Discover CID

Version : 1.4

Date : Apr 2016

Editor : Kelis

Author(s) : Thibaut Arribe
contact: contact@cid-protocol.org

Table of contents

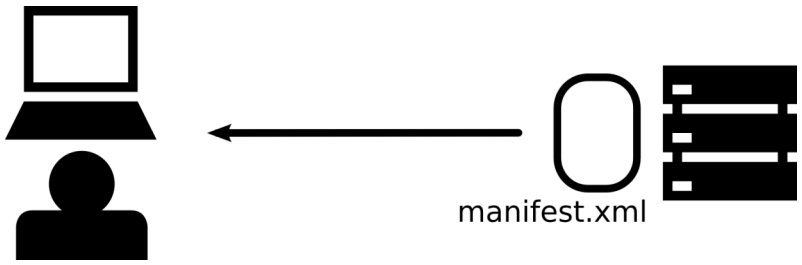
1. The CID document exchange protocol

Content Interactive Delivery (CID) is a web service protocol which provides rules to exchange contents such as documents and their metadata. CID defines how a server should describe its data exchange processes, and how a client should execute a process.

How does it work ?

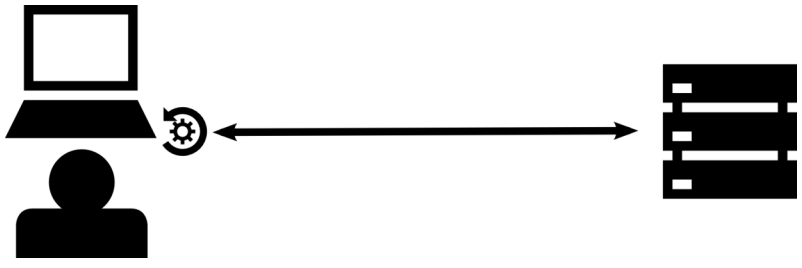
1. Download the manifest

   The CID server must define its processes in a manifest accessible through a simple GET HTTP request.

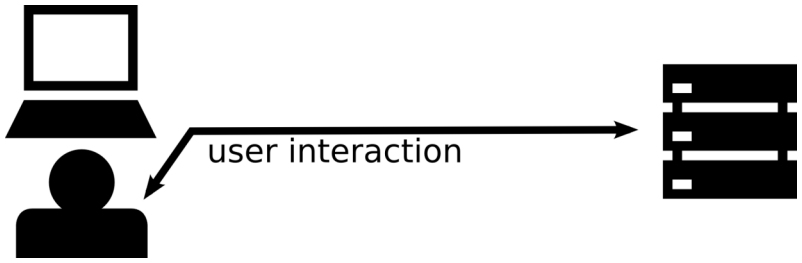


Download the manifest

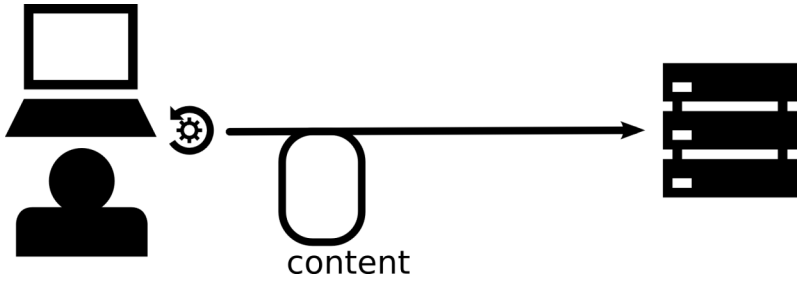
2. Then, the client must interpret the manifest and follow its steps. Three kind of steps can be used :



Regular exchange



Direct interaction



Content upload

This document is dedicated to the discovery of the CID protocol. It describes some CID example cases from a simple file upload into a remote repository to more complicated cases and introduces some code examples

(generic HTML client and PHP servers).

2. Example 1: Upload a file to a remote repository

In this case, the server is composed of a single content repository. The CID process allows client applications to upload new files in the repository.

Server manifest

```
<?xml version="1.0" encoding="UTF-8"?>
<cid:manifest xmlns:cid="http://www.cid-protocol/schema/v1/core" xmlns:xml="http://www.w3.org/XML/1998/namespace"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.cid-protocol/schema/v1/core ../manifest/cid-core.xsd ">
    <cid:process>
        <cid:label xml:lang="fr">Téléversement de fichier</cid:label>
        <cid:label xml:lang="en">File upload</cid:label>
        <cid:meta name="File-name" is="http://purl.org/dc/elements/1.1/title">
            <cid:label xml:lang="fr">Nom du fichier</cid:label>
            <cid:label xml:lang="en">File name</cid:label>
            <cid:doc xml:lang="en">
                The file-name will be used in the public url of the file
                Example: http://.../1d57fe87zr45sdz796m/monImage.png
            </cid:doc>
        </cid:meta>
        <cid:meta name="Public-url" is="http://schema.org/URL"/>
        <!--
            The following fragment is the main part of the process declaration. It
            describes each step of the process.
            Each step declares the (previously-documented) used (optional), needed
            or returned metadatas. This process contains only one step: the upload
            step which is required to complete the file deposit and which is
            composed by only one request (attribute step="unique"). This step
            accepts the File-name metadata and returns the Public-url of the
            uploaded file.
         -->
        <cid:upload url="http://example.com/upload.php" useMetas="File-name" returnMetas="Public-url" required="true"/>
    </cid:process>
    <!--
      The second part of the manifest is dedicated to the enabled authentication
      methods.
    -->

    <!--
      The last part of the manifest is dedicated to the transport declaration.
      In this example, the transport is a single HTTP request. The properties
      attribute defines how the metadata should be sent.
      The returned properties must be inserted in a single JSON object in the
      body of the HTTP response.
     -->
    <cid:transports>
        <cid:webTransport>
            <cid:authentications>
                <cid:basicHttp/>
            </cid:authentications>
            <cid:webUpload>
                <cid:request method="GET" properties="header queryString"/>
                <cid:request method="PUT" properties="header queryString"/>
                <cid:request method="POST" properties="header queryString"/>
```

```
48                <cid:request method="POST;multipart/form-data" properties="header
  queryString post"/>
49              </cid:webUpload>
50          </cid:webTransport>
51      </cid:transports>
52  </cid:manifest>
```

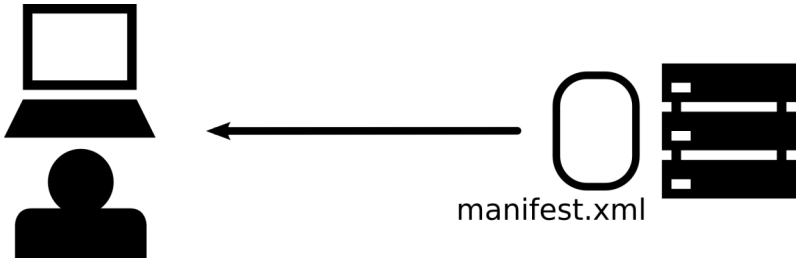
The process step by step

1. Manifest exposition

   The server must expose the XML manifest to an accessible URL. For example, at ***http://example.com/manifest.xml***.

