Specification: CID protocol

Version :	1.4
Date :	Apr 2016
Editor :	Kelis
Author(s) :	Thibaut Arribe contact: contact@cid-protocol.org
License :	CC-BY Kelis

Table of contents

Introduction

⚠ Warning

This section is non-normative.

Plenty of platforms and services are designed to handle documents. A quick overview must include the management platforms (such as Enterprise Content Management or Digital Asset Management platform), platforms which exploit these documents for a dedicated use (such as e-learning or MOOC platforms) and applications that produces the document (such as office applications, LCMS platforms or publishing chains). Depending of its purpose and its technological bases, each platform has its own way to handle documents.

Such particularities require the development of specific connectors to exchange documents⊙ between two platforms (or sometimes between two versions of platforms). Content Interactive Delivery (CID) aims to furnish a generic protocol to define these document exchanges in order to send or receive documents from any document system.

CID allows a server to define its document processes in a manifest and allows a client to download and execute this manifest. The main originality of the protocol is the specification of direct interactions between the user and the server platform. In order to be in conformance with the particularities of document handling of each system, a server could furnish a web page to allow the user to specify the document exchange details.

1. Design goals

⚠ Warning

This section is non-normative.

Purpose

The main purpose of the Content Interactive Delivery (CID) protocol is to delimit a common frame to define and execute documents transactions through the internet. It furnishes a XML schema for the definition of the server processes in a manifest and the implementation requirements for the client and the server.

Goal 1: semi-automated transaction

The main difficulty in the exchange of documents comes from the heterogeneity of systems which prevent any universal automation for the exchanges of documents. CID offers to circumvent these difficulties by proposing the definition of interactions between three kind of actors:

- Interaction between the client software and the server software
- Interaction between the user and the server software (through a web frame neutrally displayed by the client software)
- Interaction between the client software and the user (like any client software do it).

Goal 2: open and extensible

The CID protocol is open and extensible. "Open" because it is released under an open source license. "Extensible" because it defines the terms of its future extensions.

Goal 3: easy to integrate

The design is thought to:

- Allow the description of server processes already implemented by a specific server
- Allow the quick development of clients dedicated to one specific case
- Allow the use of a generic client

2. Definitions

Document transaction

A document transaction is an interaction between client and server software in order to transmit documents and/or their metadata. For example, a document transfer, the selection of a document public url, the retrieving of documents metadata such as the title or the author are considered as document transaction.

Server

A server is an application which is accessible through an intranet or the internet and which proposes document transactions.

Client

A client is an application which can access to an intranet or the internet and which needs to process document transactions with servers.

User

In this document, the user is the person which use the client involved in the document transaction.

3. Conformance

Server conformance

A compliant server must:

- expose a **manifest** that is conform to the CID schema [p.17] and accessible through a single unauthenticated HTTP GET request;
- support **processes** and **transports** defined in this manifest following the normative chapter [p.22] of this document.

Specific client conformance

A compliant specific client must support all the client side **processes** and **transports** of at least **one manifest** following the normative chapter [p.22] of this document.

Generic client conformance

A compliant generic client must support all the client side **processes** and **transports** of **any manifest** following the normative chapter [p.22] of this document.

4. Basic concepts

Warning

This section is non-normative.

This section describes how to conceive and implement a simple CID process such as the first examples of the Discover CID document*[http://www.cid-protocol.org/docs/discover/web]*.

What is a manifest?

A manifest is a XML file which defines the processes implemented by a server. It must be accessible through an unauthenticated *HTTP GET* request. The validity of an XML manifest could be controlled with the XML schema [p.17] of these specifications.

A manifest is composed of two main parts :

- A list of processes
- A list of transports

A process neutrally defines document transactions. It defines a list of steps without the technical information of the transport. The transport part could define several transport alternatives using the single transport family of these specifications: web transport; or using a dedicated transport family defined as an extension of these specifications.

Defining a basic process

Defining the steps

Let consider a simple process of file upload from a client to a server, it is possible to define this process in three steps :

- a pre-upload exchange to check the authentication before the main request;
- the file upload;
- a post-upload interaction between the client and the user.

The minimal valid definition of this process is :

```xml
<cid:process>
    <cid:exchange url="http://example.com/checkAuth" required="false"/>
    <cid:upload url="http://example.com/upload" required="true"/>
    <cid:interact url="http://example.com/interact" required="true"/>
</cid:process>
```

Each step must defines its *URL* and a *required* attribute. A required step must be processed by the client while a non-required simply may be processed by the client.

Defining the metadata

This process could use specific metadata such as the file name or the content type. It could also return other meta such as the id of the uploaded file or its public *URL*. All the meta used in the process must be defined at the beginning of the process definition.

These meta could then be called by the steps in three different attributes :

- ***useMetas***: which means the metadata may be sent by the client;
- ***needMetas***: which means the metadata must be sent by the client;
- ***returnMetas***: which means the metadata must be returned by the server at the end of the step.

