RIBAPI

Repository in a Box Application Programmer's Interface

Jeremy Millar[*], Paul McMahan[*], and Jack J. Dongarra[*]

Abstract: The Repository in a Box Application Programmer's Interface (RIBAPI) provides a means for third party applications to access data objects stored in metadata repositories created with Repository in a Box (RIB). RIBAPI allows developers to interoperate with RIB repositories and to leverage the information contained within them.

Keywords: digital libraries, interoperable repositories, RIB

This work was funded in part by the National Computational Science Alliance and by the Department of Defense (DoD) High Performance Computing Modernization Program (HPCMP).

[*] Department of Computer Science, University of Tennessee, Knoxville 37996-1301

Table of Contents

Introduction

Repository in a Box (RIB) provides a toolkit for building and maintaining metadata repositories. RIB was developed by the National HPCC Software Exchange (NHSE) Technical Team at the University of Tennessee, Knoxville. Initially, RIB only provided tools for the creation of software repositories. The recent release of RIB v2.0 allows RIB to create general metadata repositories. The creation of software metadata repositories remains the primary application of RIB.

RIB has two primary design goals: promotion of software reuse and interoperability.

RIB promotes software reuse by providing tools to build metadata repositories. These repositories contain information pertaining to software packages and routines; an abstract, licensing information, point of contact, etc. These repositories are intended to be discipline oriented and are meant to act as a central access point for software information.

The interoperability features of RIB allow repositories to share information in a scalable and efficient manner. Additionally, these features allow domain-specific repositories to be gathered into larger repositories with a common access point. For example, NHSE represents an aggregation of several other repositories. NHSE provides a common access point to these repositories – so far as the user is concerned, all the data is contained within NHSE.

Repositories created with RIB can only interoperate automatically with other repositories created with RIB, limiting the usefulness of the interoperability features. This document specifies an application programmer's interface (RIBAPI) to repositories created with RIB v2.0, allowing interoperation between RIB and other web-based repositories (e.g. PSES).

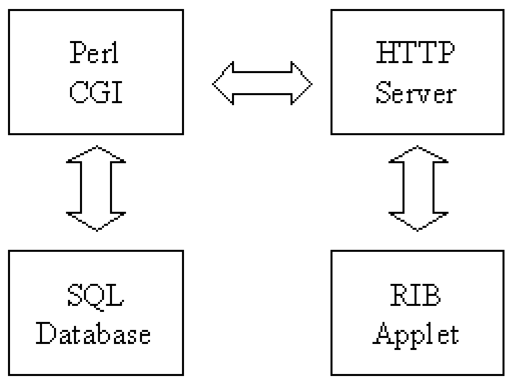
RIBAPI provides a set of methods for accessing the RIB backend. RIBAPI allows 3rd party developers to leverage the information and data stored within RIB repositories. Initial applications of RIBAPI include the coupling of pre-existing RIB repositories with the Globus MDS project and with NetSolve.

This specification consists of an overview of the RIB internals, and a discussion of the RIBAPI functions.

RIB Overview

A RIB repository consists of four major components: a java applet for administration, an HTTP server to facilitate communication, a set of CGI scripts for backend management (written in perl), and an SQL database backend. Figure 1 illustrates the component layout and dataflow of a RIB repository.

Figure 1. RIB Layout and Dataflow



Java Applet

RIB provides a java applet for repository administration. The applet runs in a web browser and provides the repository administrator with an intuitive, graphical interface. The administration applet allows the administrator to perform a variety of functions: creation and deletion of data objects, creation and deletion of repository interoperations, alteration of the repository data model, etc. RIBAPI provides many of these functions to 3rd party programmers.

The fact that the administration applet runs in a browser has several important consequences. Most importantly, the HTTP server inserts a level of indirection between the applet and the SQL database. In particular, changes made in the applet must be committed to the database through a call to a CGI script. Other consequences include flexibility; the administrator can perform his duties from any machine connected to the network.

HTTP Server

RIB provides an HTTP server to facilitate communication between the various repository components. Additionally, repository users contact this server to browse the repository contents. RIB uses the Apache HTTP server, version 1.3.6.

Because CGI scripts manage the RIB SQL database, all contact with the repository must be via HTTP. Consequently, any 3rd party applications will access the database in the same indirect manner as the administration applet. All RIBAPI function calls are made through the HTTP server; therefore, all RIBAPI implementations will need HTTP POST capability.

