CmapServer HTTP API

Institute for Human and Machine Cognition

Internal Report

November 20, 2011

Abstract

This report describes the CmapServer's HTTP API. The API includes functionality for creating listing, accessing, creating, updating and removing resources in the CmapServer.

Last Revision: November 20, 2011

Institute for Human and Machine Cognition
15 SE Osceola Ave. - Ocala, FL 34471
http://www.ihmc.us

Contents

1. Introduction

The CmapServer HTTP API intends to provide a simpler method of access to the resources on the CmapServer. The API is inspired by the representational state transfer (REST) architectural style [1, 2]. In the CmapServer architecture the main entity is the resource, and the API simply allows to perform operations on resources. A resource can either be a folder or a file in the CmapServer. In some cases operations can be performed without needing to know the type of resource that the operation is being performed upon, but in other cases, other operations are only permitted for certain type of resources, like for example, changing permissions is only allowed for folders.

As the most important entity of the architecture is the resource, the most important part of the API is how we refer to those resources. In this API, resources are referenced by their URLs. Every resource in the CmapServer has a unique identifier, which is a sequence of characters, like for example `1J9Z7K038-10GZ44G-K5N`. Even though the resources in the CmapServer are organized in folders, the URLs and identifiers don't reflect that hierarchical structure. This is done to allow users to move resources without having to worry about breaking links. Still, in order to navigate the CmapServer structure, the hierarchical structure as defined by folders and sub-folders must be followed.

The following is the scheme of the URL to a resource:

```
http://<host-name>/rid=<resource-id>
```

where **`resource-id`** is the identifier of the resource. This URL allows to get the resource. But to perform operations on the resource, the scheme to be used is the following:

```
http://<host-name>/resources/rid=<resource-id>/
```

This distinction was made to allow HTML based navigation of the CmapServer. The former URL scheme returns HTML representations of folders and Concept Maps, while the latter URL scheme returns XML representations.

Only two HTTP verbs are used in the API: **`GET`** and **`POST`**. We agree that it would have been more desirable to have used the other HTTP verbs to have a cleaner API, but due to limitations of HTTP client APIs on browsers, the operations that could have been performed using other HTTP verbs like **`PUT`** and **`DELETE`**, needed to be overloaded using **`GET`** and **`POST`**. As a consequence, the only distinction between **`GET`** and **`POST`** would be that **`GET`** is used for operations that don't require extra information to be sent on the body of the request, while **`POST`** is used when the operation requires extra information in the body of the request, like for example, when uploading resources.

The real semantics of the operation being performed on the resources is determined by an additional parameter provided with the request called **`cmd`**. With **`GET`** requests, this parameter is provided in the URL, like for example:

```
http://<host-name>/resources/rid=<resource-id>/?cmd=get.resmeta
```

For **POST** requests, this parameter can still be provided in the URL, but can also be provided in the body, if the body contains an encoded HTML form.

Finally, the HTTP API of the CmapServer uses HTTP Basic Authentication [3, 4]. A valid user ID and password should be used when performing operations on resources that don't have general permissions for everyone. The authentication information must be sent with every request.

2. General Operations on Resources

The following sections describe operations that can be performed on any resource, regardless of the type of resources.

2.1. Metadata

```
http://<host-name>/resources/rid=<resource-id>/?cmd=get.resmeta
```

This command returns the metadata of the resource referenced by the provided identifier. The metadata is an XML document which main element is **<res-meta>** and which structure is described in `http://cmap.ihmc.us/xml/CXL.html#res-meta`. The following is an example of the XML document returned:

```
<res-meta>
    <dc:title>Groups</dc:title>
    <dc:format>x-nlk-project/x-binary</dc:format>
    <dc:identifier>
        http://umaps.ihmc.us:8080/id=1J9Z7K038-10GZ44G-K5N/
    </dc:identifier>
    <dc:description/>
    <dcterms:created>2011-04-04T14:44:02-05:00</dcterms:created>
    <dcterms:modified>2011-04-04T14:44:02-05:00</dcterms:modified>
    <dc:type/>
    <dc:publisher>FIHMC CmapTools 5.04.03  </dc:publisher>
    <dcterms:extent>18 children</dcterms:extent>
    <dc:source>
        cmap:1G64NSVLQ-2SSFWR-1:1H8BG1T5M-Y2WFLG-Q:1J9Z7K038-10GZ44G-K5N
    </dc:source>
</res-meta>
```

2.2. Deleting

```
http://<host-name>/resources/rid=<resource-id>/?cmd=delete.resource
```

This command returns a 200 OK HTTP status code if successful, or an error code with an indication of why the operation couldn't be performed.

2.3. Renaming

```
http://<host-name>/resources/rid=<resource-id>/?cmd=rename.resource
            &name=<new name>
```

`name` : new name for the resource.

On successful completion, this operation will return the **`<res-meta>`** of the modified resource.

2.4. Moving

```
http://<host-name>/resources/rid=<resource-id>/?cmd=move.resource
        &dst.fid=<id of the destination folder>
        &overwrite=<boolean>
```

`dst.fid` : identifier of the new parent folder for the resource.

`overwrite` : boolean flag indicating if any resource with the same name in the destination folder should be overwritten.

On successful completion, this operation will return the **`<res-meta>`** of the modified resource.

2.5. Copying

```
http://<host-name>/resources/rid=<resource-id>/?cmd=copy.resource
        &name=<name for the cloned resource>
        &dst.fid=<id of the destination folder>
        &overwrite=<boolean>
```

`name` : name for the cloned resource.

