Table of Contents

About this Document
Technologies and Standards
Java
J2EE
XML
SOAP
Scala
Tomcat
MySQL
Hibernate
Heritrix
NutchWAX
Wayback
ARC
SRB
Jargon
METS
DRP
Jetty
Restlet
Ruby

System Overview
Architecture
CDL Common Framework
CDL Common Framework Shared Services
Component Registries
Event Logger
Security Manager
Transaction Auditor
Encryption Service

Restlet

Functional Overview
Collection Service
Collection Sync Daemon
org.cdlib.framework.dataManagement.CollectionManager_NutchQueue
Method initialize
Method startSynchThread
org.cdlib.framework.collection.CollectionSync
org.cdlib.framework.capture.CaptureDescriptionManager_HIBERNATE
Method query
org.cdlib.framework.collection.CollectionSync.ManifestCollection
Method listObjects
Method listRecords
org.cdlib.framework.collection.CollectionClient
Method request

AddObject
org.cdlib.framework.collection.CollectionReceiver
Method doPost
org.cdlib.framework.collection.HandlerAddObject
Method handle
org.cdlib.framework.security.SecurityManager_MYSQL
Method checkUser
org.cdlib.framework.dataManagement.CollectionManager_NutchQueue
Method addObject
Method processQueue
Method extractSegment
org.cdlib.framework.dataModel.WorkQueue
org.cdlib.framework.dataManagement.RepoExtractManager_MYSQL
Method getComponent
org.cdlib.framework.archivalStorage.StorageManager_SR
Method getInputStream

AddRecord
org.cdlib.framework.collection.CollectionReceiver
org.cdlib.framework.security.SecurityManager_MYSQL
org.cdlib.framework.dataManagement.RepoExtractManager_MYSQL
org.cdlib.framework.archivalStorage.StorageManager_SRBD
org.cdlib.framework.collection.HandlerAddRecord
Method handle
Method createArchiveFile
WorkQueue
org.cdlib.framework.dataManagement.CollectionManager_NutchQueue
RemoveObject
org.cdlib.framework.collection.CollectionReceiver
org.cdlib.framework.collection.HandlerRemoveObject
Method handle

Wayback Service
org.cdlib.framework.wayback.ReplayServlet
org.cdlib.framework.wayback.NutchResourceIndex
Method query
org.cdlib.framework.wayback.ResourceStoreManager
Method retrieveResource
org.cdlib.framework.wayback.JSReplayRenderer
Method renderResource
org.cdlib.framework.wayback.ResourceStore_PRESERVE
Method retrieveResource

NutchWAX Service
org.cdlib.framework.textIndex.webindex.OpenSearchServlet_CF
org.cdlib.framework.textIndex.webindex.WebIndex
Method getNutchBean
org.apache.nutch.searcher.NutchBean
Method search
org.cdlib.framework.ingest.AIPValidator_WARCIndex

Crawler Service
Adding a Job
org.cdlib.crawler.localCrawler.CrawlJobListResource
Method handlePost
org.cdlib.crawler.heritrix.Heritrix
Method addJob
Method getIdleCrawler
org.cdlib.crawler.heritrix.LocalCrawler
Method addJob
Method start
Method startCrawling
org.cdlib.crawler.heritrix.MBeanWrapper
Method invoke
Method invokeNoArgs

Querying a Job's Status
org.cdlib.crawler.localCrawler.CrawlJobFinder
Method createResource
org.cdlib.crawler.heritrix.Heritrix
Method getJobByUid
org.cdlib.crawler.CrawlJobResource
Method handleGet
org.cdlib.crawler.views.CrawlJobMetsView
Method toXML

Modifying a Job
org.cdlib.crawler.localCrawler.CrawlJobFinder
org.cdlib.crawler.heritrix.Heritrix
org.cdlib.crawler.CrawlJobResource
Method handlePut
org.cdlib.crawler.heritrix.LiveLocalCrawlJob
Method resume
Method pause
org.cdlib.crawler.heritrix.LocalCrawler
Method stop

Deleting a Job
org.cdlib.crawler.localCrawler.CrawlJobFinder
org.cdlib.crawler.heritrix.Heritrix
org.cdlib.crawler.CrawlJobResource
Method handleDelete
org.cdlib.crawler.heritrix.LiveLocalCrawlJob
Method delete

Querying the Crawler Status
org.cdlib.crawler.localCrawler.CrawlerListResource
Method handleGet
org.cdlib.crawler.heritrix.Heritrix
Method getCrawlers
Method getInstances

Query the Status of a Crawler Instance
org.cdlib.crawler.localCrawler.LocalCrawlerFinder
About this Document

This document provides a technical overview of Web Archiving Service (WAS) Web Services Version 1.0. WAS Web Services is a component of the WAS application, which is designed to capture, curate and preserve Internet content. The development of WAS is part of the California Digital Library's (CDL's) Web at Risk project, which is funded by a grant from the National Digital Information Infrastructure Program (NDIIPP).