2. Download the manifest

   In order to understand the process, a client must download the server manifest and analyze it.

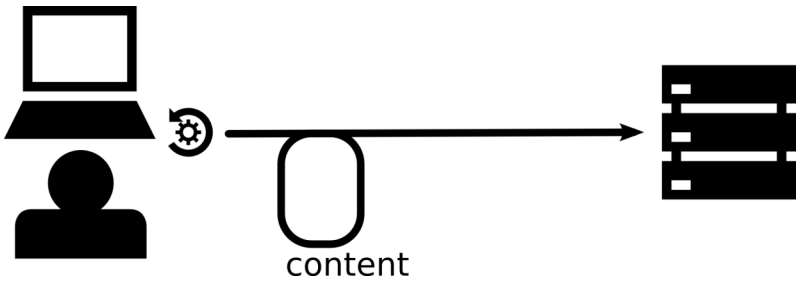


Download the manifest

3. Single upload

   The client can now execute the defined step. In this case, the client must send the file in a single *HTTP* request following one of the valid transport descriptions of the manifest.

   For example, it could send the file in the body of a *PUT* request. The *File-name* metadata could be sent in the header of the request. The body of the server response must be a *JSON* object containing the returned metadata: *Public-url*.



Content upload

3. Example 2: Retrieve the public URL of a remote content

In this case, the server handles files (like an ECM platform). A client needs the unique *URL* of a remote content. The CID process allows a user to select a content and defines how the client can retrieve its *Public-url*.

Server manifest

```
<?xml version="1.0" encoding="UTF-8"?>
<cid:manifest xmlns:cid="http://www.cid-protocol/schema/v1/core" xmlns:xml="http://www.w3.org/XML/1998/namespace"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.cid-protocol/schema/v1/core ../manifest/cid-core.xsd ">
    <cid:process is="http://schema.org/FindAction http://schema.org/CheckAction">
        <cid:label xml:lang="en">Public URL retrieving</cid:label>
        <cid:doc xml:lang="en">Browse your online contents to retrieve a public URL.</cid:doc>
        <cid:meta name="Public-url" is="http://schema.org/url"/>
        <!-- The process is composed by a single interaction step. -->
        <cid:interact url="http://.." returnMetas="Public-url" required="true"/>
    </cid:process>
    <!--
    The web interaction begins with a single HTTP request and ends with a message event
    (http://www.w3.org/TR/webmessaging) containing a cidInteraction property stored in the data
    part of the event. This property must be set to "ended" (in case of regular end of the step)
    or "aborted" (in case of step cancelled by the user). The client must instantiate a web frame
    and fill it with the result of the first request. It gets the returned metadata in the data
    part of the message event. In this example, the server does not furnish any authentication scheme.
    It could handle a web authentication in the frame or let a public access to the platform.
    -->
    <cid:transports>
        <cid:webTransport>
            <cid:authentications/>
            <cid:webInteract>
                <cid:request method="GET" properties="header queryString"/>
                <cid:request method="POST;application/x-www-form-urlencoded" properties="queryString header post"/>
                <cid:request method="POST;multipart/form-data" properties="header queryString post"/>
            </cid:webInteract>
        </cid:webTransport>
    </cid:transports>
</cid:manifest>
```

The process step by step

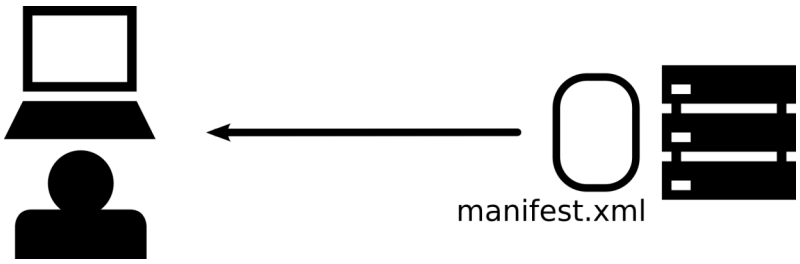
1. Manifest exposition

   The server must expose the XML manifest to an accessible URL. For example, at ***http://example.***

com/manifest.xml.

2. Download the manifest

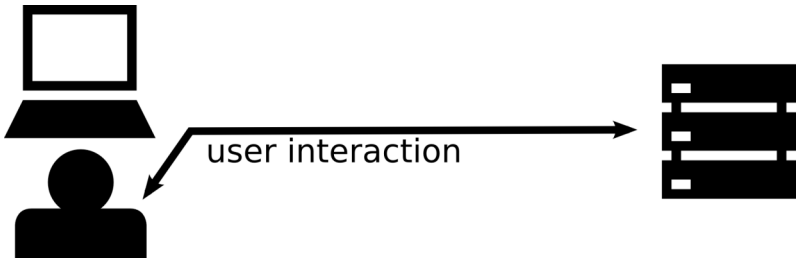
   In order to understand the process, a client must download the server manifest and analyze it.



Download the manifest

3. Direct interaction

   The client can now execute the defined step. In this case, the client must send a single *HTTP* request and show the result in a web frame. The user should then interact directly with the server using the frame.



Direct interaction

   At the end of the process, the web page of the frame must post a message event throught the message API. The client must catch this event and read the *endInteraction* property store in the data part of the event. This property must be set to ended when the interaction is regularly closed or aborted when the user cancelled the process during the interaction. In case of regularly ended interaction, the returned metadata documented in this step are also stored: the *Public-url* in this case.

4. Example 3: Upload an archived complex content

In this case, the server accepts complex contents like an archived website. The server needs more metadata than in the previous cases. The process is composed of two steps, one optional interaction step which helps the user to deploy his website, and the required upload step.