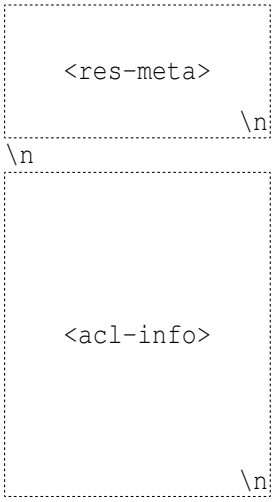

Figure 1: Layout of the body for a request to create a folder

dst.fid : identifier of the new parent folder for the resource. This parameter is optional. If not provided, the copy will be stored in the same folder as the original.

overwrite : boolean flag indicating if any resource with the same name in the destination folder should be overwritten.

3. Operations Specific to Folders

The following sections describe operations that can be performed only on folders.

3.1. Creating

```
http://<host-name>/resources/rid=<parent folder-id>/?cmd=create.folder
```

For this method, as the resource doesn't exist yet, then the identifier provided must be the identifier of the folder that will contain the resource. This method must be invoked using **POST**, and the body of the request must contain the **<res-meta>** for the new folder, followed by a blank line, and then followed by the **acl-info** (http://cmap.ihmc.us/xml/CmapWebService.html#acl-info) specifying the permissions for the new folder. Figure 1 shows the layout of the body for a request to create a folder.

3.2. Listing

For listing the contents of a folder, there are two alternative formats that can be requested from the CmapServer. The first alternative is to get a list of **res-meta** objects enclosed by a **res-meta-list** object (http:

//cmap.ihmc.us/xml/CXL.html#res-meta-list).

```
http://<host-name>/resources/rid=<folder-id>/?cmd=get.resmeta.list
```

The second alternative is to get a compact list of resources, with the main attributes of the resources flattened as attributes:

```
http://<host-name>/resources/rid=<folder-id>/?cmd=get.compact.resmeta.list
```

The following is an example of the returned XML:

```
<res-meta-list count="1" version="compact">
   <path>
        <path-element title="Groups"
             identifier="http://umaps.ihmc.us:8080/id=1J9Z7K038-10GZ44G-K5N/"/>
   </path>
   <res-meta
      title="Group 1"
      format="x-nlk-project/x-binary"
      identifier="http://umaps.ihmc.us:8080/id=1JL4QH992-KB2WG6-54KX/"
      description=""
      creator="::"
      source="cmap:1G64NSVLQ-2SSFWR-1:1J9Z7K038-10GZ44G-K5N:1JL4QH992-KB2WG6-54KX"
      created="2011-06-27T14:33:39-05:00" modified="2011-06-27T14:33:39-05:00"/>
</res-meta-list>
```

This compact format also contains a **`path`** element describing the breadcrumbs from the root folder.

3.3. Permissions

There are two operations related to permissions. The first operation allows you to get the permissions of a folder:

```
http://<host-name>/resources/rid=<folder-id>/?cmd=get.permissions
```

This method returns an **`<acl-info>`** element.

The second operation allows to set the permissions of a folder:

```
http://<host-name>/resources/rid=<folder-id>/?cmd=set.permissions
```

This operation must be invoked with **POST**, and the body of the request must contain the **<acl-info>** with the new permissions.

4. Operations Specific to Non-Folder Resources

4.1. Creating

Creating a resource is a multi-request operation. First a request indicating the beginning of a creation is sent, including the **<res-meta>** of the new resource:

```
http://<host-name>/resources/rid=<parent-folder-id>/?cmd=begin.creating.resource
```

As the resource doesn't exist yet, the identifier that must be provided is the identifier of the containing folder. This operation must be invoked with **POST**, and the body of the request must contain the **<res-meta>** of the new resource. On successful completion, this method returns a string token to be used to upload the actual resource.

The second part is to make a request to upload a part for each part of the resource (in case of multi-part resources, like Concept Maps). Single part resources must also upload at least one part.

```
http://<host-name>/resources/rid=<token>/?cmd=write.resource.part
        &partname=<part name>
        &mimetype=<MIME type of the part>
```

partname : name for the part being created, for single part resources, or for a Cmap file main part, the value for this parameter should be **cmap**.

mimetype : mime type for the part, or for the resource if the resource has only one part.

The body of the request must contain the contents for the part. For XML resources, the contents should be an XML document, for binary resources such as images, the contents should be the bytes of the image.

Finally, once all parts are written, the operation can be completed with a call with an indication that you are done writing the resource:

```
http://<host-name>/resources/rid=<token>/?cmd=done.saving.resource
```

This method returns the **<res-meta>** of the new resource.

4.2. Replacing

Replacing a resource follows the same process as creating a resource. The only part of the process that changes is how the operation is initiated. As we already have the identifier for the resource being replaced then we don't longer use the identifier of the containing folder.

```
http://<host-name>/resources/rid=<resource-id>/
                                ?cmd=begin.replacing.resource.using.RID
```

This operation must be invoked with **POST**, and the body of the request must contain the **<res-meta>** of the resource. On successful completion, this method returns a string token to be used to upload the new version of the resource. After getting the token, the process continues as when creating resources (write parts and close the operation).

References

[1] Wikipedia, "Representational state transfer."

[2] R. Fielding, *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, 2000.

[3] Wikipedia, "Basic access authentication." http://en.wikipedia.org/wiki/Basic_access_authentication.

[4] IETF, "RFC 1945: Hypertext Transfer Protocol – HTTP/1.0." http://www.ietf.org/rfc/rfc1945.txt.