WAS Web Services relies on the Digital Preservation Repository (DPR) to preserve captured content. DPR is outside the scope of this document. For more information on the DPR visit http://www.cdlib.org/inside/projects/preservation/dpr/.

Technologies and Standards

WAS Web Services is built on the following technologies and standards.

Java

Java is an object-oriented programming language. For all services except the Crawler Service, WAS requires the Java Runtime Environment (JRE) version 1.5 or greater. For the Crawler Service JRE version 1.6 or greater is required. For more information about Java visit http://java.sun.com/.

J2EE

J2EE is a Java specification for enterprise functionality such as Web services and servlets. For more information about J2EE visit http://java.sun.com/javaee/index.jsp.

XML

Information about Extensible Markup Language (XML) can be found at http://www.w3.org/XML/.

SOAP

The Simple Object Access Protocol (SOAP) is a lightweight XML-based protocol for exchanging structured information in a decentralized, distributed environment. For more information about SOAP visit http://www.w3.org/TR/soap/.

Scala

Scala is a general purpose programming language designed to express common programming patterns in a concise, elegant, and type-safe way. Scala is used by the Crawler Service. For more information about Scala visit http://www.scala-lang.org/.

Tomcat

Tomcat is an open source Web container for executing Java servlets, a component of J2EE technology. WAS Web Services uses Tomcat version 5.5.23. For more information about Tomcat visit http://tomcat.apache.org/.

MySQL

MySQL is an open source relational database. WAS Web Services uses MySQL version 4.1.12. For more information about MySQL visit http://www.mysql.com/.

Hibernate

Hibernate is an object to relational database mapping and persistence service written in Java. For more information about Hibernate visit http://www.hibernate.org/.

Heritrix

Heritrix is an open source Web crawler. For more information about Heritrix visit http://crawler.archive.org/.

NutchWAX

Nutch with Web Archiving Extensions (NutchWAX) is an open source search and indexing engine for Web archive collections. For more information about NutchWAX visit http://archive-access.sourceforge.net/projects/nutch/.

Wayback

Wayback is a Java Web application that allows Web browsers to access content stored in ARC files. For more information about Wayback visit http://archive-access.sourceforge.net/projects/wayback/.

ARC

ARC files are compressed archives of captured Web content. Content captured by Heritrix is stored in the ARC file format. The ARC file format is similar to the commonly used ZIP file format.

SRB

Storage Resource Broker (SRB) is software that manages preserved content in archival storage. For more information about SRB visit http://www.sdsc.edu/srb/index.php/Main_Page.
Jargon


METS

The Metadata Encoding and Transmission Standard (METS) is an XML schema for encoding descriptive, administrative and structural metadata associated with objects in a digital library. For more information about METS visit [http://www.loc.gov/standards/mets/mets-home.html](http://www.loc.gov/standards/mets/mets-home.html).

DPR

The Digital Preservation Repository (DPR) is an application developed by CDL to manage the long-term preservation of digital objects. WAS Web Services uses DPR to preserve captured Web content. For more information on DPR visit [http://www.cdlib.org/inside/projects/preservation/dpr/](http://www.cdlib.org/inside/projects/preservation/dpr/) or refer to the document: "UC Libraries Digital Preservation Repository System Design Guide".

Jetty

Jetty is an open source Web container. It is embedded in the Crawler Service. For more information about Jetty visit [http://www.mortbay.org/](http://www.mortbay.org/).

Restlet

Restlet is a Java framework built on top of J2EE servlets that follows the REST architectural style. For more information about Restlet visit [http://www.restlet.org/](http://www.restlet.org/).

JMEX


Ruby


System Overview

WAS Web Services is a set of related applications that provide an Application Programing Interface (API) for curating Web content. Each service offers a logically grouped set of functionality that can be invoked by SOAP-based clients. There are four services that comprise WAS Web Services.

Table 1. WAS Web Services

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Service Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>Manages search indexes of collections built from Web crawls.</td>
</tr>
<tr>
<td>Wayback</td>
<td>Displays crawled Web content stored in ARC files.</td>
</tr>
<tr>
<td>NutchWAX</td>
<td>Searches and indexes crawled Web content stored in ARC files.</td>
</tr>
<tr>
<td>Crawler</td>
<td>Uses the Heritrix Web crawler to schedule and manages crawl jobs.</td>
</tr>
</tbody>
</table>

In addition, two stand-alone applications work with WAS Web Services to trigger important events such as batch crawls and ingests.