Perl CGI Scripts

RIB contains a suite of perl CGI scripts for database and repository management. These scripts are the heart of RIB, providing nearly all of its functionality. Capabilities provided by the CGI scripts include object creation and deletion, repository creation and deletion, and authentication and access control.

RIB uses the mod_perl extension to the Apache HTTP server to improve performance. Mod_perl provides a persistent embedded Perl interpreter to avoid the overhead associated with starting an external interpreter. Consequently, the start-up time for the RIB Perl scripts is significantly reduced.

The administration applet provides a graphical front-end to these scripts to the repository administrator. RIBAPI provides a programmatic front-end to 3rd party developers.

SQL Database

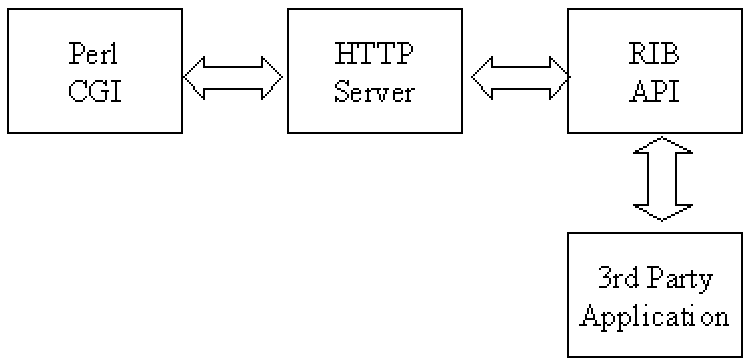
RIB stores data in a relational SQL database. RIB uses mySQL version 3.22 or later. Table formats follow a user-defined data model.

RIBAPI provides access to the SQL tables via the CGI scripts and should be used exclusively. Directly accessing the tables may destroy data integrity or produce undefined behavior.

RIBAPI

RIBAPI is an application programmer's interface (API) for Repository in a Box (RIB). It provides 3rd party developers with access to the data contained within RIB repositories. Figure 2 illustrates the dataflow of a 3rd party application accessing a RIB repository.

Figure 2. RIBAPI Dataflow



This section documents the functions contained in RIBAPI, their behaviors, input parameters, and return codes.

RIBAPI Functions

RIBAPI specifies a set of functions for accessing RIB repositories. As Figure 2 illustrates, a 3rd party application makes a RIBAPI function call to access the RIB CGI scripts.

Because RIBAPI accesses the RIB CGI scripts via HTTP, each RIBAPI function must perform an HTTP POST operation. The details of this operation are beyond the scope of this document.

RIBAPI includes the following functions:

- approveObjects
- changeContact
- changeName
- changePasswd
- createInterop
- createObject
- deleteInterops

- deleteObjects
- editConfig
- editObject
- listInteropObjects
- listInterops
- listObjects
- listRepositories
- object
- repository
- requireObjectApproval
- unapproveObjects

Function Descriptions

This section describes each RIBAPI function in detail. Function descriptions take the following format:

Function Name
Input Parameters
Return Value
Description

The following input parameters take integer values:

- rh (repository handle)
- oh (object handle)
- ih (interoperation handle)

All repository passwords are passed as clear text.

approveObjects

Input Parameters: rh (repository handle)
objectClass
oh*n* (object handle, where *n* is a sequence number)
repoPasswd (repository password)

Returns: ok

Description: approveObjects sets the approval status of each object specified by oh*n* in repository rh. Multiple objects may be specified by providing sequence numbers. For example, to approve objects 2 and 3, approveObjects must be called with the parameters oh1=2 and oh2=3.

changeContact

Input Parameters: rh (repository handle)
newContact (new contact email address)
repoPasswd (repository password)

Returns: ok

Description: changes the contact information for repository rh to newContact.

changeName

Input Parameters: rh (repository handle)
newName (new repository name)
repoPasswd (repository password)

Returns: ok

Description: changes the name of repository rh to newName.

changePasswd

Input Parameters: rh (repository handle)
newPasswd (new password)
repoPasswd (repository password)

Returns: ok

Description: changes the password of repository rh to newPasswd.

createInterop

Input Parameters: rh (repository handle)
url (location of repository to interoperate with)
updateInterval (update interval in seconds for interop)
interopName (name of interoperation)
repoPasswd (repository password)

Returns: ok

Description: creates a new interoperation for repository rh. The repository (and its location) to interoperate with is specified in url. updateInterval specifies how often the interop should be updated. interopName specifies the name of interop - this will appear in the Java administration applet.

