not deploy to the JBoss Application Server without modification. The root cause is a duplicate copy of Log4J.jar that is included in the Fedora distribution (JBoss AS includes its own copy). Symptoms of the problem are errors similar to these in the JBoss AS log file:

```
INFO [FEDINFO:TomcatDeployer] deploy, ctxPath=/fedora,
warUrl=../tmp/deploy/tmp42671fedora-exp.war/
ERROR [FEDINFO:STDERR] log4j:ERROR A "org.jboss.logging.util.OnlyOnceErrorHandler"
object is not assignable to a "org.apache.log4j.spi.ErrorHandler" variable.
ERROR [FEDINFO:STDERR] log4j:ERROR The class "org.apache.log4j.spi.ErrorHandler" was loaded by
ERROR [FEDINFO:STDERR] log4j:ERROR \[WebappClassLoader
delegate: false
repositories:
/WEB-INF/classes/
\-----> Parent Classloader: java.net.FactoryURLClassLoader@148bd9e\] whereas object of type
ERROR [FEDINFO:STDERR] log4j:ERROR "org.jboss.logging.util.OnlyOnceErrorHandler" was loaded by
[FEDINFO:org.jboss.system.server.NoAnnotationURLClassLoader@c2a132].
```

This is similar to the circumstances reported by a Java Forum user <http://forum.java.sun.com/thread.jspa?threadID=5112910>

How do I uninstall Fedora Commons ?

When you installed Fedora, you'll have specified a value for FEDORA_HOME - you will need to delete this folder.

If you chose the installation option to use the supplied Tomcat, you won't need to do anything further.

If you used an existing Tomcat, you will need to delete the Fedora web application. In your Tomcat home folder, go to webapps and delete both the war files **and** directories that Fedora installed:

- fedora
- fedora-demo
- fop
- imagemanip
- saxon

Depending on the installation options you chose, not all of these may have been installed.

If you specified using an external SQL database, you will also need to delete the database and tables that Fedora uses - these will be whatever you used for the instructions [here] <http://www.fedora-commons.org/confluence/display/FCR30/Installation+and+Configuration+Guide#InstallationandConfigurationGuide-database>].

Which Linux Distribution is best for Fedora Commons?

Fedora Commons is written in the Java programming language and is therefore platform independent. Thus it should run on the vast majority of operation systems - provided the requirements for the installation are met. Given that, a sensible strategy could be to opt for the distro you're most familiar with because that will be the distro you're mostly likely to maintain, keep secure and obtain dependencies in a way you're used to.

Usage FAQ

- "Policy blocked datastream resolution" error while adding datastream
- How can I implement collections in Fedora Commons?
- How can I retrieve the audit datastream ?
- How do I customize the HTML views produced by the REST API?
- How do I set my own PIDs (at runtime) for Fedora objects?
- Mime type for SDep Output
- Must I provide DC, RELS-EXT and RELS-INT as Inline XML Metadata?

- Resource Index date comparison with ITQL
- Where is the list of Dublin Core and FOXML properties?

"Policy blocked datastream resolution" error while adding datastream

Question

I have created a bare object in Fedora and I am trying to add in a dataStream via the REST API. However, I get the following error (in fedora.log) even though it looks like the datastream was added (which it was not). This is fedora 3.3.

```
INFO 2010-02-07 16:06:35.417 http-8080-4 (DefaultManagement) Completed addDatastream(pid: eureka:22, dsID: Source, altIDs: , dsLabel: Source file, versionable: true, MIMEType: application/pdf, formatURI: null, dsLocation: file:///Users/test.pdf, controlGroup: M, dsState: A, checksumType: null, checksum: null, logMessage: Added source)
ERROR 2010-02-07 16:06:35.418 http-8080-4 (DatastreamResource) Unexpected error fulfilling REST API request
fedora.server.errors.HttpServiceNotFoundException: DefaultExternalContentManager returned an error. The underlying error was a fedora.server.errors.HttpServiceNotFoundException
The message was "Policy blocked datastream resolution" .
```

```
at fedora.server.storage.DefaultExternalContentManager.getExternalContent(DefaultExternalContentManager.java:145)
at fedora.server.storage.DefaultDOManager.doCommit(DefaultDOManager.java:1198)
at fedora.server.storage.SimpleDOWriter.commit(SimpleDOWriter.java:498)
at fedora.server.management.DefaultManagement.addDatastream(DefaultManagement.java:527)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
...
```

I've been trying to find the right place where the policy is defined but cannot locate it.