In this case, the definition could look like behind.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cid:manifest xmlns:cid="http://www.cid-protocol/schema/v1/core">
    <cid:process>
        <cid:meta name="file-name"/>
        <cid:meta name="content-type"/>
        <cid:meta name="internal-id"/>
        <cid:meta name="public-url"/>
        <cid:exchange url="http://example.com/checkAuth" needMetas="content-type" required="false"/>
        <cid:upload url="http://example.com/upload" needMetas="content-type" useMetas="file-name" required="true" returnMetas="internal-id"/>
        <cid:interact url="http://example.com/interact" needMetas="internal-id" returnMetas="public-url" required="true"/>
    </cid:process>
</cid:manifest>
```

In this example, the server creates its own file name if the client does not provide it. At the end of the upload step, the server returns the internal id which must be sent back by the client in order to let the server builds the interaction GUI (in a conventional stateless way).

Documenting the process for the client software

There is no reserved name for metadata. The manifest could provide its own name for any meta. For example the content type could be called *contentType* or just *type* or whatever else.

To allow the client to understand these names, the manifest could describe the metadata by an *IRI* which formally qualifies the content. Such an *IRI* is written in an attribute called *is*. This attribute could be used on the *process*, *meta* and *step* elements.

The main needed *IRIs* could be find on schema.org*[http://www.schema.org]* for generic actions and concepts or on purl.org/dc*[http://www.purl.org/dc]* (the Dublin-Core) for the definition of specific meta .

Enriched with the appropriated *IRIs*, the manifest should look like behind.

```xml
<cid:process is="http://schema.org/SendAction">
    <cid:meta name="file-name" is="http://purl.org/dc/elements/1.1/title"/>
    <cid:meta name="content-type" is="http://www.w3.org/TR/html4/sgml/dtd.html#ContentType"/>
    <cid:meta name="internal-id" is="http://schema.org/productID"/>
    <cid:meta name="public-url" is="http://schema.org/URL"/>
    <cid:exchange url="http://example.com/checkAuth" needMetas="content-type" required="false" is="http://schema.org/AuthorizeAction"/>
    <cid:upload url="http://example.com/upload" needMetas="content-type" useMetas="file-name" required="true" returnMetas="internal-id"/>
    <cid:interact url="http://example.com/interact" needMetas="internal-id" returnMetas="public-url" required="true"/>
</cid:process>
```

The *is* attribute is used by the client to determine how to fill a meta or how to choose if an optional step need to be executed. Consequently, required steps not need for *IRIs*.

Documenting the process for human

A client which implement *IRI* recognition could automate the filling of the needed metadata. However, a client which does not implement this feature or which does not know the specific *IRIs* provided by the manifest could start this process alone.

In order to help the client to build GUI dedicated to the users, the manifest could also contain human-oriented documentation.

```
<cid:label xml:lang="en">CID manifest of the ECM service of my Example Company</cid:label>
<cid:process is="http://schema.org/SendAction">
    <cid:label xml:lang="en">Send a new file</cid:label>
    <cid:doc xml:lang="en">Send a new file into the ECM platform. This action allows you to send directly your files without using a classical webupload.</cid:doc>
    <cid:meta name="file-name" is="http://purl.org/dc/elements/1.1/title">
        <cid:label xml:lang="en">File name</cid:label>
    </cid:meta>
    <cid:meta name="content-type" is="http://www.w3.org/TR/html4/sgml/dtd.html#ContentType">
        <cid:label xml:lang="en">Content Type</cid:label>
         <cid:doc xml:lang="en">The type of the uploaded file writen following the MIME standard (RFC 2045). For example application/xml</cid:doc>
    </cid:meta>
    <cid:meta name="internal-id" is="http://schema.org/productID">
        <cid:label xml:lang="en">Internal identifier</cid:label>
<    /cid:meta>
    <cid:meta name="public-url" is="http://schema.org/URL">
        <cid:label xml:lang="en">Public URL</cid:label>
        <cid:doc xml:lang="en">The URL where the uploaded content could be retrieved.</cid:doc>
    </cid:meta>
    <cid:exchange url="http://example.com/checkAuth" needMetas="content-type" required="false" is="http://schema.org/AuthorizeAction"/>
    <cid:upload url="http://example.com/upload" needMetas="content-type" useMetas="file-name" required="true" returnMetas="internal-id"/>
    <cid:interact url="http://example.com/interact" needMetas="internal-id" returnMetas="public-url" required="true"/>
</cid:process>
```

Defining the transport

The second part of the manifest is dedicated to the transport. Theoretically, it is possible to define any application layer following the OSI model (HTTP, FTP, SMTP, etc.). However, these specifications define only a generic web transport (so with HTTP request). The transport definition include also the authentication scheme of the process. In this example, the server accepts basic authenticated requests (see RFC 2617*[https://www.ietf.org/rfc/rfc2617.txt]* for more details).

A manifest which define a kind of step (interact, exchange, upload) must also define the transport modalities of this step. It is possible to define several possibilities for each kind of step. A transport-generic server should include :