createObject

Input Parameters: rh (repository handle)
newObject (XML document specifying object params)
repoPasswd (repository password)

Returns: URL

Description: creates a new object in repository rh. The object's attributes and relationships are specified in the XML document newObject. newObject should follow the RIBAPI DTD (e.g. <!DOCTYPE ribapi>).

deleteInterops

Input Parameters: rh (repository handle)
ih*n* (interoperation handle, where *n* is an sequence number)
repoPasswd (repository password)

Returns: ok

Description: deletes the interoperation ih*n* from repository rh. Multiple interoperations may be specified by providing sequence numbers. For example to delete interoperations 2 and 3, deleteInterops must be called with parameters ih1=2 and ih2=3.

deleteObjects

Input Parameters: rh (repository handle)
objectClass (object class)
oh*n* (object handle, where *n* is an sequence number)
repoPasswd (repository password)

Returns: ok

Description: deletes the object oh*n* from repository rh. Also deletes any references to oh*n*. Multiple objects may be specified by providing sequence numbers. For example, to delete objects 2 and 3, deleteObjects must be called with parameters oh1=2 and oh2=3.

editConfig

Input Parameters: rh (repository handle)
newConfig (XML document specifying new config)
repoPasswd (repository password)

Returns: ok

Description: changes data model of repository rh to that specified in newConfig. newConfig should follow the RIB configuration DTD (e.g. <!DOCTYPE ribconfig>).

editObject

Input Parameters: rh (repository handle)
newObject (XML document specifying the object)
oh (object handle)
repoPasswd (repository password)

Returns: URL

Description: replaces data contained in object oh with that contained in newObject. newObject should follow the RIBAPI DTD (e.g. <!DOCTYPE ribapi).

listInteropObjects

Input Parameters: rh (repository handle)
ih*n* (interoperation handle, where *n* is an optional sequence number)

Returns: XML document

Description: lists all objects provided by the interoperation ih to the repository rh. Also lists data about each interoperation object - name, url, owner's repository handle, and date last modified. Note that the object class is not provided, since interoperation objects must be of the repository's primary class. Multiple interoperations may be specified by providing sequence numbers. For instance, to list all objects provided by interoperations 1 and 2, listInteropObjects must be called with ih1=1 and ih2=2. The XML document returned by listInteropObjects will follow the RIBAPI DTD (e.g. <!DOCTYPE ribapi>).

listInterops

Input Parameters: rh (repository handle)

Returns: XML document

Description: lists all interoperations for the repository specified by rh. Also lists data about each interoperation - name, url, interoperation handle, last attempt, last success, update interval, checkpoint, and last failure. The XML document returned by listInterops will follow the RIBAPI DTD (e.g. <!DOCTYPE ribapi>).

listObjects

Input Parameters: rh (repository handle)
class (class name)

Returns: XML document

Description: lists all objects of type class contained within the repository specified by rh. Also lists data about each object - name, object handle, date last modified, approval status, date created, and URL. The XML document returned by listObjects will follow the RIBAPI DTD (e.g. <!DOCTYPE ribapi>).

listRepositories

Input Parameters: None

Returns: XML document

Description: lists all repositories managed by a particular installation of RIB. Also lists baseline data about each repository - name, contact, primary class, primary attribute, config file, and repository handle. The XML document returned by listRepositories will follow the RIBAPI DTD (e.g. <!DOCTYPE ribapi>).

object

Input Parameters: rh (repository handle)
class (object's class)
oh (object handle)
repoPasswd (repository password - optional)

Returns: XML document

Description: lists all attribute value pairs and relationships for a particular object. The repository password need only be supplied if object approval is enabled and the user wishes to view unapproved objects. The XML document returned by object will follow the RIBAPI DTD (e.g. <!DOCTYPE ribapi>).

repository

Input Parameters: rh (repository handle)

Returns: XML document

Description: lists object classes contained within the repository specified by rh. The XML document returned by repository will follow the RIBAPI DTD (e.g. <!DOCTYPE ribapi>).

requireObjectApproval

Input Parameters: rh (repository handle)
repoPasswd (repository password)
required (Boolean value)

Returns: ok

Description: sets the require object approval flag for the repository. If required = true, objects must be approved before they appear in the catalog. If required = false, all objects appear in the catalog.

unapproveObjects

Input Parameters: rh (repository handle)
objectClass (object class)
oh*n* (object handle, where *n* is an optional sequence number)
repoPasswd (repository password)

Returns: ok

Description: sets the approval status for each object oh*n* to unapproved. Multiple objects may be specified by providing sequence numbers. For example, to unapprove objects 2 and 3, unapproveObjects must be called with parameters oh1=2 and oh2=3.