Answer

The policy which is responsible for blocking the resolution of this datastream URI is located in deny-unallowed-file-resolution.xml (see fedora.fcgi for the repository policies directory).

The file also contains an example rule how to selectively enable file:// URIs. If you remove the comments and adapt the regex to your file's location the addDatastream operation should work.

The "Completed addDatastream(..)" log message you are referring to is probably a bit misleading in this context. What it intends to say is that the code block was executed successfully independent of the operation's outcome (there's an issue for that: <http://fedora-commons.org/jira/browse/FCREPO-629>).

How can I implement collections in Fedora Commons?

Question:

1. How do we implement collections in Fedora Commons?
2. Can we have xacml policies pertaining to a collection?
3. And also I want to know if these objects can be searched based on collection?

Answer:

In Fedora, you can establish relationships using a RELS-EXT datastream. See [this page](#) for more information about relationships. A collection object is just another fedora object, which has the rdf element like this example:

```
<rdf:Description rdf:about="info:fedora/abc:1"> <!-- (ie the pid of the collection object) -->
  <rel:isCollection>true</rel:isCollection>
</rdf:Description>
```

which says it is a collection.

A member of this collection would have a RELS-EXT datastream with the element

```
<rdf:Description rdf:about="info:fedora/abc:2"> <!-- (ie the pid of the member object) -->
  <rel:isMemberOf rdf:resource="info:fedora/abc:1" />
</rdf:Description>
```

This could equally apply to a child collection of the parent collection, in which case it would also include the element

```
<rel:isCollection>true</rel:isCollection>
```

Here is the full RELS-EXT datastream element for the parent collection above.

```

<foxml:datastream CONTROL_GROUP="X" ID="RELS-EXT" STATE="A" VERSIONABLE="true">
  <foxml:datastreamVersion CREATED="2009-01-12T14:23:20.112Z" ID="RELS-EXT.0" LABEL="Metadata"
  MIMETYPE="text/xml" SIZE="271">
    <foxml:contentDigest DIGEST="none" TYPE="DISABLED"/>
    <foxml:xmlContent>
      <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rel="info:fedora/fedora-system:def/relations-external#">
        <rdf:Description rdf:about="info:fedora/abc:1">
          <rel:isCollection>true</rel:isCollection>
        </rdf:Description>
      </rdf:RDF>
    </foxml:xmlContent>
  </foxml:datastreamVersion>
</foxml:datastream>

```

The xacml policies can then applied to the collection objects.

All relationships are indexed by the resource index provided it is turned on. You can query the resource index for all relationship based data. See [this page](#) for more information about the resource index.

How can I retrieve the audit datastream ?

As of version 3.3 it is not yet possible to directly retrieve the audit datastream. Therefore, in order to see the audit datastream, you'll need to export the xml of the object. For example: <http://localhost:8080/fedora/objects/changeme:1/objectXML>. It will appear in the export as a datastream called "AUDIT".

In the future, we hope this will be simpler. See <https://fedora-commons.org/jira/browse/FCREPO-635>

How do I customize the HTML views produced by the REST API?

When using a web browser, the REST API provides HTML views of objects, datastreams, methods and their profiles which can be used as a basic tool to navigate through the repository. (XML output is also available by using the format parameter, or through content negotiation).

These views are generated using XSLT stylesheets, which are stored in the /server/access and /server/management directories, and can therefore be customised by editing the XSLT.

The XML input to these views is the XML output of the corresponding [REST API methods](#).

How do I set my own PIDs (at runtime) for Fedora objects?

There is one configuration parameter in `$FEDORA_HOME/server/config/fedora.fcfg` that controls PID assignment:

pidNamespace – The default namespace for objects that don't specify a PID. A unique PID in this namespace will automatically be assigned by Fedora as needed. If the PID is specified, any valid namespace can be chosen. Allowed patterns for namespaces are:

- can be from 1 to 17 characters
- may only contain A-Z, a-z, 0-9, '.', or '-' (dash)



All ingested objects' specified PIDs will be retained and the only time a PID will be auto-generated is if the ingested object specifies no PID at all.