Table 2. WAS Web Services Stand-Alone Applications

<table>
<thead>
<tr>
<th>Application Name</th>
<th>Application Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>crawl_lifecycle_manager</td>
<td>Periodically checks the database for new crawl or ingest requests. Schedules crawls by calling the Crawler Service. Schedules ingests of completed crawls by calling the DPR's Feeder Service.</td>
</tr>
<tr>
<td>scheduler</td>
<td>Periodically checks the database to determine if any regularly scheduled batch crawls should be run.</td>
</tr>
</tbody>
</table>

WAS Web Services is built on top of DPR Web Services. DPR Web Services preserves crawled content and related metadata, authenticates users, and provides other preservation related functionality. An overview of the WAS system, with WAS Web Services highlighted in light blue, is given in Figure 1, "WAS Web Services System Overview".

Figure 1. WAS Web Services System Overview
Architecture

The architecture of WAS Web Services is based on Representational State Transfer (REST) and CDL's Common Framework (CF). The Collection Service is built on the CF. The Crawler Service is built on REST. The Wayback and NutchWAX Services are not based on either of these architectures. Wayback and NutchWAX may be refactored in the future to use the CF or REST architectures.

CDL Common Framework

The CF is an open, services-oriented architecture written in Java. The architecture follows the Reference Model for an Open Archival Information System (OAIS) published by the Consultative Committee for Space Data Systems (CCSDS). The fundamental principles of the CF are:

- Clear separation of applications from underlying services
- Easy reuse of underlying services in multiple applications
- Simple, consistent design patterns and interfaces for services
- Consistent exposure of services through SOAP and Java Client API
- Clear separation of services from underlying data storage and other resources
- Easy integration of local and third-party solutions to specific problems through a plug-in approach
- Scalability and flexibility
- Platform independence

All CF services are exposed using SOAP with attachments over HTTP.

The implementation of a service is any Java class extending `org.cdlib.framework.service.RESTReceiver` or `org.cdlib.framework.service.SOAPService`. `RESTReceiver` and `SOAPService` are Java servlets customized to accept specific types of requests. The `SOAPService` servlet handles HTTP requests containing a SOAP message. The `RESTReceiver` servlet handles HTTP GET requests that pass data through the Query String or path of the request's URI.

Service classes support logical groups of functionality, such as Ingest or Full Text Indexing. The HTTP request sent to a service specifies the type of functionality it wants to use. For example, to request the addition of content to the repository, the Ingest Service is passed an HTTP request containing the header value "AddVersion". Each functionality of a service is mapped to a Handler class. A Handler is a class implementing the `org.cdlib.framework.service.Handler` interface. The implementing class defines functionality-specific processing logic needed to complete the request. This logic is specified by implementing the single method of the `Handler` interface called `handle`. 
Handlers often call a lower layer of code known as the Manager layer. This layer is composed of Managers, which are classes that encapsulate functionality for accessing external resources such as databases, Web services, or other Enterprise Information Systems (EIS). Managers are also used as wrappers for third party libraries. Managers ensure that services are not tied to specific data sources or vendor implementations. For example, the Ingest Service handlers call an interface named StorageManager when reading or writing archived files. Since DPR uses SRB to manage archival storage, a manager named org.cdlib.framework.archivalStorage.StorageManager_SRB exists that implements the StorageManager interface. This manager implementation exposes the low level SRB client called Jargon through the generic API of StorageManager. If, in the future, different archival storage software is used, a class implementing the StorageManager interface can be created that encapsulates the functionality of the new software. This class can then be "plugged" into the DPR using configuration files.

Services are deployed as individual Web Archive Module (WAR) files. Each WAR runs in a separate Web Container instance.

There is no "state" kept by the CF between requests; the system is stateless.

Figure 2 shows the "Example" CF service at startup.

Figure 2. Example CF Service at Startup

![Figure 2. Example CF Service at Startup](image)

Figure 3 shows the "Example" CF service handling a request.

Figure 3. Example CF Service Handling a Request

![Figure 3. Example CF Service Handling a Request](image)
CDL Common Framework Shared Services

The CF provides shared services that handle infrastructure-related tasks, such as authentication and component registration. These services are used by the functional services implemented on top of the CF. The services provided include:

- **Component registries** that enable new functions or protocol handlers to be added dynamically to the system
- An **event logger** that centralizes the recording of errors and informational messages
- A **security manager** that authenticates requests for service functionality
- **Auditing functionality** that provides a historical record of each critical operation and its outcome
- An **encryption utility** that ensures sensitive information is stored in a secure format

**Component Registries**

Component registries provide a means for identifying components to the system and for the system to locate components and execute their methods in the appropriate context. For example, a registry of Submission Information Package (SIP) parsers and their associated handlers is used by the Ingest Service. This allows the Ingest Service to choose a handler based on the format of a SIP stream (for example, METS).

Component registries are implemented by the Common Framework class `org.cdlib.framework.utility.Framework`. Component registries are configured by Resource files, which are loaded during application startup by the `Framework` class. An example from the `IngestService.properties` file illustrates how a METS SIP parser is registered with its handler.