RIBAPI Return Codes

Each RIBAPI function is guaranteed to return a string. The contents of the return string depend upon the particular function and the success or failure of the function call. On a failure, each function returns a string describing the error. On a success, the string may be any of the following: a URL, an XML document, or the string “ok”. Table 1 lists the RIBAPI functions by return code.

Table 1. RIBAPI Functions by Return Code

Return Code	RIBAPI Function
Error	All
ok	approveObjects
	changeContact
	changeName
	changePasswd
	createInterop
	deleteInterops
	deleteObjects
	editConfig
	requireObjectApproval
	unapproveObjects
XML Document	listInteropObjects
	listInterops
	listObjects
	listRepositories
	object
	repository
URL	createObject
	editObject

XML Document Structure

RIB makes use of XML for internal representation of data and for communication with RIBAPI. Application developers will encounter these XML documents in two places: as a return value from certain RIBAPI functions, and as input to the editConfig, createObject, and editObject functions. All XML documents returned by RIBAPI functions follow the same Document Type Definition (DTD). The RIB configuration files follow a different DTD; therefore, the XML document specified as input to editConfig differs from the documents returned by other functions.

RIB API DTD

The DTD used by RIBAPI functions returning XML documents is as follows:

```
<!DOCTYPE ribapi [
<!ELEMENT rib            (repository+)               >
<!ELEMENT repository     (class*)                    >
<!ELEMENT class          (object*)                   >
<!ELEMENT object         (attribute*,relationship*)  >
<!ELEMENT attribute      (#PCDATA)                   >
<!ELEMENT relationship   (#PCDATA)                   >
]>

<!ATTLIST rib
      version                     NMTOKEN   #REQUIRED>

<!ATTLIST repository
     name                         CDATA     #REQUIRED
     config                       CDATA     #IMPLIED
     handle                       NMTOKEN   #IMPLIED
     password                     CDATA     #IMPLIED
     contact                      CDATA     #IMPLIED
     primary_class                CDATA     #IMPLIED
     primary_attribute            CDATA     #IMPLIED
     join_enabled                 NMTOKEN   #IMPLIED
     whats_new_enabled            NMTOKEN   #IMPLIED
     object_approval_required     NMTOKEN   #IMPLIED
     is_locked                    NMTOKEN   #IMPLIED  >

<!ATTLIST class
     name                  CDATA     #REQUIRED
     objects               CDATA     #IMPLIED  >

<!ATTLIST object
     name                  CDATA     #REQUIRED
     handle                NMTOKEN   #IMPLIED
     last_modified         NMTOKEN   #IMPLIED
     approved              NMTOKEN   #IMPLIED
     created               NMTOKEN   #IMPLIED
     url                   CDATA     #IMPLIED  >

<!ATTLIST attribute
     name               CDATA     #REQUIRED >

<!ATTLIST relationship
     name               CDATA     #REQUIRED >
```

RIB Configuration DTD

The DTD specifying RIB configuration files is as follows:

```
<!DOCTYPE ribconfig [
<!ELEMENT rib            (class+)                        >
<!ELEMENT class          (attribute*,relationship*)      >
<!ELEMENT attribute      (#PCDATA | term)*               >
<!ELEMENT relationship   (#PCDATA)                 >
<!ELEMENT term           (term*)                   >
]>

<!ATTLIST rib
      version      NMTOKEN     #REQUIRED  >

<!ATTLIST class
      name          CDATA      #REQUIRED
      extends       CDATA      #REQUIRED  >

<!ATTLIST attribute
      name          CDATA      #REQUIRED
      cardinality   CDATA      #IMPLIED
      dtype         CDATA      #IMPLIED
      status        CDATA      #IMPLIED   >

<!ATTLIST relationship
      name          CDATA      #REQUIRED
      cardinality   CDATA      #IMPLIED
      status        CDATA      #IMPLIED   >

<!ATTLIST term
      value         CDATA      #REQUIRED  >
```