Server manifest

```
<?xml version="1.0" encoding="UTF-8"?>
<cid:manifest xmlns:cid="http://www.cid-protocol/schema/v1/core" xmlns:xml="http://www.w3.org/XML/1998/namespace"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.cid-protocol/schema/v1/core ../manifest/cid-core.xsd ">
    <cid:process is="http://schema.org/SendAction">
        <cid:label xml:lang="fr">Dépot de d'archives de ressources</cid:label>
        <cid:meta name="Folder-path">
            <cid:label xml:lang="en">Storage folder</cid:label>
            <cid:doc xml:lang="en">
                Path to the folder where the archive is unarchived.</cid:doc>
        </cid:meta>
        <!--
          The content-type metadata describes the type of the archive (and not of
          its content). In this description, the server furnishes three possible
          values. The client must select a value between them.
        -->
        <cid:meta name="Content-type" is="http://purl.org/dc/elements/1.1/type">
            <cid:label xml:lang="en">Available archived formats</cid:label>
            <cid:value>application/zip</cid:value>
            <cid:value>application/jar</cid:value>
            <cid:value>application/x-tar</cid:value>
        </cid:meta>
        <!--
          The names of the files compressed in a zip archive could be encoded
          following several schemes. In order to correctly decode it, this server
          request the encoding scheme as a metadata.
        -->
        <cid:meta name="Zip-file-names-encoding">
            <cid:label xml:lang="fr">File name encoding of the archive</cid:label>
            <cid:doc xml:lang="fr">This metadata should be sent if the Content-type is "application/zip".</cid:doc>
            <cid:value>UTF-8</cid:value>
            <cid:value>ISO-8859-1</cid:value>
            <cid:value>CP437</cid:value>
        </cid:meta>
        <!--
          This process is composed of two steps :

          1. An optional interaction step (which could use the Folder-path
          metadata). This interaction allows the user to define the path where
          the file will be unarchived. In this example, the Folder-path could be
          sent by the client in the request. The server could then build a
          dedicated GUI where this path is already selected. The server returns
          the overloaded Folder-path value in the "cid-interaction-ended" event.

          2. A required upload step, which needs the folder-path and the
          content-type metadata and which could use the Zip-file-names-encoding
```

```
47          metadata.
48        -->
49        <cid:interact url="http://..." useMetas="Folder-path" returnMetas="Folder-path"
required="false"/>
50        <cid:upload url="http://..." needMetas="Zip-file-names-encoding Content-type
Folder-path" required="true"/>
51    </cid:process>
52    <!--
53      The web transport specified by this server includes a session property.
54      This property could be returned whith the metadata at each step of the
55      process. When this property is returned, the client must send it back in the
56      next step of the process. This feature enables statefull servers.
57    -->
58    <cid:transports>
59        <cid:webTransport sessionProperties="session-ID">
60            <cid:authentications>
61                <cid:basicHttp/>
62                <cid:webAuthentication url="http://example.com/auth"/>
63            </cid:authentications>
64            <cid:webInteract>
65                <cid:request method="GET" properties="header queryString"/>
66                <cid:request method="POST;application/x-www-form-urlencoded" properties=
"queryString header post"/>
67                <cid:request method="POST;multipart/form-data" properties="header
queryString post"/>
68            </cid:webInteract>
69            <cid:webUpload>
70                <cid:request method="GET" properties="header queryString"/>
71                <cid:request method="PUT" properties="header queryString"/>
72                <cid:request method="POST" properties="queryString header"/>
73                <cid:request method="POST;multipart/form-data" properties="header
queryString post"/>
74            </cid:webUpload>
75        </cid:webTransport>
76    </cid:transports>
77</cid:manifest>
```

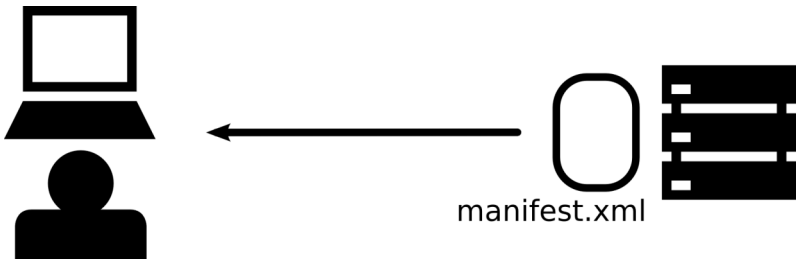
The process step by step

1. Manifest exposition

   The server must expose the XML manifest to an accessible URL. For example, at ***http://example.com/manifest.xml***.

2. Download the manifest

   In order to understand the process, a client must download the server manifest and analyze it.

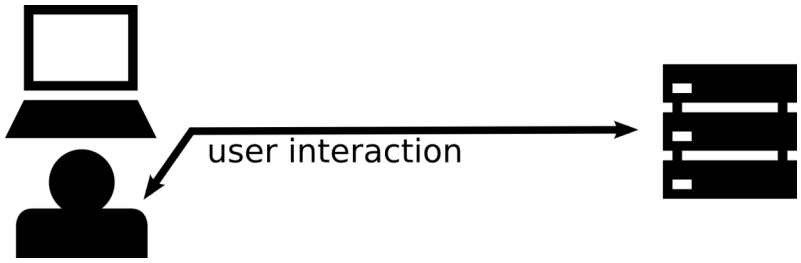


Download the manifest

3. Optional interaction

   Depending on the context, the client could be unable to furnish a valid *Folder-path* metadata. This interaction step allows the server to:

   - optionally get a *Folder-path* given by a client ;
   - interact directly with the user in order to overload this value;
   - return the overloaded value to the client in data part of the message event.



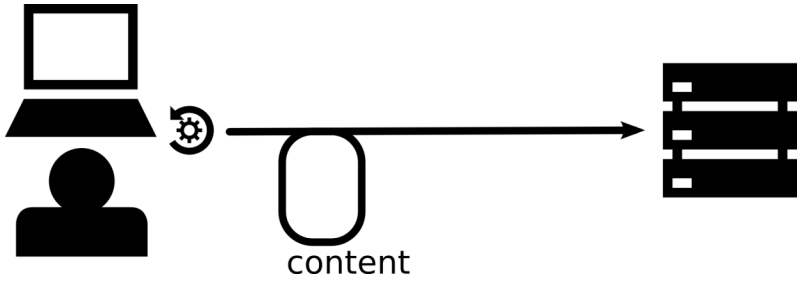
Direct interaction

   In addition, the server could also return the defined session property (*session-id*) in the body of the custom event.

4. Archive upload

   At this stage, the client should be able to send all the required properties either with or without interaction. The client must send the upload request following one of the defined upload transports.

   For example, it could send the content via an *HTTP POST* request, the archive in the body and the metadata as a query string (see rfc3986[*http://tools.ietf.org/html/rfc3986*]).



Content upload

5. Example 4: Deploy a *SCORM* content

In this case, the CID server is an e-learning platform which accepts SCORM[*http://www.adlnet.org/scorm/*] packages. The deployment of *SCORM* packages needs the intervention of a user in order to fill some platform-specific information and to then deploy the package in a pedagogical activity.

