Journaling

Unable to render {include} The included page could not be found.

Table of Contents

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

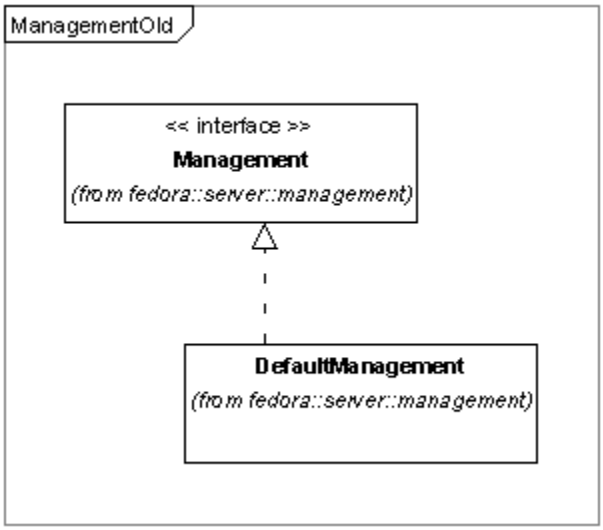
A Decorator on the Management Module

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:

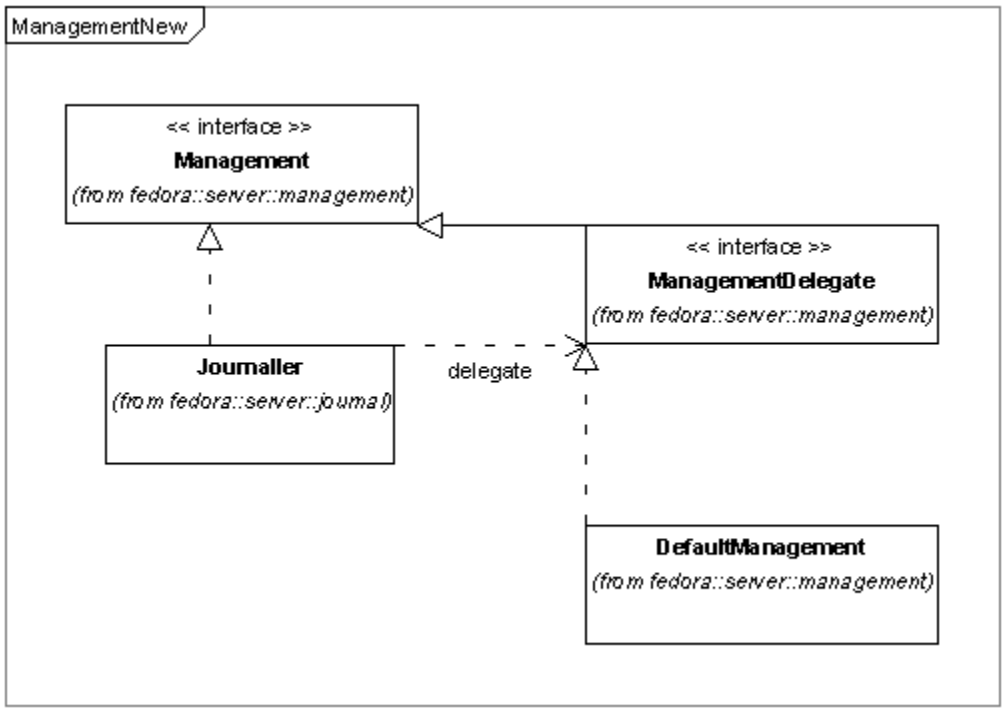


Configuring the Management module within fedora.fcfg:

```
<module role="org.fcrepo.server.management.Management"
  class="org.fcrepo.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="org.fcrepo.server.management.Management"
  class="org.fcrepo.server.journal.Journaler">
</module>
<module role="org.fcrepo.server.management.ManagementDelegate"
  class="org.fcrepo.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.fcfg`.

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 fcrepo.server.journal.JournalWriter. Required for normal mode. There is no default. Note that this component may require additional parameters.
  - fcrepo.server.journal.readerwriter.multifile.MultiFileJournalWriter
    Writes a collection of Journal files to a specified directory on disk. Best class for most operating scenarios.
  - fcrepo.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 fcrepo.server.journal.JournalReader. Required for recover mode. There is no default. Note that this component may require additional parameters.
  - fcrepo.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.
  - fcrepo.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.
  - fcrepo.server.journal.readerwriter.multifile.MultiFileFollowingJournalReader
    Same as LockingFollowingJournalReader, but does not recognize the "lockRequestedFilename" and "lockAcceptedFilename" options.
  - fcrepo.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 fcrepo.server.journal.recoverylog.JournalRecoveryLog. Required for recover mode. There is no default. Note that this component may require additional parameters.

    - fcrepo.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.
    - fcrepo.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.
    - fcrepo.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 fcrepo.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 `fcrepo.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 processs 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:

- fcrepo.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.
- fcrepo.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="org.fcrepo.server.management.Management" class="org.fcrepo.server.journal.Journaler">
  <param name="journalFileSizeLimit" value="100M"/>
  <param name="journalFileAgeLimit" value="1H"/>
  <param name="journalWriterClassname"
         value="org.fcrepo.server.journal.readerwriter.multicast.MulticastJournalWriter"/>
  <param name="transport.local.classname"
         value="org.fcrepo.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="org.fcrepo.server.journal.readerwriter.multicast.rmi.RmiTransport"/>
  <param name="transport.follow1.crucial" value="false"/>