**Example 1. METS SIP Parser in Component Registry**

```java
class.SIPParser,METS=org.cdlib.framework.ingest.SIPParser_METS
```

**Event Logger**

The event logger enables functional components to send informational or error messages to a centrally maintained collection of event logs. The event logger is implemented by the class `org.cdlib.framework.utility.FileLogger`. 
Security Manager

The security manager authenticates users by checking the validity of their credentials in the user registry. The user registry is implemented using MySQL and is accessed through the class `org.cdlib.framework.security.SecurityManager_MYSQL`.

Transaction Auditor

The transaction auditor stores the details of each service request. Storing transaction details aids in the auditing and debugging of CF services. The transaction auditor is implemented by the class `org.cdlib.framework.dataManagement.TransactionManager_MYSQL`.

Encryption Service

Sensitive information, such as user passwords, must be stored in a secure format. The MD5 message digest algorithm is used by the CF. Encryption functionality is implemented by the class `org.cdlib.framework.security.SecurityUtil`.

Restlet

Representational State Transfer (REST) is a style of software architecture used for building distributed applications. The REST architecture separates an application into uniquely addressable resources. Resources address each other through a uniform interface and communication between resources is stateless. These broad principles define the architecture of RESTful applications such as the World Wide Web (WWW).

Restlet is a Java framework for implementing RESTful Web applications. The framework maps REST concepts, such as Resources and Representations, directly to Java classes, such as `org.restlet.resource.Resource` and `org.restlet.resource.Representation` respectively. A Restlet application can be run by a Java Web container, such as Tomcat, or by the Noelios Restlet Engine (NRE). The NRE is a reference implementation of the Restlet API.

Functional Overview

This section describes the functionality of each application and service of WAS Web Services. For each functionality, a processing flow diagram shows the route of execution, at a high level, through the application. Relevant classes and methods are defined. The list of classes and methods is not exhaustive. Instead, it is meant to highlight a limited number of critical sub-processes.

Collection Service

Collections are thematically-related sets of content selected by WAS users from one or more completed crawl jobs. The Collection Service updates the NutchWAX search indexes built from collection content. Updates to the indexes are made as content is added to and deleted from collections. The Collection Service is driven by a daemon process that continuously checks for collection modifications. When modifications are detected the daemon process sends requests to the Collection Service API through the `CollectionClient` class to update the appropriate NutchWAX search index.

Collection Sync Daemon

The Collection Sync Daemon is a background thread that is initialized during Collection Service startup.

Figure 4. Collection Sync Call Stack

This class updates the NutchWAX Search Index with the searchable text of a completed crawl. It retrieves the crawl from SRB based on the objectID passed to the Collection Service. This class also spawns two background threads. The first thread reads a work queue of pending requests to update the search indexes of a collection. The second thread is a daemon thread that checks the database every hour for modifications that have been made to collections. If a modification is detected, the thread uses the `CollectionClient` to send an AddObject, AddRecord, or RemoveObject request to the Collection Service.
Method initialize

Signature:

```java
public void initialize(Framework fw)
```

This method initializes the Manager. It is called once on service startup.

Method startSynchThread

Signature:

```java
private Thread startSyncThread()
```

This method creates a daemon thread to detect modifications to collections.

org.cdlib.framework.collection.CollectionSync

This class detects modifications to collections and updates their search indexes accordingly. The class runs as a background thread. It becomes active once an hour.

org.cdlib.framework.capture.CaptureDescriptionManager_HIBERNATE

This class executes SQL queries against the MySQL database using Hibernate.

Method query

Signature:

```java
public List query(String q)
```

This method executes a query against the MySQL database and returns the results.

org.cdlib.framework.collection.CollectionSync.ManifestCollection

This class wraps a `ManifestCollection.txt` file, which contains a list of collections and their associated objects and records. The `ManifestCollection.txt` file stores the state of collections once every hour. Before the `ManifestCollection.txt` file is updated, the collections in the `ManifestCollection.txt` file are compared to the collections in the database. The database holds the most up-to-date version of the collections. By comparing the current state of collections against their state an hour previous, changes to collections are detected. These changes can then be pushed to the collection search index.

Method listObjects

Signature:

```java
private Set<ManifestCollectionObject> listObjects()
```

This method retrieves all objects (crawl jobs) associated with existing collections.

Method listRecords

Signature:

```java
private Set<ManifestCollectionObject> listRecords()
```

This method retrieves all records (files) associated with existing collections. Records are files that have been individually associated with a collection, not the files contained in the collection's crawls.

org.cdlib.framework.collection.CollectionClient

This class generates SOAP over HTTP requests for Collection Service functionality.

Method request

Signature:

```java
public ResultPackage request(RequestPackage requestPackage)
```

This method generates a SOAP over HTTP request for Collection Service functionality.