Server manifest

```
<?xml version="1.0" encoding="UTF-8"?>
<cid:manifest xmlns:cid="http://www.cid-protocol/schema/v1/core" xmlns:xml="http://www.w3.org/XML/1998/namespace" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.cid-protocol/schema/v1/core ../manifest/cid-core.xsd ">
    <!--
        This manifest declares two transports : the first without cookies, the
        second with. Two processes are also defined. They are respectively binded
        to the two different webtransports.
    -->

    <!--
        This process is binded to standard Transport. It uses a session property to
        enable statefull servers.
    -->
    <cid:process transports="standardTransport" is="http://schema.org/SendAction">
        <cid:label xml:lang="en">SCORM deployment</cid:label>

        <!--
            A restriction is similar to a meta with the difference that the
            restriction is implicit in the process. The value is unique and set only
            once in the manifest and never sent by the client.
        -->
        <cid:restriction name="Zip-file-names-encoding" value="UTF-8">
            <cid:label xml:lang="en">Encoding of the files-name</cid:label>
        </cid:restriction>

        <!-- Definition of custom mimeType -->
        <cid:meta name="Content-type" is="http://purl.org/dc/elements/1.1/type">
            <cid:value>application/xx-scorm;v1.2</cid:value>
            <cid:value>application/xx-scorm;v2004</cid:value>
        </cid:meta>

        <cid:upload url="http://example.com/upload?response=noCookies" needMetas="Content-type" required="true"/>
        <cid:interact url="http://example.com/interact?response=noCookies" required="true"/>
    </cid:process>

    <!--
            This process is binded to the second transport. It needs a client which
            supports cookies
        -->
    <cid:process transports="transportWithCookies" is="http://schema.org/SendAction">
        <cid:restriction name="Zip-file-names-encoding" value="UTF-8"/>

        <!-- Definition of custom mimeType -->
        <cid:meta name="Content-type" is="http://purl.org/dc/elements/1.1/type">
            <cid:value>application/xx-scorm;v1.2</cid:value>
```

```
            <cid:value>application/xx-scorm;v2004</cid:value>
        </cid:meta>

        <cid:upload url="http://example.com/upload?response=withCookies" needMetas="Content-type" required="true"/>
        <cid:interact url="http://example.com/interact?response=withCookies" required="true"/>
    </cid:process>

    <cid:transports>
        <cid:webTransport id="standardTransport">
            <cid:authentications>
                <cid:basicHttp/>
            </cid:authentications>
            <cid:webInteract>
                <cid:request method="GET" properties="header queryString"/>
                <cid:request method="POST;application/x-www-form-urlencoded" properties="queryString header post"/>
                <cid:request method="POST;multipart/form-data" properties="header queryString post"/>
            </cid:webInteract>

            <cid:webUpload>
                <cid:request method="GET" properties="header queryString"/>
                <cid:request method="POST" properties="queryString header"/>
                <cid:request method="POST;multipart/form-data" properties="header queryString post"/>
            </cid:webUpload>
        </cid:webTransport>

        <cid:webTransport id="transportWithCookies" needCookies="true">
            <cid:authentications>
                <cid:basicHttp/>
            </cid:authentications>
            <cid:webInteract>
                <cid:request method="GET" properties="header queryString"/>
                <cid:request method="POST;application/x-www-form-urlencoded" properties="queryString header post"/>
                <cid:request method="POST;multipart/form-data" properties="header queryString post"/>
            </cid:webInteract>

            <cid:webUpload>
                <cid:request method="GET" properties="header queryString"/>
                <cid:request method="PUT" properties="header queryString"/>
                <cid:request method="POST" properties="queryString header"/>
                <cid:request method="POST;multipart/form-data" properties="header queryString post"/>
            </cid:webUpload>
        </cid:webTransport>
    </cid:transports>
</cid:manifest>
```

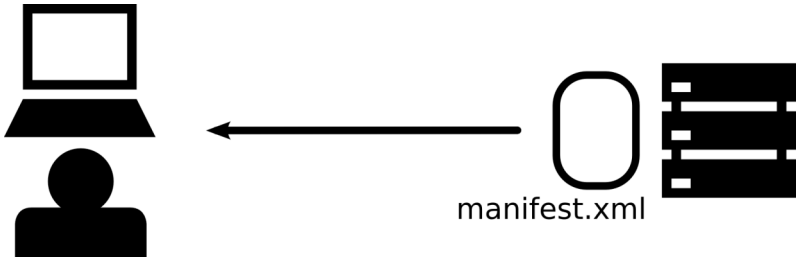
The process step by step

1. Manifest exposition

   The server must expose the XML manifest to an accessible URL. For example, at ***http://example.com/manifest.xml***.

2. Download the manifest

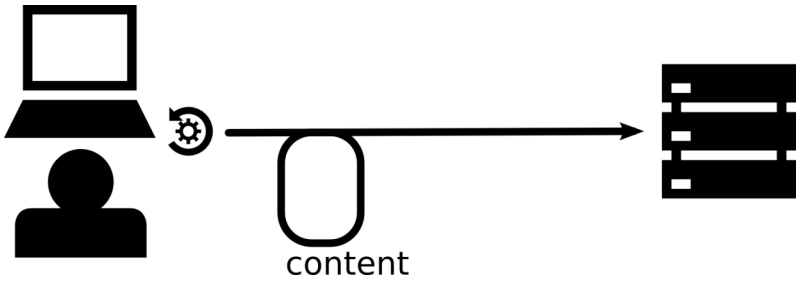
   In order to understand the process, a client must download the server manifest and analyze it.



Download the manifest

3. *SCORM* upload

   A client must send packages which are conform to the suggested values of the *content-type* metadata and to the imposed value of *Zip-file-names-encoding* (which is *UTF-8*). In this example, a client could send a form in a *POST* request which contains the metadata and the *SCORM* package in a field named *cidContent*.

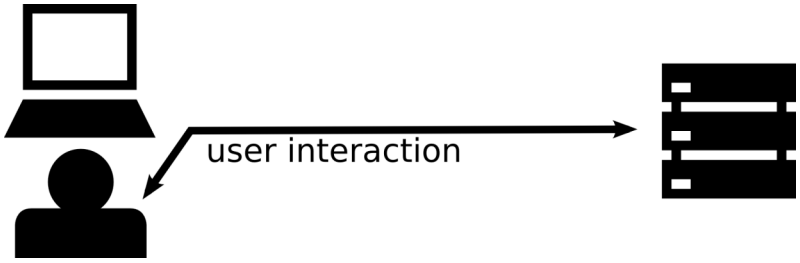


Content upload

4. *SCORM* deployment

   The second step of the process is required. It allows the user to interact directly with the remote server. The latter furnishes a form to fill more metadata or an interface to deploy the *SCORM* package.

   This interaction must begin by one of the documented *webInteraction* method. For example, it could begin by an *HTTP POST* with the session property in a form data.



Direct interaction

6. Example 5: Upload video content into a remote DAM server

In this case, the CID server is a Digital Asset Management (*DAM*) server. This server accepts large contents such as video files. It supports an extra optional step which allows the client to check the authentication and authorization of the user before sending the content.

This example includes a pre-upload interaction step which allows the user to fill platform-specific metadata and a post-upload wait step which is optional and allows the user to monitor the server processing.