```
<cid:transports>
    <cid:webTransport>
        <cid:authentications>
            <cid:basicHttp/>
        </cid:authentications>

        <cid:webExchange>
            <request method="GET" properties="header queryString"/>
            <request method="POST;application/x-www-form-urlencoded" properties="queryString header post"/>
            <request method="POST;multipart/form-data" properties="header queryString post"/>
        </cid:webExchange>

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

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

The definition of the *HTTP* method *POST* must be followed by the content type of the form when such a form is used. The *properties* attribute list the possibilities of metadata storage :

- *header* means that the server accepts the metadata in the header of the *HTTP* request.

  A "text/plain" value of the "content-type" meta should be inserted as "content-type":"text/plain" inside the *HTTP* header.

- *queryString* means that the server accepts the metadata in the *URL* as query string (see RFC 3986*[https://tools.ietf.org/html/rfc3986]* section 3.4 for more details).

  A "text/plain" value of the "content-type" meta should be inserted to the *URL* as "(?|&)content-type=text/plain".

- *post* means that the server accepts the metadata in the form of a POST request.

  The value is stored in field which share its name with the name of the meta ("content-type" in this example).

Implementation details

The complete manifest can be seen behind. It must be exposed by the server and accessible through a single *HTTP GET* request without any authentication.

```
<?xml version="1.0" encoding="UTF-8"?>
<cid:manifest xmlns:cid="http://www.cid-protocol.org/schema/v1/core">
    <cid:label xml:lang="en">CID manifest of the ECM service of my Example Company</cid:
label>
    <cid:process is="http://schema.org/SendAction">
        <cid:label xml:lang="en">Send a new file</cid:label>
        <cid:doc xml:lang="en">Send a new file into the ECM platform. This action
allows you to send directly your files
            without using a classical webupload.
        </cid:doc>
        <cid:meta name="file-name" is="http://purl.org/dc/elements/1.1/title">
            <cid:label xml:lang="en">File name</cid:label>
        </cid:meta>
        <cid:meta name="content-type" is="http://www.w3.org/TR/html4/sgml/dtd.
html#ContentType">
            <cid:label xml:lang="en">Content Type</cid:label>
            <cid:doc xml:lang="en">The type of the uploaded file writen following the
MIME standard (RFC 2045). For
```

```
                example application/xml
            </cid:doc>
        </cid:meta>
        <cid:meta name="internal-id" is="http://schema.org/productID">
            <cid:label xml:lang="en">Internal identifier</cid:label>
        </cid:meta>
        <cid:meta name="public-url" is="http://schema.org/URL">
            <cid:label xml:lang="en">Public URL</cid:label>
            <cid:doc xml:lang="en">The URL where the uploaded content could be retrieved.</cid:doc>
        </cid:meta>
        <cid:exchange url="http://example.com/checkAuth" needMetas="content-type" required="false"
                      is="http://schema.org/AuthorizeAction"/>
        <cid:upload url="http://example.com/upload" needMetas="content-type" useMetas="file-name" required="true"
                    returnMetas="internal-id"/>
        <cid:interact url="http://example.com/interact" needMetas="internal-id" returnMetas="public-url"
                      required="true"/>
    </cid:process>

    <cid:transports>
        <cid:webTransport>
            <cid:authentications>
                <cid:basicHttp/>
            </cid:authentications>

            <cid:webExchange>
                <request method="GET" properties="header queryString"/>
                <request method="POST;application/x-www-form-urlencoded" properties="queryString header post"/>
                <request method="POST;multipart/form-data" properties="header queryString post"/>
            </cid:webExchange>

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

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

The client must download the manifest to analyze and execute the process.

1. The client could begin the transaction by an authenticated exchange request containing the ***content-type*** meta. It could support one of the defined transport possibilities :
    - A *HTTP GET* method with metadata stored in the header of the request or in the *URL* as query string.
    - A *HTTP POST* method containing a URL encoded form. The metadata could be stored in the

header, in the URL as query string or in the form.

- A *HTTP POST* method containing a multipart form. The metadata could be stored in the header, in the URL as query string or in the form.

The server must support any of the configuration written in the manifest.

2. The client must then upload the document in a single authenticated request containing the content-type meta and optionally the file name. It could support one of the defined transports possibilities :

   - A *HTTP PUT* method with the file in the body and the meta in the header of the request or in the URL as query string.
   - A *HTTP POST* method with the file in the body and the meta in the header of the request or in the URL as query string.
   - A *HTTP POST* method with the file in a *cidContent* field and the meta in the form, in the header of the request or in the URL as query string.

   Note that it is not possible to send binary files in a URL encoded form.

   The server must support any of the configuration written in the manifest.

   The server must include the returned meta in the response. With a web transport and for the exchange and upload steps, a returned meta must be inserted in a javascript object in the body of the response (for example: {"internal-id":"0X001242"}).

3. The client must then begin an interaction between the user and the server. The client must send the first request, which must be authenticated and which must contain the internal-id meta. The client must support one of the following transport methods for the first request:

   - A *HTTP GET* method with metadata stored in the header of the request or in the URL as query string.
   - A *HTTP POST* method containing a URL encoded form. The metadata could be stored in the header, in the URL as query string or in the form.
   - A *HTTP POST* method containing a multipart form. The metadata could be stored in the header, in the URL as query string or in the form.

   The server must support any of the configuration written in the manifest.

   The server must include a HTML page in the body of the response. The client must show this page in a web frame. The user could now interact directly with the server through this web frame.

   At the end of the interaction (for example, by clicking on "validation" button), the HTML page must send a custom event called "cid-interaction-ended". The body of this event must embed a javascript object containing the returned metadata (the public url). The web page could also throw a custom event called "cid-interaction-aborted" to express the failure of the interaction. This second event does not require any body.

5. Advanced Concepts

5.1. Process

Warning

This section is non-normative.

Defining several processes in a single manifest

The manifest schema [p.17] of this specification allows the definition several processes. They are defined one by one in the first part of the manifest.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cid:manifest xmlns:cid="http://www.kelis.fr/cid/v2/core">
    <cid:process>...</cid:process>
    <cid:process>...</cid:process>
    <cid:process>...</cid:process>
    <cid:transports>...</cid:transports>
</cid:manifest>
```