AddObject

The AddObject request adds a completed crawl to a collection's search index. The crawl is indexed by the NutchWAX indexer.

Figure 5. Collection AddObject Call Stack
org.cdlib.framework.collection.CollectionReceiver

This class receives HTTP SOAP requests for Collection Service functionality.

Method doPost

Signature:

```java
protected void doPost(HttpServletRequest req, HttpServletResponse res)
```

This method receives HTTP SOAP requests for Collection Service functionality.

org.cdlib.framework.collection.HandlerAddObject

This class adds completed crawls to a collection.

Method handle

Signature:

```java
public CollectionIndexResult handle(CollectionIndexRequest collectionIndexRequest)
```

This method processes an AddObject request. It calls the `CollectionManager_NutchQueue` class to retrieve and index a completed crawl.

org.cdlib.framework.security.SecurityManager_MYSQL

This class authenticates users of WAS Web Services. User credentials are stored in a MySQL database.

Method checkUser

Signature:

```java
public boolean checkUser(String userID, String password)
```

This method authenticates a user's credentials, which are submitted in the form of a username and password.

org.cdlib.framework.dataManagement.CollectionManager_NutchQueue

See `org.cdlib.framework.dataManagement.CollectionManager_NutchQueue`

Method addObject

Signature:
public void addObject(String indexID, String indexProfile, ObjectID objectID, boolean deleteFirst)

This method places an AddObject request into a shared WorkQueue object for processing by a background thread.

Method processQueue

Signature:

public void processQueue()

This method is invoked by a background thread. It reads and executes requests for Collection functionality that have been queued in a shared WorkQueue object.

Method extractSegment

Signature:

protected synchronized File extractSegment(ObjectID objectID, String indexID)

This method adds the text of an object to a NutchWAX search index.

org.cdlib.framework.dataModel.WorkQueue

This class is a thread-safe data queue.

org.cdlib.framework.dataManagement.RepoExtractManager_MYSQL

This class builds a digital object by reading data from various storage locations. The Collection Service uses this class to retrieve ingested ARC files for indexing by NutchWAX.

Method getComponent

Signature:

public RepoExtractResult getComponent(ObjectID objectID, String componentID)

This method builds and returns a digital object by reading data from various storage locations, such as the MySQL database and SRB-managed storage.

org.cdlib.framework.archivalStorage.StorageManager_SRB

This class reads and writes data from SRB-managed storage.

Method getInputStream

Signature:

public InputStream getInputStream(ObjectID objectID, int version, String componentID)

This method connects to SRB-managed storage and returns an InputStream object. The InputStream object is used to read files preserved in DPR.

AddRecord

The AddRecord request adds a file or group of files from a completed crawl to a collection's search index. The files are indexed by the NutchWAX indexer.

Figure 6. Collection AddRecord Call Stack
org.cdlib.framework.collection.CollectionReceiver

See org.cdlib.framework.collection.CollectionReceiver

org.cdlib.framework.security.SecurityManager_MYSQL

See org.cdlib.framework.security.SecurityManager_MYSQL

org.cdlib.framework.dataManagement.RepoExtractManager_MYSQL

See org.cdlib.framework.dataManagement.RepoExtractManager_MYSQL

org.cdlib.framework.archivalStorage.StorageManager_SRB

See org.cdlib.framework.archivalStorage.StorageManager_SRB

org.cdlib.framework.collection.HandlerAddRecord

This class adds files from completed crawls to a collection.

Method handle

Signature:

```java
public CollectionIndexResult handle(CollectionIndexRequest collectionIndexRequest)
```

This method processes an AddRecord request. It calls the org.cdlib.framework.dataManagement.RepoExtractManager_MYSQL class to retrieve ARC records from SRB-managed storage. The CollectionManager_HubQueue class is used to index the ARC records (files). Once a crawl has been retrieved from storage it is cached in memory to optimize performance.

Method createArchiveFile

Signature:

```java
private File createArchiveFile(IPComponent ipComponent)
```

This method creates a file object from the retrieved binary data of a completed crawl stored in SRB-managed storage.
RemoveObject

The RemoveObject request removes a file or completed crawl from a collection's search index.

Figure 7. Collection RemoveObject Call Stack

org.cdlib.framework.collection.CollectionReceiver

See org.cdlib.framework.collection.CollectionReceiver

org.cdlib.framework.collection.HandlerRemoveObject

This class removes files or completed crawls from a collection.

Method handle

Signature:

```
public CollectionIndexResult handle(CollectionIndexRequest collectionIndexRequest)
```

This method removes a file or completed crawl from the NutchWAX search index.

Wayback Service

The Wayback Service retrieves crawled content from ARC files using the Internet Archives open source Wayback software. The content is displayed by the WAS User Interface. The CF architecture is not used by the Wayback Service.