Mime type for SDep Output

Question

I am interested in generating an SVG file from an XML data file using an XSL transform however, the mime type is not set correctly ('text/html') when it is sent to the browser. (I am using FCR 3.3 on OSX 10.6 with Apache 2.2 and viewing on Firefox 3.6)

In the sDep I set the wsdlMsgTOMIME

```

<fmm:MethodReturnType wsdlMsgName="xslt_response" wsdlMsgTOMIME="image/svg+xml"/>

```

and the WSDL output

```
<wsdl:output>
  <mime:content type="image/svg+xml"/>
</wsdl:output>
```

Do I need to set the mime-type anywhere else? Are there on certain output mime-types that are recognized?

Answer

I suppose you are using the Saxon xslt service? Then you have to set the media-type attribute on the output element of your stylesheet. That will overrule all the other declarations.

```
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes" media-type="image/svg+xml"/>
  ....
```

Must I provide DC, RELS-EXT and RELS-INT as Inline XML Metadata?

Since Fedora 3.4, the DC, RELS-EXT and RELS-INT datastreams may be provided either as Inline XML Metadata or as Managed Content datastreams.



Ensure DC, RELS-EXT and RELS-INT are versionable if using Managed Content

Due to an outstanding bug [FCREPO-849](#), if you use Managed Content for DC, RELS-EXT or RELS-INT then please make sure these datastreams are versionable (the default setting for versionable is "true", so if you haven't specified this datastream property then you are safe).

If no DC datastream is provided as part of an ingest, Fedora creates a minimal DC datastream specifying the identifier and label of the object. The Control Group ("X" or "M") used when Fedora creates this datastream is specified in the `DOManager` section of `fedora.fcfg` using the `defaultDCControlGroup` parameter.

The RELS-EXT and RELS-INT datastreams may be managed using the relationships methods of the API (`addRelationship`, `purgeRelationship`). If there is no existing RELS-EXT or RELS-INT datastream when a new relationship is added, Fedora creates the datastream using the control group specified in the `DOManager` section of `fedora.fcfg` using the `defaultRELSControlGroup` parameter.

Resource Index date comparison with ITQL

Question

I want to know how to query the resource index against the date. What is the predicate I can use in the ITQL for date comparison ?

Answer

To do this, you'll need to query against a datatyping model. The full documentation for this is here: <http://docs.mulgara.org/itqloperations/datatypingmodels.html#o265>

Fedora does maintain a `<#xsd>` datatyping model, so in your example below, you will need to use something like:

```
select $object
from <#ri>
where $object <fedora-model:hasModel> <info:fedora/MY-CModel>
and $object <fedora-view:lastModifiedDate> $modified
and $ modified <mulgara:after> '2010-02-22T12:54:59.265Z'^^<xml-schema:dateTime> in <#xsd>
```

Note: Mulgara does not parse time zones for `dateTime`.

Where is the list of Dublin Core and FOXML properties?

Return to FCKB:The Official Fedora Commons FAQ



Page being updated

This page is in the process of being updated.

Where is the list of Dublin Core and FOXML properties?

The Dublin Core XML namespace,

```
http://purl.org/dc/elements/1.1/
```

, and the Fedora XML namespace,

```
info:fedora/fedora-system:def/foxml#
```

, are used to derive three RDF namespaces:

a)

```
http://purl.org/dc/elements/1.1/
```

b)

```
info:fedora/fedora-system:def/model#
```

c)

```
info:fedora/fedora-system:def/view#
```

There is a fourth RDF namespace,

```
http://www.w3.org/1999/02/22-rdf-syntax-ns#
```

, for which there is a single property, type, that records for each digitalObject, the type (*FedoraObject*, *FedoraBMechObject*, *FedoraBDefObject*) from the namespace

```
info:fedora/fedora-system:def/model#
```

The RDF properties for each of these namespaces for each Fedora digitalObject are asserted and retracted during the operation of Fedora. While these properties must never be asserted via the RELS-EXT datastream, they may be used by designers of Fedora applications in itql or rdql queries.

From

```
http://purl.org/dc/elements/1.1/
```

:
title
creator
subject
description
publisher
contributor
date
type
format
identifier
source
language
relation
coverage
rights
From

info:fedora/fedora-system:def/model#

:
contentModel
createdDate
definesMethod
implementsBDef
label
owner
state
usesBMech
Active
Deleted
Inactive
FedoraBDefObject
FedoraBMechObject
FedoraObject
From

info:fedora/fedora-system:def/view#

:
disseminates
disseminationType
isDirect
isVolatile
lastModifiedDate
mimeType

[Return to FCKB:The Official Fedora Commons FAQ](#)