5.2. Transport

Warning

This section is non-normative.

Asynchronous upload

An asynchronous upload is a specific upload declared by the server when the time needed to process the uploaded file does not allow a synchronous response. The server could then define a user-oriented and/or a system oriented wait system.

- The user-oriented system reuse the interaction mechanism.
- The system-oriented system reuse the exchange mechanism.

User and system oriented wait mechanism must declare a *url* attribute and could declare the *needMetas*, *useMetas* and *returnMetas* step attributes.

```
<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>
```

To implement a system wait, the client must send a HTTP GET requests until it gets a 200 response code. A negative response must contains a 202 response code.

Allowing statefull server

A statefull server need to know which session is related to a request. To perform this, CID provides two principles: cookies and session properties.

Cookies

A web transport definition could specify in the manifest that the client must support cookies to complete the transaction. To perform this, the *webTransport* element must contain a *needCookies* attribute set to true. By default (without any attribute) the value is considered as false.

```
<cid:webTransport needCookies="true">
    ...
</cid:webTransport>
```

Session properties

A session property is a pair of key, value which are sent and received following the same scheme than the metadata. The key of the session properties must be declared in a dedicated attribute of the *webTransport* element. A client which operate a process through such a transport must check if the defined property is returned by the server at each step. When this is the case, the client must send back this property.

```
<cid:webTransport sessionProperties="session-id">
   ...
</cid:webTransport>
```

It is possible to declare several session properties in the attribute.

Defining several transports in a single manifest

As for the processes or the authentication, it is possible to define several transport. All the transport should be inserted as a new child of the *transports* element.

By default, a client could use any process with any transport. It is possible to dedicate a process to a specific transport by the use of ids. A non-generic transport must declare its *id* in an attribute. A process could bind this transport by the use of the *transports* attribute.

```xml
<cid:process transports="sessionPropTrans">
...
</cid:process>
...
<cid:transports>
    <cid:webTransport id="sessionPropTrans" sessionProperties="session-id">
    ...
    </cid:webTransport>
</cid:transports>
```

5.3. Extending CID

Warning

This section is non-normative.

The CID protocol specifies the modalities of its extension. The schema [p.17] allows the insertion of unknown elements in the *authentication* and *transport* elements.

A CID extension is characterized by a new schema defining the new XML elements and a specifications document which defines the required implementations.

6. 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>
```

7. Implementation requirements

7.1. General requirements

Manifest handling

Validity of the manifest

A valid manifest MUST be conform to the manifest schema [p.17]. Even a CID extension CAN NOT require an improper manifest.

Server exposition

The manifest MUST be accessible through a single unauthenticated HTTP GET request.

Client retrieving

The downloading of the manifest is the first step of the transaction. The client MUST download a new manifest at the beginning of each process.

Process and transport selection

Once the manifest is downloaded by the client, this one MUST select one process and one transport to complete the transaction.

Default behavior

A manifest defines a list of processes and a list of transports. By default, all the processes MUST be compatible with any of the transports, which mean that all the transports MUST defines all the requests defined in the processes (webExchange, webUpload and webInteract).

Example

Lets consider a manifest that contains two processes defined as following :