The Wayback Service has five components: a servlet, a resource search index, a resource manager, a resource renderer, and a resource store. These components are CDL implementations or extensions of Wayback base classes and interfaces, which allow different types of storage and search indexes to take advantage of Wayback functionality.

Figure 8. Wayback Service Call Stack
org.cdlib.framework.wayback.ReplayServlet

This servlet is an extension of the Internet Archive's \texttt{org.archive.wayback.core.WaybackServlet} servlet. The archived URL desired for display is passed to the \texttt{ReplayServlet}. The \texttt{ReplayServlet} uses the URL and the other \texttt{Wayback} components to retrieve the desired Web content from the appropriate ARC file.

org.cdlib.framework.wayback.NutchResourceIndex

This class encapsulates a NutchWAX search index built from ARC files. It implements the \texttt{org.archive.wayback.resourceIndex} interface, which allows different search indexes to be used with \texttt{Wayback}.

**Method query**

**Signature:**

\begin{verbatim}
   public SearchResults query(WaybackRequest wbRequest)
\end{verbatim}

This method calls the \texttt{Wayback QueryServlet} over HTTP to retrieve the location of a desired URL in its containing ARC file.

org.cdlib.framework.wayback.ResourceStoreManager

This class implements the \texttt{org.archive.wayback.resourceStore} interface. The \texttt{ResourceStore} interface allows ARC records to be retrieved from different types of storage. In this case, the storage is managed by \texttt{SRB}.

**Method retrieveResource**

**Signature:**

\begin{verbatim}
   public Resource retrieveResource(SearchResult result)
\end{verbatim}

This method accesses storage media to retrieve an ARC record. The ARC record is returned in the form of a \texttt{org.archive.wayback.core.Resource} object.

org.cdlib.framework.wayback.JSReplayRenderer

This class extends the \texttt{org.archive.wayback.replay.BaseReplayRenderer} class. The \texttt{org.cdlib.framework.wayback.ReplayServlet} uses it to render objects of type \texttt{org.archive.wayback.core.Resource} into different formats, such as HTML.

**Method renderResource**

**Signature:**

\begin{verbatim}
   public void renderResource(HttpServletRequest httpRequest, HttpServletResponse httpResponse, WaybackRequest wbRequest, SearchResult result)
\end{verbatim}

This method renders a \texttt{org.archive.wayback.core.Resource} object as HTML.

org.cdlib.framework.wayback.ResourceStore\_PRESERVE

This class uses the \texttt{org.cdlib.framework.dataManagement.RepoExtractManager\_MYSQL} class to access ARC files in SRB-managed storage. Once the ARC files are retrieved they are cached to optimize performance.

**Method retrieveResource**

**Signature:**

\begin{verbatim}
   public Resource retrieveResource(SearchResult result)
\end{verbatim}

This method retrieves ARC files from DPR and caches them in memory.

**NutchWAX Service**

The NutchWAX Service queries and indexes crawled content stored in ARC files. The CF architecture is not used by the NutchWAX Service.

**Figure 9. NutchWAX Service Call Stack**

\begin{figure}
\centering
\includegraphics[width=\textwidth]{image}
\caption{NutchWAX Service Call Stack}
\end{figure}

org.cdlib.framework.textIndex.webindex.OpenSearchServlet\_CF

This servlet exposes the functionality of the NutchWAX search module. It accepts an HTTP request with a search term, searches for content matching the term,
org.cdlib.framework.textIndex.webindex.WebIndex

This class retrieves a org.apache.nutch.searcher.NutchBean object.

Method getNutchBean

Signature:

```java
public NutchBean getNutchBean(Properties prop)
```

This method retrieves and caches a org.apache.nutch.searcher.NutchBean object.

org.apache.nutch.searcher.NutchBean

This class provides an interface for searching NutchWAX-indexed content. It is part of the NutchWAX codebase.

Method search

Signature:

```java
public Hits search(Query query, int numHits, int maxHitsPerDup, String dedupField, String sortField, boolean reverse)
```

This method searches a NutchWAX index for Web pages matching a query.

org.cdlib.framework.ingest.AIPValidator_WARCIndex

This class indexes ARC files prior to their ingest into the DPR. It is configured in the WasIngestService.properties file of the DPR’s Ingest Service. The NutchWAX index files are ingested into the DPR.

Crawler Service

The Crawler Service schedules and manages crawl jobs. Its functionality is exposed through the embedded Jetty Web Container. Crawls are performed by the Internet Archive’s Heritrix Web crawler. The Crawler Service is designed using the Restlet framework and is written in the Scala programming language. In order to run, the Crawler Service requires the Scala software distribution.

Adding a Job

Figure 10. Add Job Call Stack

org.cdlib.crawler.localCrawler.CrawlJobListResource

This class extends the Restlet Resource class. The Resource class represents a REST resource, which is a URI addressable source of information. The CrawlJobListResource adds a crawl job to an instance of a Web crawler.