Server manifest

```
<?xml version="1.0" encoding="UTF-8"?>
<cid:manifest xmlns:cid="http://www.cid-protocol/schema/v1/core" xmlns:xml="http://www.w3.org/XML/1998/namespace" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.cid-protocol/schema/v1/core ../manifest/cid-core.xsd ">
    <cid:process is="http://schema.org/SendAction">
        <cid:label xml:lang="en">DAM</cid:label>
        <cid:meta name="Content-type" is="http://purl.org/dc/elements/1.1/type">
            <cid:value>image/png</cid:value>
            <cid:value>video/mpeg4</cid:value>
        </cid:meta>

        <cid:meta name="Public-url"  is="http://schema.org/url"/>
        <cid:meta name="dc-title" is="http://purl.org/dc/elements/1.1/title"/>
        <cid:meta name="dc-creator" is="http://purl.org/dc/elements/1.1/creator"/>
        <cid:meta name="dc-modified" is="http://purl.org/dc/terms/modified"/>
        <cid:meta name="tags" is="http://schema.org/keywords"/>

        <!--
            An exchange step represents a default HTTP request.

            In this process, the content to upload could be really large. This first
            optional step allows a client to check the authentication and
            authorization of a user before sending the content.
        -->
        <cid:exchange url="http://example.com/checkAuthorization" is="http://schema.org/AuthorizeAction" required="false"/>
        <cid:interact url="http://example.com/interact-pre-delivery" useMetas="dc-title dc-creator dc-modified" returnMetas="dc-title dc-creator dc-modified tags" required="true"/>
        <!--
            An async upload could be used when the post-upload processing need a long
            time. After such an upload, the server could return a 202 HTTP code.
            A system-oriented wait will then allow the client to send regular exchange
            requests to the systemWait URL until it gets a different HTTP code than 202.
            A user-oriented wait will then allow the client to instanciate an interact
            step at the userWait URL.
        -->
        <cid:upload url="http://example.com/upload?step=start" needMetas="dc-title dc-creator dc-modified tags" required="true">
            <cid:systemWait url="http://example.com/wait?method=system" returnMetas="Public-url"/>
            <cid:userWait url="http://example.com/wait?method=user" returnMetas="Public-url"/>
        </cid:upload>
```

```
37
38      </cid:process>
39
40      <cid:transports>
41          <cid:webTransport sessionProperties="session-ID">
42              <cid:authentications>
43                  <cid:basicHttp/>
44                  <cid:webAuthentication  url="http://example.com/auth"/>
45              </cid:authentications>
46
47              <cid:webExchange>
48                  <cid:request method="GET" properties="header queryString"/>
49                  <cid:request method="POST;application/x-www-form-urlencoded" properties=
    "queryString header post"/>
50                  <cid:request method="POST;multipart/form-data" properties="header
    queryString post"/>
51              </cid:webExchange>
52
53              <cid:webInteract>
54                  <cid:request method="GET" properties="header queryString"/>
55                  <cid:request method="POST;application/x-www-form-urlencoded" properties=
    "queryString header post"/>
56                  <cid:request method="POST;multipart/form-data" properties="header
    queryString post"/>
57              </cid:webInteract>
58
59              <cid:webUpload>
60                  <cid:request method="GET" properties="header queryString"/>
61                  <cid:request method="PUT" properties="header queryString"/>
62                  <cid:request method="POST" properties="queryString header"/>
63                  <cid:request method="POST;multipart/form-data" properties="header
    queryString post"/>
64              </cid:webUpload>
65          </cid:webTransport>
66      </cid:transports>
67  </cid:manifest>
```

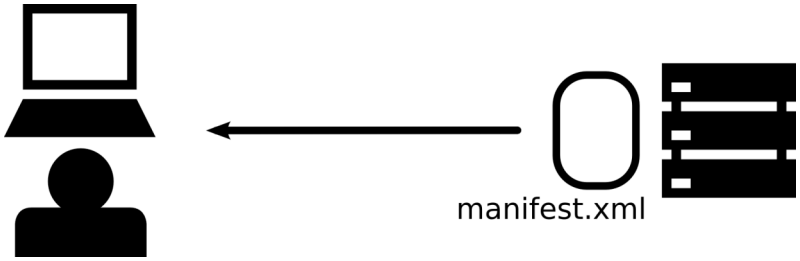
The process step by step

1. Manifest exposition

   The server must expose the XML manifest to an accessible URL. For example, at ***http://example.com/manifest.xml***.

2. Download the manifest

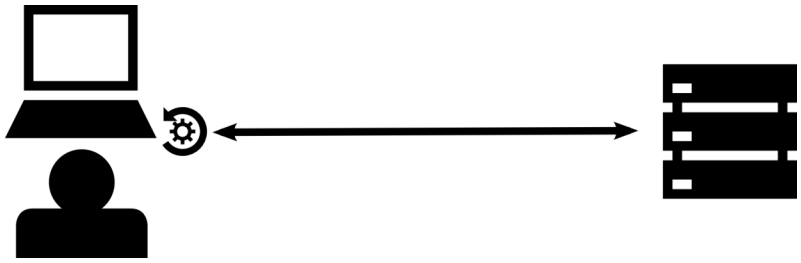
   In order to understand the process, a client must download the server manifest and analyze it.



Download the manifest

3. Optional exchange

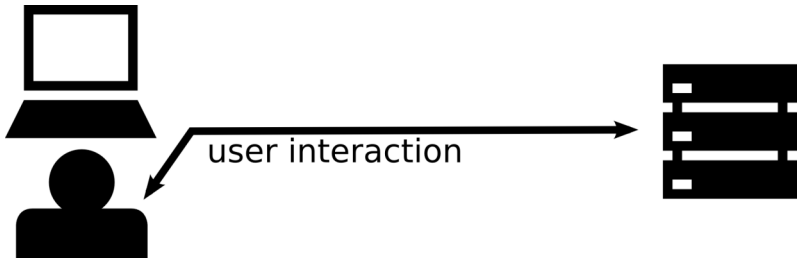
   The client could send a first optional exchange request in order to check the authorization and the authentication of the user. In this case, the client could send an *HTTP GET* request which contains a basic or digest authentication header.



Regular exchange

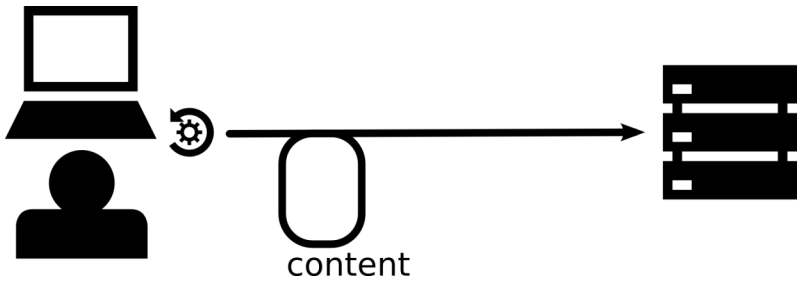
4. Required interaction

   The first interaction is required. It allows the server to interact directly with the user in order to negotiate the content processing.