```
<cid:process>
    <cid:label>Upload with post interation</cid:label>
    <cid:upload url="http://example.com/upload" required="true"/>
    <cid:interact url="http://example.com/interact" required="true"/>
</cid:process>
<cid:process>
<cid:label>Direct upload</cid:label>
<cid:upload url="http://example.com/upload" required="true"/>
</cid:process>
```

All the transports of the manifest MUST defines one or more webUload requests AND one or more webInteract requests.

Therefore, the following manifest is valid :

```
<?xml version="1.0" encoding="UTF-8"?>
<cid:manifest xmlns:cid="http://www.cid-protocol.org/schema/v1/core">
    <cid:process>
        <cid:label>Upload with post interation</cid:label>
        <cid:upload url="http://example.com/upload" required="true"/>
        <cid:interact url="http://example.com/interact" required="true"/>
    </cid:process>
    <cid:process>
        <cid:label>Direct upload</cid:label>
        <cid:upload url="http://example.com/upload" required="true"/>
    </cid:process>

    <cid:transports>
        <cid:webTransport>
            <cid:authentications>
                <cid:basicHttp/>
            </cid:authentications>

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

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

And the following one is not valid.

```
<?xml version="1.0" encoding="UTF-8"?>
<cid:manifest xmlns:cid="http://www.cid-protocol.org/schema/v1/core">
    <cid:process>
        <cid:label>Upload with post interation</cid:label>
        <cid:upload url="http://example.com/upload" required="true"/>
        <cid:interact url="http://example.com/interact" required="true"/>
    </cid:process>
    <cid:process>
        <cid:label>Direct upload</cid:label>
        <cid:upload url="http://example.com/upload" required="true"/>
    </cid:process>

    <cid:transports>
        <cid:webTransport>
            <cid:authentications>
                <cid:basicHttp/>
            </cid:authentications>

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

            <cid:webUpload>
                <request method="PUT" properties="header queryString"/>
                <request method="POST" properties="header queryString"/>
```

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

        <cid:webTransport>
            <cid:authentications>
                <cid:basicHttp/>
            </cid:authentications>

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

Transport id

All the transports definition could include an *id* attribute. This id could be called in the *transports* attribute of a *process* to restrain the compatibility to one or more specific transports definition.

Example

With an *id* and *transports* attributes, the previously not valid manifest could be restrained as following.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cid:manifest xmlns:cid="http://www.cid-protocol.org/schema/v1/core">
    <cid:process transports="myTransportId">
        <cid:label>Upload with post interation</cid:label>
        <cid:upload url="http://example.com/upload" required="true"/>
        <cid:interact url="http://example.com/interact" required="true"/>
    </cid:process>
    <cid:process>
        <cid:label>Direct upload</cid:label>
        <cid:upload url="http://example.com/upload" required="true"/>
    </cid:process>

    <cid:transports>
        <cid:webTransport id="myTransportId">
            <cid:authentications>
                <cid:basicHttp/>
            </cid:authentications>

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

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

        <cid:webTransport>
            <cid:authentications>
                <cid:basicHttp/>
            </cid:authentications>

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

Select a pair

Once determined the list of valid pairs of process and transport, the client must choose one of them. The way this choice is done is out of the scope of the CID protocol. Basically, a client could build a GUI for its user using the *labels* and *docs* of the manifest or compute an automatic choice using the *is* attributes.

7.2. Process definition

Minimal process definition

A document transaction available on a server is the composition of several steps in order to achieve one goal such as document upload, metadata retrieving, document selection, etc. A server is defined by one *process* element in the manifest.

As explained in the general requirements [p.22], a process must be compatible with all the transport or specifically bind one or more compatible processes with the *transports* attribute.

Warning

The minimal process definition must contains at least one step as showed behind.

Example

```
<cid:process>
    <cid:upload url="http://example.com/upload" required="true"/>
</cid:process>
```

Describe a process

A server may describe its processes. There is two ways to do this description.

System oriented : one or more IRI can be defined in the *is* attribute. These IRI allow a client to automate the process recognition and selection.

User oriented : one or more localized *label* and *doc* attributes can be used to describe the process.

Example

A fully described minimal process could be like following.

```
<cid:process is="http://schema.org/SendAction">
    <cid:label xml:lang="en">Upload a document onto Example.com server</cid:label>
    <cid:doc xml:lang="en">This process allows a client to upload any resource file
onto our Example.com server. Once uploaded, the file will be automatically located into
the resources folder of your account
    </cid:doc>
    <cid:upload url="http://example.com/upload" required="true"/>
</cid:process>
```

Metadata

Any metadata used by a step of the process MUST be previously declared. A used metadata is a metadata called by the *needMetas*, *useMetas* and *returnMetas* attributes of the steps.

Minimal metadata declaration

The minimal declaration of a metadata is composed of a single element *meta* and its *name* attribute.

Example

```
<cid:meta name="title"/>
```

The *name* attribute is required because its content is used by the *needMetas*, *useMetas* and *returnMetas* attributes of the step.

