No part of this document may be reproduced, in whole or in part, without the express written permission of Tecnoteca s.r.l.

CMDBuild ® uses many great technologies from the open source community: PostgreSQL, Apache, Tomcat, Eclipse, Ext JS, JasperReports, IReport, Enhydra Shark, TWE, OCS Inventory, Liferay, Alfresco, GeoServer, OpenLayers, Prefuse, Quartz, BiMserver.
We are thankful for the great contributions that led to the creation of these products.

CMDBuild ® is a project of Tecnoteca Srl. Tecnoteca is responsible of software design and development, it's the official maintainer and has registered the CMDBuild logo.

In the project also the Municipality of Udine was involved as the initial customer.

CMDBuild ® is released under AGPL open source license (http://www.gnu.org/licenses/agpl-3.0.html)

CMDBuild ® is a registered trademark of Tecnoteca Srl. Everytime the CMDBuild® logo is used, the official maintainer "Tecnoteca srl" must be mentioned; in addition, there must be a link to the official website: http://www.cmdbuild.org.

CMDBuild ® logo:
- cannot be modified (color, proportion, shape, font) in any way, and cannot be integrated into other logos
- cannot be used as a corporate logo, nor the company that uses it may appear as author / owner / maintainer of the project
- cannot be removed from the application, and in particular from the header at the top of each page

The official website is http://www.cmdbuild.org
Contents

Introduction...................................................................................................................................... 4
  CMDBuild modules.................................................................................................................... 4
  Available documentation......................................................................................................... 5
Interoperability standards........................................................................................................ 6
  Service-Oriented Architecture (SOA)........................................................................................ 6
Web service and SOAP protocol................................................................................................. 7
  Examples of usage.................................................................................................................. 7
  Provided services................................................................................................................... 7
  Description of support data structures.................................................................................. 10
  Description of web service methods........................................................................................ 13
  Example of creating a “client”................................................................................................. 16
  Definition of authentication policy file................................................................................... 18
REST web service....................................................................................................................... 20
  Services.......................................................................................................................................... 20
APPENDIX: Glossary.................................................................................................................. 110
Introduction

CMDBuild is an Open Source web application designed to model and manage assets and services controlled by the ICT Department, therefore it handles the related workflow operations, if necessary according to ITIL best practices.

The management of a Configuration Database (CMDB) means keeping up-to-date, and available to other processes, the database related to the components in use, their relations and their changes over time.

With CMDBuild, the system administrator can build and extend its own CMDB (hence the project name), modeling the CMDB according to the company needs; the administration module allows you to progressively add new classes of items, new attributes and new relations. You can also define filters, "views" and access permissions limited to rows and columns of every class.

CMDBuild provides complete support for ITIL best practices, which have become a "standard de facto" by now, a non-proprietary system for services management with process-oriented criteria.

Thanks to the integrated workflow engine, you can create new workflow processes with external visual editors, and import / execute them inside the CMDBuild application according to the configured automatisms.

A task manager integrated in the user interface of the Administration Module is also available. It allows to manage different operations (process starts, e-mail receiving and sending, connector executions) and data controls on the CMDB (synchronous and asynchronous events). Based on their findings, it sends notifications, starts workflows and executes scripts.

CMDBuild includes also JasperReports, an open source report engine that allows you to create reports; you can design (with an external editor), import and run custom reports inside CMDBuild.

Then it is possible to define some dashboards made up of charts which immediately show the situation of some indicators in the current system (KPI).

CMDBuild integrates Alfresco, the popular open source document management system. You can attach documents, pictures and other files.

Moreover, you can use GIS features to georeference and display assets on a geographical map (external map services) and / or an office plan (local GeoServer) and BIM features to view 3D models (IFC format).

The system includes also a SOAP and a REST webservice, to implement interoperability solutions with SOA.

CMDBuild includes two frameworks called Basic Connector and Advanced Connector, which are able - through the SOAP webservice - to sync the information recorded in the CMDB with external data sources, for example through automatic inventory systems (such as the open source OCS Inventory) or through virtualization or monitoring systems.

Through the REST webservice, CMDBuild GUI Framework allows to issue custom webpages on external portals able to interact with the CMDB.

A user interface for mobile tools (smartphones and tablets) is also available. It is implemented as multi-platform app (iOS, Android) and linked to the CMDB through the REST webservice.

CMDBuild modules

The CMDBuild application includes two main modules:
• the Administration Module, used to define the data model and set config options (classes and relations, users and permissions, reports and workflows, main options and preferences)
• the Management Module, used to manage cards and relations, add attachments, run workflow processes, visualize dashboards and execute reports

The Administration Module is available only to the users with the "administrator" role; the Management Module is used by all the users who view and edit data.

Available documentation

This manual is for computer technicians, who manage external applications and who are interested in interoperating with CMDBuild. It supplies the detail information necessary to implement the communication system for the read-write access of its data and functions.