Direct interaction

5. Upload



Content upload

   When the server is unable to return the meta properly in the HTTP response, the upload step could define an asynchronous upload. This asynchronous upload step defines wait strategies to let the client get the final meta later. One of these wait strategies must be operated by the client when the upload response of the server contains a HTTP response code 202.

   It is possible to define two wait strategies :

   - A user-oriented wait: the client instantiates a web-frame following the same scheme than an interact step.
   - A system-oriented wait: the client must send regularly an exchange request until the returned HTTP code is different than 202.

7. Open-source generic CID client

How to use ?

A generic CID client could be used on the CID protocol website[*http://www.cid-protocol.org/co/client.html*].

Compatibility

This CID client uses the HTML5 File[*http://www.w3.org/TR/FileAPI/*] and Messaging[*http://www.w3.org/TR/webmessaging/*] API. The client should work in IE10, Firefox 3.6, Chrome 12, Safari 6, Opera 11.5 and later. See the cross-document messaging[*http://caniuse.com/#feat=x-doc-messaging*] and file[*http://caniuse.com/#feat=fileapi*] API support for more details.

Use the client

1. Open the client[*http://www.cid-protocol.org/co/client.html*] in a compatible browser ;
2. Fill the URL input with a manifest location ;
3. Let the client follow the instructions of the manifest.

How to embed ?

The client is open source and could be embed on any web platform.

Get the sources

```
1 git clone git@github.com:scenari/cid.git
```

The client should be store at the path `cid/client-CID`.

Install the client on a web server

Upload the `index.html` file and the `js` folder onto a web server. The client uses only static contents so there is no need for php or sql servers.

Overload the sources

The client is made of a simple HTML file and a javascript library. The HTML can be completed to be properly integrated to a web platform.

8. Open-source generic servers

8.1. SingleCIDServer - a single folder remote repository

8.1.1. Presentation

SingleCIDServer is a very simple server written in PHP. It allows the uploading of files into a single remote folder. Its manifest contains only two steps :

1. an optional exchange which checks the authorization and authentication ;
2. a required upload which contains the file to deploy.

SingleCIDServer manifest

```
<?xml version='1.0' encoding='UTF-8'?>
<cid:manifest xmlns:cid="http://www.cid-protocol/schema/v1/core" xmlns:xml="http://www.w3.org/XML/1998/namespace"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.cid-protocol/schema/v1/core ../manifest/cid-core.xsd ">
    <cid:process>
        <cid:label xml:lang='en'>Deposit file</cid:label>
        <cid:doc xml:lang='en'>This server accepts any content and allows the sender to choose the storage directory.
        </cid:doc>
        <cid:meta name='file-name' is='http://schema.org/name'>
            <cid:label>File name</cid:label>
        </cid:meta>
        <cid:meta name='file-url' is='http://schema.org/url'/>
        <cid:exchange url="http://www.example.com?cdaction=testAuth\" required='false'
                      is='http://schema.org/AuthorizeAction'/>
        <cid:upload url="http://www.example.com?cdaction=upload\" required='true' useMetas='file-name'/>
        <cid:interact url="http://www.example.com?cdaction=negotiationFrame\" required='true' returnMetas='file-url'/>
    </cid:process>
    <cid:transports>
        <cid:webTransport sessionProperties='uploadedFile'>
            <cid:authentications>
                <cid:basicHttp/>
            </cid:authentications>
            <cid:webExchange>
                <cid:request method='GET' properties='header queryString'/>
                <cid:request method='POST;application/x-www-form-urlencoded' properties='post header queryString'/>
                <cid:request method='POST;multipart/form-data' properties='post header queryString'/>
            </cid:webExchange>
            <cid:webUpload>
                <cid:request method='PUT' properties='header queryString'/>
                <cid:request method='POST' properties='header queryString'/>
                <cid:request method='POST;multipart/form-data' properties='post header queryString'/>
            </cid:webUpload>
            <cid:webInteract>
                <cid:request method='GET' properties='queryString'/>
```

```
35                <cid:request method='POST;application/x-www-form-urlencoded' properties=
'post queryString'/>
36                <cid:request method='POST;multipart/form-data' properties='post
queryString'/>
37             </cid:webInteract>
38         </cid:webTransport>
39     </cid:transports>
40 </cid:manifest>
```

8.1.2. Install SingleCIDServer

1. Get SingleCIDServer

   ```
   1 git clone git@github.com:scenari/cid.git
   ```

   The file should be stored at the path `cid/singleCIDRep/SingleCIDRep.php`.

2. Move the php file on a web server

   The SingleCIDServer.php file should be uploaded on a web PHP server.

3. Initialize your SingleCIDServer
   - Ensure that the web server has writing permission on the parent folder.
   - Open the file in a web browser.
   - Fill the form:
     - set up a login and a password
     - *Auto-dezip* option allows SingleCIDServer to automatically dezip archived content. It should be checked if you wish to send archived website.
     - *Authorize php upload* option allows SingleCIDServer to receive php file. This feature is a serious lack of security and should be let unchecked unless your server is deployed in a secure context.
   - Click on the initialize button.
   - The SingleCIDServer is ready to be used.

   Upload size

   The page of initialization shows the maximum upload size of php. This size could be change in the php parameters.

   Remove permission on the parent folder

   Once the server is initialized, the writing permission of the web server on the parent folder should be removed.

8.1.3. Use SingleCIDRep

Get the manifest

Open the php file in a web browser. For example: ***http://www.example.com/SingleCIDRep.php***.

Access to the uploaded files

Open the parent folder in a web browser. For example: ***http://www.example.com***.

Control the initialization of the SingleCIDServer

Open the php file with the *control* option. For example: http://www.example.com/SingleCIDRep.php?cdaction=control

It controls the writing permission and shows the maximum upload size of php.

Reset the login and password

Remove the file param.php which was created at the same place than the SingleCIDRep.php file. Open SingleCIDRep.php in a web browser to reinitialize the server.

8.2. SimpleCIDServer - a multi folders remote repository

8.2.1. Presentation

SimpleCIDServer is a very simple server written in PHP. It allows the uploading of files into a tree of folders. It contains three steps :

1. an optional exchange which checks the authorization and authentication ;
2. a required upload which contains the file to deploy ;
3. a required interaction step which allows the user to choose the destination of the uploaded file.