  <param name="transport.follow1.hostName" value="follow1.nsdl.org"/>
  <param name="transport.follow1.service" value="RmiJournalReceiver"/>
  <param name="transport.follow2.classname"
         value="org.fcrepo.server.journal.readerwriter.multicast.rmi.RmiTransport"/>
  <param name="transport.follow2.crucial" value="false"/>
  <param name="transport.follow2.hostName" value="follow2.nsdl.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="org.fcrepo.server.management.Management"
     class="org.fcrepo.server.journal.Journaler">
  <param name="journalDirectory"
     value="/usr/local/ndr-content/journals/journalFiles"/>
  <param name="journalWriterClassname"
     value="org.fcrepo.server.journal.readerwriter.multifile.MultiFileJournalWriter"/>
  <param name="journalFileSizeLimit" value="100M" />
  <param name="journalFileAgeLimit" value="1H" />
</module>
<module role="org.fcrepo.server.management.ManagementDelegate"
     class="org.fcrepo.server.management.ManagementModule">
 <!-- ManagementModule configuration params go here -->
</module>
```

Configuring for recover mode:

```
<module role="org.fcrepo.server.management.Management"
        class="org.fcrepo.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="org.fcrepo.server.journal.readerwriter.multifile.MultiFileJournalReader"/>
  <param name="journalRecoveryLogClassname"
         value="org.fcrepo.server.journal.recoverylog.RenamingJournalRecoveryLog"/>
  <param name="recoveryLogFilename"
         value="/usr/local/ndr-content/journals/journal.recovery.log"/>
</module>
<module role="org.fcrepo.server.management.ManagementDelegate"
        class="org.fcrepo.server.management.ManagementModule">
 <!-- ManagementModule configuration params go here -->
</module>
```

Configuring for recover mode in a "following" server:

```
<module role="org.fcrepo.server.management.Management"
        class="org.fcrepo.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="org.fcrepo.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="org.fcrepo.server.journal.recoverylog.RenamingJournalRecoveryLog"/>
  <param name="recoveryLogFilename"
         value="/usr/local/ndr-content/journals/journal.recovery.log"/>
  <param name="followPollingInterval" value="10"/>
</module>
<module role="org.fcrepo.server.management.ManagementDelegate"
        class="org.fcrepo.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 gonnna 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:

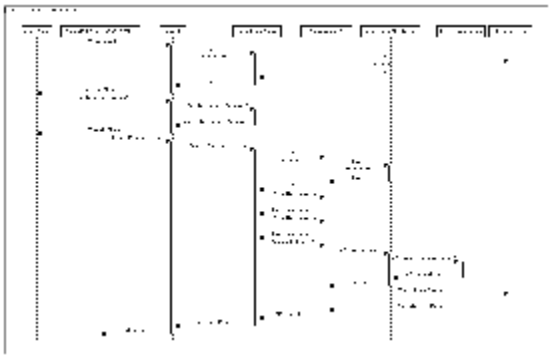
```
2006-06-15T09:45:36.605-0400: Event: method='getNextPid',
            file='C:\fedoraJournal20060615.134531.152Z',
            entry='2006-06-15T09:45:30.167-0400'
    context=fcrepo.server.journal.entry.JournalEntryContext
        environmentAttributes
            urn:fedora:names:fedora:2.1:environment:httpRequest:authType
                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
```

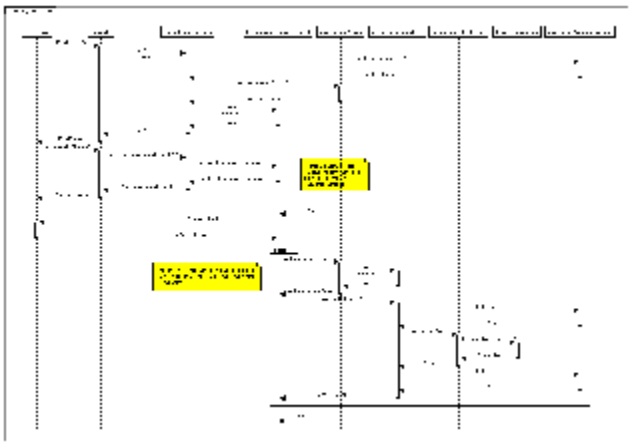
Sequence Diagrams

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).



Unable to render {include} The included page could not be found.