Describe a meta

A meta could be described in a user-oriented way by *label* and *doc* elements and in system-oriented way by the *is* attribute.

Example

```
<cid:meta name="title" is="http://purl.org/dc/elements/1.1/title https://schema.
org/title">
    <cid:label xml:lang="en">Title of the document</cid:label>
    <cid:label xml:lang="fr">Titre du document</cid:label>
    <cid:doc xml:lang="en">This meta should contain the title of document and
not name of the file</cid:doc>
    <cid:doc xml:lang="fr">Cette meta devrait contenir le titre du document et
non le nom du fichier</cid:doc>
</cid:meta>
```

Warning

The descriptions elements does not change any client or server implementation requirement.

Restrain the possible values of a meta

A server could restrain the valid values of a meta in the manifest. This restriction is done with *value* elements containing the proposed values.

Example

```
<cid:meta name="Content-type" is="http://purl.org/dc/elements/1.1/type">
    <cid:label xml:lang="en">Available archive formats</cid:label>
    <cid:value>application/zip</cid:value>
    <cid:value>application/jar</cid:value>
    <cid:value>application/x-tar</cid:value>
```

```
</cid:meta>
```

Warning

At this stage of the process declaration, client and server does not have any implementation requirements.

Restriction

A restriction is an implicit meta whose value is set by the server. The client should not send the value but must consider the meta and its value as required for the execution of the process.

Minimal restriction declaration

A minimal restriction declaration contains a *restriction* element, the *name* attribute and a *value* attribute.

Example

```
<cid:restriction name="Zip-file-names-encoding" value="UTF-8"/>
```

Describe a restriction

A restriction could contain the user oriented (*localized* label and *doc* elements) or system oriented (IRIs in the *is* attribute) description.

Steps definition

The steps are the main part of the process. They define how the client and the server should communicate to process the document transaction. One step represents an exchange of information between the client and the server.

Each step could be defined using the same attributes :

- *required* : contains a boolean which indicates if the processing of this step is mandatory or not.

  Mandatory attribute
- *url* : contain the url of the server where the client must send the request in order to start the processing of this step.

  Mandatory attribute
- *useMetas* : contains the names (the names declared in the *meta* elements) of the optional metadata.

  Optional attribute
- *needMetas* : contains the names (the names declared in the *meta* elements) of the required metadata.

  Optional attribute
- *returnMetas* : contains the names (the names declared in the *meta* elements) of the metadata returned by the server once the step processed.

  Optional attribute
- *is* : contains the IRIs that qualifies the step.

  Optional attribute

Exchange step

An exchange step define a regular request from the client to the server. The user is not directly involved in this transaction.

The technical details of the implementation is defined in the transport section [p.29].

Declaration example

```
<cid:exchange url="http://example.com/checkAuthorization" is="http://schema.org
/AuthorizeAction" required="false"/>
```

Interact step

An interact step define an exchanges between the user and the server, then between the server and the client.

The technical details of the implementation is defined in the transport section [p.29].

Declaration example

```
<cid:interact url="http://example.com/interact-pre-delivery" needMetas="dc-title"
 useMetas="dc-creator dc-modified" returnMetas="dc-title dc-creator dc-modified
tags" required="true"/>
```

Upload step

An upload step define an exchanges between the client and the server .

The technical details of the implementation is defined in the transport section [p.29].

Declaration example

```
<cid:upload url="http://example.com/upload.php" useMetas="File-name" returnMetas=
"Public-url" required="true"/>
```

Asynchronous upload

An asynchronous upload is a specific upload declared by the server when the time needed by the processing of the uploaded file does not allow a synchronous response. The server could then define a user-oriented and/or a system oriented wait system.

- The user-oriented system reuse the interaction mechanism.
- The system-oriented system reuse the exchange mechanism. The client must execute the wait step until a valid server response.

User and system oriented wait mechanism must declare a *url* attribute and could declare the *needMetas* , *useMetas* and *returnMetas* step attributes.

Declaration example

```
<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>
```

7.3. Transport : web transport

Once an available process selected by the client, the transport declaration defines how client and server must communicate together. The first section [p.22] of this chapter defines how a client must determine process/transport compliance.

The selection of a process/transport couple is left to the client. It could automate the selection or build a GUI to delegate this choice to the user.

These specification only defines one transport possibilities : web transport which use a various web standard such as HTTP requests, HTTP authentication or cookies.

Cid extensions could defines new transport modalities (see the next section [p.35] for more details).

Metadata and properties

Transports must define a way to communicate properties between client and server. A property is a basic pair of key-value. Metadata are considered as document properties. A transport could define its own properties.

web transport attributes

A web transport declaration could contains three attributes:

- an *id* attribute to bind processes to one or more specific transports (see the first section [p.22] of this chapter for more details);
- a *sessionProperties* attribute;
- a *needCookies* attribute.

Session properties

The sessionProperties attribute must contain a list of one or more properties names. A session property could be returned by the server at any step of the process and must then be sent by client for the following steps.