SimpleCIDServer manifest

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <cid:manifest xmlns:cid="http://www.cid-protocol/schema/v1/core" xmlns:xml="http://www.
   w3.org/XML/1998/namespace"
3                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4                xsi:schemaLocation="http://www.cid-protocol/schema/v1/core ../manifest
   /cid-core.xsd ">
5      <cid:process>
6          <cid:label xml:lang='en'>Deposit file</cid:label>
7          <cid:doc xml:lang='en'>This server accepts any content and allows the sender to
   choose the storage directory.
8          </cid:doc>
9          <cid:meta name='file-name' is='http://schema.org/name'>
10             <cid:label>File name</cid:label>
11         </cid:meta>
12         <cid:meta name='file-url' is='http://schema.org/url'/>
13         <cid:exchange url="http://www.example.com?cdaction=testAuth\" required='false'
14                       is='http://schema.org/AuthorizeAction'/>
15         <cid:upload url="http://www.example.com?cdaction=upload\" required='true'
   useMetas='file-name'/>
16         <cid:interact url="http://www.example.com?cdaction=negotiationFrame\" required=
   'true' returnMetas='file-url'/>
17     </cid:process>
18     <cid:transports>
19         <cid:webTransport sessionProperties='uploadedFile'>
20             <cid:authentications>
21                 <cid:basicHttp/>
22             </cid:authentications>
23             <cid:webExchange>
```