RIB DTD

The DTD combining the configuration and API DTDs is as follows:

```
<!DOCTYPE rib [
<!ELEMENT rib           (repository+|class+)                  >
<!ELEMENT repository      (class*)                          >
<!ELEMENT class          ((object*)|(attribute*,relationship*))  >
<!ELEMENT object          (attribute*,relationship*)            >
<!ELEMENT attribute       (#PCDATA|term)*                          >
<!ELEMENT relationship    (#PCDATA)                              >
```

```
<!ELEMENT term          (term*)                        >

<!ATTLIST rib
        version                CDATA     #IMPLIED  >

<!ATTLIST repository
        name                         CDATA       #REQUIRED
        config                       CDATA       #IMPLIED
        handle                       NMTOKEN     #IMPLIED
        password                     CDATA       #IMPLIED
        contact                      CDATA       #IMPLIED
        primary_class                CDATA       #IMPLIED
        primary_attribute            CDATA       #IMPLIED
        join_enabled                 NMTOKEN     #IMPLIED
        whats_new_enabled            NMTOKEN     #IMPLIED
        object_approval_required     NMTOKEN     #IMPLIED
        is_locked                    NMTOKEN     #IMPLIED  >

<!ATTLIST class
        name                         CDATA       #REQUIRED
        extends                      CDATA       #IMPLIED
        objects                      CDATA     #IMPLIED  >

<!ATTLIST object
        name                         CDATA       #REQUIRED
        handle                       NMTOKEN     #IMPLIED
        last_modified                NMTOKEN     #IMPLIED
        approved                     NMTOKEN     #IMPLIED
        created                      NMTOKEN     #IMPLIED
        url                          CDATA       #IMPLIED  >

<!ATTLIST attribute
        name                         CDATA      #REQUIRED
        cardinality                  CDATA      #IMPLIED
        dtype                        CDATA      #IMPLIED
        status                       CDATA      #IMPLIED   >

<!ATTLIST relationship
        name                         CDATA      #REQUIRED
        cardinality                  CDATA      #IMPLIED
        status                       CDATA      #IMPLIED   >

<!ATTLIST term
        value                        CDATA      #REQUIRED   >

]>
```

Examples

The following examples illustrate how to post to a RIB CGI script. Additionally, they illustrate how to gather information from a repository via several RIBAPI function calls.

<u>Example 1 – Posting to a CGI Script</u>

Implementations of RIBAPI will need to perform operations similar to this example in order to access the RIB CGI scripts.

connect port ###
POST /cgi-bin/listObjects.pl HTTP/1.0

rh=10&class=Asset

<u>Example 2 – Gathering Information</u>

This example illustrates how to use RIBAPI function calls to gather information about objects in a repository, and how to subsequently delete them.

Call:

listRepositories()

Returns:

<?xml version=”1.0” standalone=”yes”?>
<rib version=”2.0”><repository name=”ARL CFD” contact=”rib@nhse.org”
primClass=”Asset” primAttr=”Domain” config=”http://rib.cs.utk.edu/227/config.xml”
interop_handle=”http://rib.cs.utk.edu/cgi-bin/repository.pl?rh=227”/>
</rib>

Call:

repository(227)

Returns:

<?xml version=”1.0” standalone=”yes”?>
<rib version=”2.0”>
<repository name=”ARL CFD” handle=”227” contact=”rib@nhse.org”
primClass=”Asset” primAttr=”Domain” config=”http://rib.cs.utk.edu/227/config.xml”/>

```
<class name=”Asset” objects=”http://rib.cs.utk.edu/cgi-
bin/listObjects.pl?rh=227&amp;class=Asset”/></repository></rib>
```


Call:

listObjects(227, Asset)

Returns:


```
<?xml version=”1.0” standalone=”yes”?>
<rib version=”2.0”>
<repository handle=”227” config=”http://rib.cs.utk.edu/227/config.xml”>
<class name=”Asset”>
<object name=”GENIE++” handle=”2” last_modified=”19991001123819” approved=”1”
created=”19990818145740” url=”http://rib.cs.utk.edu/cgi-
bin/object.pl?rh=227&amp;class=Asset&amp;oh=2”/>
<object name=”Gridgen” handle=”3” last_modified=”19991001123819” approved=”0”
created=”19990818145740” url=”http://rib.cs.utk.edu/cgi-
bin/object.pl?rh=227&amp;class=Asset&amp;oh=3”/></class></repository></rib>
```


Call:

deleteObjects(227, Asset, 2, 3, password)

Returns: ok

Additional Resources

- RIB website
  http://www.nhse.org/RIB
- World Wide Web Consortium
  http://www.w3c.org
- RFC 2616 – Hypertext Transfer Protocol
  http://www.w3.org/Protocols/HTTP/1.1/rfc2616.txt.gz
- Mod_perl
  http://perl.apache.org