Cookies declaration

The *needCookies* attribute must contain a boolean value. A client which executes a process through a web transport that needs cookies itself must support this standard.

Authentication

The *authentication* element is required in the web transport definition. It could be empty or contain one or more of basic, web (or no authentication) elements.

An empty authentication element means that no authentication scheme is required to use this transport definition. When this element has one or more children, the client must respect the requirement of one of them in order to use this transport definition.

Basic HTTP authentication

Declaration

The declaration of a Basic HTTP authentication contains only the *basicHttp* element.

```
<cid:authentications>
    <cid:basicHttp/>
</cid:authentications>
```

Implementation

When a web transport declaration contains a basic HTTP authentication scheme, the server must accept basic HTTP authentication for each request that could regularly be sent with this transport following process/transport compatibility defined in the first section [p.22] of this chapter.

When a client select a Basic HTTP authentication scheme, it must send a *authorization* header in each request of the selected process.

Web authentication

Declaration

The declaration of a web authentication contains a *webAuthentication* element and an *URL* attribute.

```
<cid:authentications>
  <cid:webAuthentication  url="http://example.com/auth"/>
</cid:authentications>
```

Implementation

The server must furnish an authentication web page at the location specified in the URL attribute.

This page must be loaded by the client following the web interaction transport modalities (see further on this section).

At the end of the web authentication interaction, the server must send a message (following the HTML 5 post message API) containing a *cidAuth* property in the data part of the message. The authorized values of this property are *succeeded* and *failed*. This message must be considered by the client as the end interaction message (see web interact for further detail on end interaction mechanism).

⚠ Warning

A web authentication required the definition of at least one web interaction request.

No authentication

Declaration

The declaration of a public access is declared by a single *noAuthentication* element.

```
<cid:authentications>
    <cid:webAuthentication url="http://example.com/auth"/>
    <cid:noAuthentication/>
</cid:authentications>
```

Implementation

The *noAuthentication* element does not required any specific implementations for the server or client side.

Example

The *noAuthentication* element allows a manifest to declare only one transport for either a public and a restricted access.

Web exchange

The *webExchange* element defines how the exchange steps and system-oriented wait must be processed by the client and the server. The definition of a *webExchange* element is composed of several request definitions. Each request must define a HTTP method and a way to send params.

Defining a request

In a web exchange context, the method attribute of a request could have the *GET*, *POST;application/x-www-form-urlencoded* or *POST;multipart/form-data* values. It defines the HTTP method that the client must use to communicate with the server. In the case of a POST HTTP request, the method contains also the kind of form that is available to the client.

The properties attribute could contains the *header* and *queryString* values for a GET HTTP request plus the *post* values for the POST HTTP requests.

Example of full webExchange declaration

This example is called full because the server implements all the possibilities of exchange transports.

```
<cid:webExchange>
    <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:webExchange>
```

Implementation requirement

In order to process an exchange step, the client must select one of the defined requests. The client must then select a method to send the properties in the *properties* attribute of the selected request.

Client requirement

properties\HTTP method	GET	POST;application/x-www-form-urlencoded	POST;multipart/form-data
header	the client must send a HTTP GET request and store the key/value of all the properties in the header of the request.	the client must send a HTTP POST request and store the key/value of all the properties in the header of the request.	the client must send a HTTP POST request and store the key/value of all the properties in the header of the request.
queryString	the client must send a HTTP GET request and store the key/value of all the properties in the URL as query string.	the client must send a HTTP POST request and store the key/value of all the properties in the URL as query string.	the client must send a HTTP POST request and store the key/value of all the properties in the URL as query string.
post		the client must send a HTTP	the client must send a HTTP

		POST request and store the key/value of all the properties in a url-encoded form.	POST request and store the key/value of all the properties in a form-data.

The optional metadata or session properties returned by the server must be sent in the body or the response in a json object.

Web Interact

The *webInteract* element defines how the interact steps and user-oriented wait must be processed by the client and the server. The definition of a *webInteract* element is composed of several request definitions. Each request must define a HTTP method and a way to send params.

Defining a request

In a web interact context, the method attribute of a request could have the *GET*, *POST;application/x-www-form-urlencoded* or *POST;multipart/form-data* values. It defines the HTTP method that the client must use to communicate with the server. In the case of a POST HTTP request, the method contains also the kind of form that is available to the client.

The properties attribute could contains the *header* and *queryString* values for a GET HTTP request plus the *post* values for the POST HTTP requests.

Example of full webInteract declaration

This example is called full because the server implements all the possibilities of exchange transports.

```
<cid:webExchange>
    <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:webExchange>
```

Implementation requirement

In order to process an interact step, the client must select one of the defined requests. The client must then select a method to send the properties in the *properties* attribute of the selected request.