Method handlePost

Signature:

```java
public void handlePost()
```

This method reads a crawl order, formatted as XML, and submits the order to a crawler group, which represents a pool of Web crawlers. In this case the crawler group is a pool of Heritrix crawlers.
org.cdlib.crawler.heritrix.Heritrix

This class implements the CrawlerGroup interface. The CrawlerGroup interface represents a pool of generic Web crawlers. The Heritrix class implements the CrawlerGroup as a pool of Heritrix Web crawlers.

Method addJob

Signature:

public Option addJob(CrawlJob job)

This method chooses an idle crawler from the crawler pool. The submitted job is then scheduled on the chosen crawler. If no crawlers in the pool are idle, an informational message is returned to the calling program.

Method getIdleCrawler

Signature:

public Option getIdleCrawler()

This method checks the status of the crawlers in the crawler pool. The first crawler that is not processing a crawl job is returned to the caller. If no crawlers are available an informational message is returned.

org.cdlib.crawler.heritrix.LocalCrawler

This class represents an instance of the Heritrix Web crawler.

Method addJob

Signature:

public Option addJob(CrawlJob job)

This method uses JMX to schedule a crawl job on an instance of Heritrix.

Method start

Signature:

public void start()

This method uses JMX to activate an idle crawler.

Method startCrawling

Signature:

public void startCrawling()

This method uses JMX to request that a crawler begin executing pending crawl jobs.

org.cdlib.crawler.heritrix.MBeanWrapper

This class wraps low level JMX client functionality. It is used to invoke JMX methods and set JMX attributes on a JMX-enabled crawler instance.

Method invoke

Signature:

public Object invoke(String method, Object args[])

This method executes a JMX method with parameters by calling invoke on a MBeanServerConnection.

Method invokeNoArgs

Signature:

public Object invokeNoArgs(String method)

This method executes a JMX method without parameters by calling invoke on a MBeanServerConnection.

Querying a Job's Status

Figure 11. Query Job Call Stack
org.cdlib.crawler.localCrawler.CrawlJobFinder

This class is a Restlet Finder. A Finder maps a request to an instance of the Restlet Resource class. The CrawlJobFinder maps requests to the CrawlJobResource class.

Method createResource

Signature:
public Resource createResource(Request request, Response response)

This method retrieves the crawl job associated with the UID passed in the request by making a JMX call to Heritrix. The crawl job along with other information in the request are used to create an instance of the CrawlJobResource class, which the method returns.

org.cdlib.crawler.heritrix.Heritrix

See org.cdlib.crawler.heritrix.Heritrix

Method getJobById

Signature:
public Option getJobById(String uid)

This method calls Heritrix using JMX to retrieve a list of active crawl jobs. The crawl job with the passed UID is extracted from the list of jobs and returned to the caller.

org.cdlib.crawler.CrawlJobResource

This class is a REST resource that returns a representation of a crawl job.

Method handleGet

Signature:
public void handleGet()

This method handles a GET HTTP request for the status of a crawl job. The crawl job is transformed into an XML representation and returned in the HTTP response.

org.cdlib.crawler.views.CrawlJobMetsView

This class extracts metadata from a crawl job and returns the metadata as METS-formatted XML.

org.cdlib.crawler.CrawlJobXmlView

This class returns an XML representation of a crawl job.

Method toXML

Signature:
public Node toXml(CrawlJob job)

This method reads the attributes of a crawl job and formats them as an XML document.

Method toXML

Signature:
public Option toXml(CrawlJob job, String text)

This method reads a specified attribute of a crawl job and returns it as an XML document. For example, the text "end-time" can be passed to the method in order to return an XML representation of the crawl job's ending timestamp.
Modifying a Job

A crawl job can be modified in three ways: it can be paused, resumed, or stopped.

Figure 12. Modify Job Call Stack

![Diagram showing the process of modifying a job]

```java
org.cdlib.crawler.localCrawler.CrawlJobFinder

See org.cdlib.crawler.localCrawler.CrawlJobFinder

org.cdlib.crawler.heritrix.Heritrix

See org.cdlib.crawler.heritrix.Heritrix

org.cdlib.crawler.CrawlJobResource

See org.cdlib.crawler.CrawlJobResource

Method handlePut

Signature:

```java
public void handlePut()
```

This method handles a PUT HTTP request to modify the state of a crawl job. The crawl job's state is changed by calling Heritrix through JMX.

org.cdlib.crawler.heritrix.LiveLocalCrawlJob

This class represents an active crawl job running on a Heritrix instance.

Method resume

Signature:

```java
public boolean resume()
```

This method invokes the "resume" JMX operation on the Heritrix crawler. The "resume" operation restarts a paused crawl job.

Method pause

Signature:

```java
public boolean pause()
```

This method invokes the "pause" JMX operation on the Heritrix crawler. The "pause" operation pauses a running crawl job.

org.cdlib.crawler.heritrix.LocalCrawler

See org.cdlib.crawler.heritrix.LocalCrawler

Method stop

Signature:

```java
public void stop()
```

This method uses JMX to stop the execution of a running job, if one exists.
Deleting a Job

Figure 13. Delete Job Call Stack

- `org.cdlib.crawler.localCrawler.CrawlJobFinder`
- `org.cdlib.crawler.heritrix.Heritrix`
- `org.cdlib.crawler.CrawlJobResource`

Method `handleDelete`

Signature:

```java
public void handleDelete()
```

This method handles a DELETE HTTP request to remove a crawl job. The crawl job is removed by calling `Heritrix` through JMX.

`org.cdlib.crawler.heritrix.LiveLocalCrawlJob`

Method `delete`

Signature:

```java
public boolean delete()
```

This method invokes the "deleteJob" JMX operation on the Heritrix crawler. The "deleteJob" operation removes the specified crawl job.

Querying the Crawler Status

Figure 14. Query Crawler Status Call Stack

- `org.cdlib.crawler.localCrawler.CrawlerListResource`

This class extends the `Restlet` resource class. The `Resource` class represents a REST resource, which is a URI addressable source of information. The `CrawlerListResource` returns a list of crawlers running in the Crawler Service.

Method `handleGet`

Signature:

```java
public void handleGet()
```
This method handles a GET HTTP request to return a list of crawler instances running in the crawler service. The list is formatted as an XML document and placed in the HTTP response.

```java
public Iterable get Crawlers()
```

This method returns a list of the Heritrix instances running in the Crawler Service.

```java
public Iterable getInstances(ObjectName name, Function1 f)
```

This method uses JMX to query Heritrix for a list of registered crawler instances.

### Query the Status of a Crawler Instance

**Figure 15. Query Crawler Instance Status Call Stack**

This class is a Restlet Finder. A Finder maps a request to an instance of the Restlet Resource class. The LocalCrawlerFinder maps requests to the LocalCrawlerResource class.

```java
public Resource createResource(Request request, Response response)
```

This method retrieves the crawler associated with a specified crawler name by making a JMX call to Heritrix. The crawler along with other information in the request are used to create an instance of the LocalCrawlerResource class, which the method returns.

```java
public Option getCrawlerByName(String name)
```

This method returns the Heritrix instance associated with the passed name.

```java
public Option getInstance(ObjectName name, Function1 f)
```

This method uses JMX to query Heritrix for a list of registered crawler instances. The crawlers in the list are matched against the name of the desired instance. The matching instance is extracted from the list and returned to the caller.

### Crawl Lifecycle Manager Application

The Crawl Lifecycle Manager (CLM) application is a Ruby program that runs as a daemon process. It usually runs on the same server that hosts WAS Web Services. The application file is named `crawl_lifecycle_manager`.

The CLM checks the `cap_job` database table every 30 seconds for two types of records. It checks for records that have the value "QUEUED" in their queue status field.
Records with the value "FINISHED" in their queuestatus field represent crawl jobs that have been requested but not scheduled. The CLM schedules each queued crawl by sending an HTTP request to the Crawler Service. After each job is scheduled, the database is updated to reflect the new status of the job. Running jobs are also updated so that the queuestatus field reflects the crawl job’s most recent status.

Records with the value "FINISHED" in their queuestatus field and a non-null value in their description field represent completed crawl jobs waiting for ingest. The CLM creates a Manifest file for each one of these completed crawls. The Manifest file contains the crawl metadata formatted as a METS XML document. The CLM then calls the DPR’s Feeder service, which ingests the crawl based on the information in the Manifest file. Finally, the CLM updates the database so the queuestatus field of each ingested crawl is set to "COMPLETE".

Scheduler Application

The Scheduler application is a Ruby program that runs as a daemon process. It usually runs on the same server that hosts WAS Web Services. The application file is named `scheduler`.

The Scheduler checks the `cap_specification` database table every ten minutes for records with non-null values in their scheduled_cron field. These records represent crawl specifications that have been scheduled to run regularly as batch jobs. For example, a WAS user can schedule a crawl specification to run every week at 5:00pm PST.

Once the Scheduler has selected the batch scheduled jobs, it evaluates each job to determine if the job should be run. Jobs that are scheduled to run are then queued by creating a new record in the `cap_job` database table. The queuestatus field of these new records is set to the value "QUEUED". Eventually the CLM will detect these records and start corresponding crawl jobs.

WAS Data Model

The WAS data model is based on the DPR data model. The WAS data model contains three types of data: crawl job information, user information, and digital object information. Figure 16, "WAS Data Model" shows the WAS data model. For a full-size image of the data model click here.

Figure 16. WAS Data Model