```
24                <cid:request method='GET' properties='header queryString'/>
25                <cid:request method='POST;application/x-www-form-urlencoded' properties=
'post header queryString'/>
26                <cid:request method='POST;multipart/form-data' properties='post header
queryString'/>
27            </cid:webExchange>
28            <cid:webUpload>
29                <cid:request method='PUT' properties='header queryString'/>
30                <cid:request method='POST' properties='header queryString'/>
31                <cid:request method='POST;multipart/form-data' properties='post header
queryString'/>
32            </cid:webUpload>
33            <cid:webInteract>
34                <cid:request method='GET' properties='queryString'/>
35                <cid:request method='POST;application/x-www-form-urlencoded' properties=
'post queryString'/>
36                <cid:request method='POST;multipart/form-data' properties='post
queryString'/>
37            </cid:webInteract>
38        </cid:webTransport>
39    </cid:transports>
40</cid:manifest>
```

8.2.2. Install SimpleCIDServer

1. Get SimpleCIDServer

   ```
   1 git clone git@github.com:scenari/cid.git
   ```

   The file should be stored at the path `cid/simpleCIDServer/SimpleCIDServer.php`.

2. Move the php file on a web server

   The SimpleCIDServer.php file should be uploaded on a PHP web server.

3. Initialize your SimpleCIDServer
   - Ensure that the web server has writing permission on the parent folder.
   - Open the file in a web browser.
   - Fill the form:
     - the upload directory parameter specify the root sub-folder of the uploads
     - the tmp directory parameter specify the sub-folder which contains temporary file between the upload and the interaction steps
     - set up one or more login and a password
     - *Authorize php upload* option allows SimpleCIDServer to receive php file. This feature is a serious lack of security and should be let unchecked unless your server is deployed in a secure context.
   - Click on the initialize button.
   - The SimpleCIDServer is ready to be used.

> Upload size
>
> The page of initialization shows the maximum upload size of php. This size could be change in the php parameters.

Remove permission on the parent folder

Once the server is initialized, the writing permission of the web server on the parent folder should be removed.

8.2.3. Use SimpleCIDRep

Get the manifest

Open the php file in a web browser. For example: ***http://www.example.com/SimpleCIDRep.php***.

Access to the uploaded files

Open the parent folder in a web browser. For example: ***http://www.example.com***.

Control the initialization of the SimpleCIDServer

Open the php file with the *control* option. For example: http://www.example.com/SimpleCIDRep.php?cdaction=control

It controls the writing permission and shows the maximum upload size of php.

Reset the login and password

Remove the file param.php which was created at the same place than the SimpleCIDServer.php file. Open SimpleCIDServer.php in a web browser to reinitialize the server.

9. Manifest schema

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
           targetNamespace="http://www.cid-protocol/schema/v1/core" xmlns:cid=
"http://www.cid-protocol/schema/v1/core">
    <xs:import schemaLocation="http://www.w3.org/2001/xml.xsd" namespace="http://www.w3.
org/XML/1998/namespace"/>

    <xs:element name="manifest">
        <xs:complexType>
            <xs:sequence>
                <!-- Human oriented documentation -->
                <xs:element name="label" minOccurs="0" maxOccurs="unbounded" type="cid:
localized-element"/>
                <xs:element name="doc" minOccurs="0" maxOccurs="unbounded" type="cid:
localized-element"/>

                <!-- list of processes -->
                <xs:element name="process" maxOccurs="unbounded">
                    <xs:complexType>
                        <xs:sequence>
                            <!-- Human oriented process documentation -->
                            <xs:element name="label" minOccurs="0" maxOccurs="unbounded"
 type="cid:localized-element"/>
                            <xs:element name="doc" minOccurs="0" maxOccurs="unbounded"
type="cid:localized-element"/>
                            <!-- Meta and restriction declarationwebAsyncUpload -->
                            <xs:element name="restriction" minOccurs="0" maxOccurs=
"unbounded" type="cid:restriction-element"/>
                            <xs:element name="meta" minOccurs="0" maxOccurs="unbounded"
type="cid:meta-element"/>
                            <!-- Steps declaration -->
                            <xs:choice maxOccurs="unbounded">
                                <xs:element name="exchange" type="cid:asyncStep-element"
/>
                                <xs:element name="upload" type="cid:asyncStep-element"/>
                                <xs:element name="interact" type="cid:regularStep-
element"/>
                            </xs:choice>
                        </xs:sequence>

                        <!-- Bind a this process to a specific transport -->
                        <xs:attribute name="transports" type="cid:NCName-list"/>
                        <!-- Computer oriented process description -->
                        <xs:attribute name="is" type="cid:uri-list"/>
                    </xs:complexType>
                </xs:element>

                <!-- List of defined transports -->
                <xs:element name="transports">
                    <xs:complexType>
                        <xs:choice maxOccurs="unbounded">
                            <!-- Web transport declaration -->
                            <xs:element name="webTransport">
                                <xs:complexType>
                                    <xs:all>
                                        <!-- First transport declaration step :
supported authentication -->
                                        <xs:element name="authentications">
```

```
                                        <xs:complexType>
                                            <!-- An empty authentications element
means "no authentication needed" -->
                                            <xs:choice minOccurs="0" maxOccurs=
"unbounded">
                                                <xs:element name="noAuthentication">
                                                    <xs:complexType/>
                                                </xs:element>
                                                <xs:element name="basicHttp">
                                                    <xs:complexType/>
                                                </xs:element>
                                                <!-- Must be operated as a web
interact -->
                                                <xs:element name="webAuthentication"
>
                                                    <xs:complexType>
                                                        <xs:attribute name="url"
type="xs:anyURI"/>
                                                    </xs:complexType>
                                                </xs:element>
                                            </xs:choice>
                                        </xs:complexType>
                                    </xs:element>

                                    <!-- List of step transport declaration -->
                                    <xs:element name="webExchange" minOccurs="0">
                                        <xs:complexType>
                                            <xs:sequence>
                                                <xs:element name="request" type=
"cid:regularRequest" maxOccurs="unbounded"/>
                                            </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                    <xs:element name="webInteract" minOccurs="0">
                                        <xs:complexType>
                                            <xs:sequence>
                                                <xs:element name="request" type=
"cid:regularRequest" maxOccurs="unbounded"/>
                                            </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                    <xs:element name="webUpload" minOccurs="0">
                                        <xs:complexType>
                                            <xs:sequence>
                                                <xs:element name="request" type=
"cid:uploadRequest" maxOccurs="unbounded"/>
                                            </xs:sequence>
                                        </xs:complexType>
                                    </xs:element>
                                </xs:all>

                                <!--  Id needed to bind a process to a specific
transport -->
                                <xs:attribute name="id" type="xs:NCName"/>
                                <!--  True if the server use a cookies -->
                                <xs:attribute name="needCookies" type="xs:boolean"/>
                                <!--
                                    Names of session properties. A session
properties must be neutrally handled
                                     (returned by the client if sent by the server
                                -->
```

```
                                    <xs:attribute name="sessionProperties" type="cid:
NCName-list"/>
                                </xs:complexType>
                            </xs:element>
                            <xs:any namespace="##other"/>
                        </xs:choice>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>

<!-- ********************************************************************
             SIMPLE TYPE LIBRARY
********************************************************************* -->

    <!-- List of uris (is attribute) -->
    <xs:simpleType name="uri-list">
        <xs:restriction>
            <xs:simpleType>
                <xs:list itemType="xs:anyURI"/>
            </xs:simpleType>
            <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- List of properties name (sessionProperties attribute) -->
    <xs:simpleType name="NCName-list">
        <xs:restriction>
            <xs:simpleType>
                <xs:list itemType="xs:NCName"/>
            </xs:simpleType>
            <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- request method token definition (get or post) -->
    <xs:simpleType name="getPost-token">
        <xs:restriction base="xs:token">
            <xs:enumeration value="GET"/>
            <xs:enumeration value="POST;application/x-www-form-urlencoded"/>
            <xs:enumeration value="POST;multipart/form-data"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- request method token definition (get, put or post) -->
    <xs:simpleType name="getPutPost-token">
        <xs:restriction base="xs:token">
            <xs:enumeration value="GET"/>
            <xs:enumeration value="PUT"/>
            <xs:enumeration value="POST"/>
            <xs:enumeration value="POST;multipart/form-data"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- list of properties storage method (header, querystring or post) -->
    <xs:simpleType name="postProp-list">
        <xs:restriction>
            <xs:simpleType>
                <xs:list>
```

```
                    <xs:simpleType>
                        <xs:restriction base="xs:token">
                            <xs:enumeration value="header"/>
                            <xs:enumeration value="queryString"/>
                            <xs:enumeration value="post"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:list>
            </xs:simpleType>
            <xs:minLength value="1"/>
        </xs:restriction>
    </xs:simpleType>

    <!-- ****************************************************************
             COMPLEXE TYPE LIBRARY
 ***************************************************************** -->

    <!-- localized informations (used to write human oriented doc) -->
    <xs:complexType name="localized-element" mixed="true">
        <xs:attribute ref="xml:lang"/>
    </xs:complexType>

    <!-- restriction element -->
    <xs:complexType name="restriction-element">
        <xs:sequence>
            <xs:element name="label" minOccurs="0" maxOccurs="unbounded" type="cid:
localized-element"/>
            <xs:element name="doc" minOccurs="0" maxOccurs="unbounded" type="cid:
localized-element"/>
        </xs:sequence>
        <xs:attribute name="name" use="required" type="xs:NCName"/>
        <xs:attribute name="is" type="cid:uri-list"/>
        <xs:attribute name="value" use="required" type="xs:string"/>
    </xs:complexType>

    <!-- meta element -->
    <xs:complexType name="meta-element">
        <xs:sequence>
            <xs:element name="label" minOccurs="0" maxOccurs="unbounded" type="cid:
localized-element"/>
            <xs:element name="doc" minOccurs="0" maxOccurs="unbounded" type="cid:
localized-element"/>
            <xs:element name="value" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType mixed="true"/>
            </xs:element>
        </xs:sequence>
        <xs:attribute name="name" use="required" type="xs:NCName"/>
        <xs:attribute name="is" type="cid:uri-list"/>
    </xs:complexType>

    <!-- common step attribute -->
    <xs:complexType name="step-commons">
        <xs:attribute name="url" use="required" type="xs:anyURI"/>
        <xs:attribute name="needMetas" type="cid:NCName-list"/>
        <xs:attribute name="useMetas" type="cid:NCName-list"/>
        <xs:attribute name="returnMetas" type="cid:NCName-list"/>
    </xs:complexType>

    <!-- Step element (used by interact and asyncStep) -->
    <xs:complexType name="regularStep-element">
```

```
        <xs:complexContent>
            <xs:extension base="cid:step-commons">
                <xs:attribute name="is" type="cid:uri-list"/>
                <xs:attribute name="required" type="xs:boolean"/>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <!-- Step element (used by upload and exchange) -->
    <xs:complexType name="asyncStep-element">
        <xs:complexContent>
            <xs:extension base="cid:regularStep-element">
                <xs:all>
                    <xs:element name="systemWait" minOccurs="0">
                        <xs:complexType>
                            <xs:complexContent>
                                <xs:extension base="cid:step-commons">
                                    <xs:attribute name="waitProperties" type="cid:
NCName-list"/>
                                </xs:extension>
                            </xs:complexContent>
                        </xs:complexType>
                    </xs:element>
                    <xs:element name="userWait" minOccurs="0" type="cid:step-commons"/>
                </xs:all>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>

    <!-- Definition of a regular request (exchange/interact) -->
    <xs:complexType name="regularRequest">
        <xs:attribute name="method" use="required" type="cid:getPost-token"/>
        <xs:attribute name="properties" use="required" type="cid:postProp-list"/>
    </xs:complexType>

    <!-- Definition of a request which contains -->
    <xs:complexType name="uploadRequest">
        <xs:attribute name="method" use="required" type="cid:getPutPost-token"/>
        <xs:attribute name="properties" use="required" type="cid:postProp-list"/>
    </xs:complexType>

    <!-- Definition of a transport property -->
    <xs:complexType name="property">
        <xs:attribute name="key" use="required" type="xs:NCName"/>
        <xs:attribute name="value" use="required" type="xs:NCName"/>
    </xs:complexType>
</xs:schema>
```