Client requirement

properties\HTTP method	GET	POST;application/x-www-form-urlencoded	POST;multipart/form-data
header	the client must send a HTTP GET request and store the key/value of all the properties in the header of the request.	the client must send a HTTP POST request and store the key/value of all the properties in the header of the request.	the client must send a HTTP POST request and store the key/value of all the properties in the header of the request.
queryString	the client must send a HTTP GET request and store the key/value of all	the client must send a HTTP POST request and store the key/value of all the	the client must send a HTTP POST request and store the key/value of all the properties

	the properties in the URL as query string.	properties in the URL as query string.	in the URL as query string.
post		the client must send a HTTP POST request and store the key/value of all the properties in a url-encoded form.	the client must send a HTTP POST request and store the key/value of all the properties in a form-data.

The result of the server must be a web page neutrally displayed in a web frame by the client. The client must then let the user in direct interaction with the server. Once the interaction is done, the web page must send an message through the postMessage API. This message must contain the metadata and properties returned by the server and a *cidInteraction* property set to *ended* or *aborted* :

- The value *aborted* means that the user wants to cancel the process and the client should close its CID GUI. The client can't continue the process after and *aborted cidInteraction* value.
- The value *ended* means that the interaction was regularly closed. The client must close the interaction and continue the process.

Web Upload

The *webUpload* element defines how upload steps and user-oriented wait must be processed by the client and the server. The definition of a *webUpload* element is composed of several request definitions. Each request must define a HTTP method and a way to send params.

Defining a request

In a web interact context, the method attribute of a request could have the *GET*, *PUT*, *POST* or *POST; multipart/form-data* values. It defines the HTTP method that the client must use to communicate with the server. In the case of a POST HTTP request, the method contains also the kind of form that is available to the client.

The properties attribute could contains the *header* and *queryString* values for a GET or PUT HTTP request plus the *post* values for the *POST;multipart/form-data* HTTP request.

Example of full webUpload declaration

This example is called full because the server implements all the possibilities of exchange transports.

```
<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>
```

Implementation requirement

In order to process an upload step, the client must select one of the defined requests. The client must then select a method to send the properties in the *properties* attribute of the selected request.

Client requirement

properties\HTTP method	PUT	GET	POST	POST;multipart/form-data
header	the client must send a HTTP PUT request containing the file to upload in the request body and store the key /value of all the properties in the header of the request.	the client must send a HTTP GET request containing the file to upload in the request body and store the key /value of all the properties in the header of the request.	the client must send a HTTP POST request containing the file to upload in the request body and store the key/value of all the properties in the header of the request.	the client must send a HTTP POST request containing a form data which store the file to upload in a ***cidContent*** field and store the key /value of all the properties in the header of the request.
queryString	the client must send a HTTP PUT request containing the file to upload in the request body and store the key /value of all the properties in the URL as query string.	the client must send a HTTP GET request containing the file to upload in the request body and store the key /value of all the properties in the URL as query string.	the client must send a HTTP POST request containing the file to upload in the request body and store the key/value of all the properties in the URL as query string.	the client must send a HTTP POST request containing a form data which store the file to upload in a ***cidContent*** field and store the key /value of all the properties in the URL as query string.
post				the client must send a HTTP POST request containing a form data which store the file to upload in a ***cidContent*** field and the key/value of all the properties.

The optional metadata or session properties returned by the server must be sent in the body or the response in a json object.

Asynchronous upload

When the process declares an asynchronous upload, the server could return an HTTP 202 code to the upload request. The client must choose one of the system-oriented or user-oriented defined in the process.

A **user-oriented** wait must be processed exactly as an interaction step just after the upload. The URL is defined by the process and the transport layer must be conform to the webInteract declaration.

To process a **system-oriented** wait, the client must repeat a request to the URL defined in the process following the webExchange declaration while the server is responding an HTTP 202 response code. The final request must be responded by a 200 or server error response code. The interval between the requests should be chosen by the client.

7.4. Cid extension

The CID protocol take the part of a strong process/transport distinction in the design of its extension possibilities. The process part is designed to be generic and reused in any context. The transport part could be extended by the addition of new kind of transport.

An valid CID extension must contains at least the schema of the extended manifest and the implementation requirement of the new transport.

Note

The manifest schema of CID, included in these specifications, defines where CID extensions should be written in the manifest : in the *transports* element.

8. References

- URL : RFC 1738*[https://www.ietf.org/rfc/rfc1738.txt]*
- HTTP: RFC 2616*[http://www.ietf.org/rfc/rfc2616.txt]*
- HTTP Authentication : RFC 2617*[https://www.ietf.org/rfc/rfc2617.txt]*
- HTTP State Management Mechanism : RFC 2109*[https://www.ietf.org/rfc/rfc2109.txt]*
- IRI : RFC 3987*[https://tools.ietf.org/html/rfc3987]*
- XML*[http://www.w3.org/standards/xml/core]*
- XML Schema*[http://www.w3.org/standards/xml/schema]*
- JSON*[http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf]*