You can find all the manuals on the official website (http://www.cmdbuild.org):
• system overview ("Overview Manual")
• system administration ("Administrator Manual")
• system usage ("User Manual")
• installation and system management ("Technical Manual")
• webservice details and configuration ("Webservice Manual")
• connectors to sync data through external systems ("ConnectorsManual")
Interoperability standards

Service-Oriented Architecture (SOA)

In order to make different applications interoperable, they must be created as components that cooperate with the services implementation, and these services must be set through high level interfaces defined under standard protocols.

CMDBuild is designed with Service-Oriented Architecture (SOA):

- decoupling the different logic levels (see the schema)
- implementing and setting in every interface external specifications as a single modality for the access to relating data and methods
- using the interfaces both for the interactive access of the web client and for the programmatic access of external applications

From a technical point of view, we chose to use the following technology of web services:

1. SOAP protocol
2. REST protocol

Through web services, and safety policy permitting, CMDBuild provides the data filed in the CMDB and its management methods to allow the use within other applications involved with the information itself, both for the technical management and for administration.
Web service and SOAP protocol

A web service is an interface that describes a collection of methods, available over a network and working using XML messages.

With web services, an application allows other applications to interact with its methods.

SOAP is a standard protocol, based on XML, to access a web service.

The SOAP specifications are standardized by W3C.

The web service solution offers important architecture advantages:

- it allows to reduce the dependence between client and server applications ("weak coupling")
- it offers an interoperability system independent from the platform and technologies
- it supports the interoperability in the web, since it is based on SOAP protocol which usually uses HTTP as basic protocol (which all firewalls enable)
- it is based on XML descriptors (WSDL)

Examples of usage

The mechanism of application interoperability, provided by CMDBuild through the SOAP web service, can be used to activate the dialogue with every other information system pre-existing in the same organization and supporting that standard protocol.

Exhaustive examples of usage can include:

- activation in non-“JSR 168-compliant” intranet portals of simple interaction with CMDBuild for non-technicians (report print, workflow start or development, etc)
- synchronization with other CMDB tools
- integration with management applications which require to recover from CMDBuild the updated inventory of fixed assets or which must take to CMDBuild administration data of a fixed asset
- integration with monitoring technical tools which need to know information on assets subject to their control

Provided services

Method categories

The CMDBuild web service provides methods for the external execution of basic functions managed in the system, dedicated in particular to manage and print cards and to perform processes.

In particular the available methods categories are as follows:

- management of cards: creation, modify, delete, search, history
- management of lists used to bind information fields to groups of predefined values:
creation, modify, delete, search on lists and their values

• management of relations among cards: creation, modify, delete, search, history
• management of cards' attachments (filed in the repository of the document system Alfresco used by CMDBuild): upload, download, modify, delete
• management of processes configured in the system (and interpreted by the workflow engine Enhydra Shark integrated in CMDBuild): start, data recording, advancing at the following step

All methods provided in the web service can be used upon authentication in the CMDBuild system. The authentication is performed upon the WSS Username Token profile 1.0 specification\(^1\) with digest password.

You can find the detail description of single available methods in the next chapter.

**Support data structures**

Web service methods exposed by CMDBuild use specific support data structures corresponding to the following object typologies:

• cards (data cards)
• attribute (single custom attribute in the card)
• lookup (value in the predefined list used to optimize an information attribute)
• query (filter query to select the card list)
• filter (specific to an atomic filter condition in the query)
• filterOperator (concatenation of filter conditions)
• order (organization among cards drawn from the filter query)
• relation (correlation among cards)
• attachment (document enclosed in a card)

You can find the detail description of single employed objects in the next chapter.

**Authentication system**

In order to have all services offered by the CMDBuild web services, the access user must be authenticated in the system.

This is possible thanks to the authentication based on UsernameToken.

CMDBuild uses Axis2 to supply its services outside and the Apache Rampart module to manage the users authentication via web service, employing the WSS Username Token profile 1.0 standard with digest password.

This standard provides the dispatch of credentials, meant as password username and hash. It is calculated according to the algorithm defined in the specification, in the header of the SOAP message.

Once the message has been received, the Rampart module verifies if CMDBuild provides a user corresponding to the credentials.

When this occurs and if the user is allowed to access the required service, then the web service

\(^1\) The reference document can be found at [http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf](http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf)
will report the result, otherwise there will be an error message.

**Error management system**

The error management in the CMDBuild system provides the use of custom error codes expressly defined.

Using the web service, besides the errors of Axis (and the Rampart module about the authentication), also CMDBuild custom errors can be returned.

In the following table you will find the list of custom error codes in CMDBuild; potentially, they can be returned from the web service with their meaning.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTFOUND_ERROR</td>
<td>Element not found</td>
</tr>
<tr>
<td>AUTH_MULTIPLE_GROUPS</td>
<td>The user is connected with multiple groups</td>
</tr>
<tr>
<td>AUTH_UNKNOWN_GROUP</td>
<td>Unknown group</td>
</tr>
<tr>
<td>AUTH_NOTAUTHORIZED</td>
<td>The authorizations are not enough to perform the operation</td>
</tr>
<tr>
<td>ORM GENERIC_ERROR</td>
<td>An error has occurred while reading/saving data</td>
</tr>
<tr>
<td>ORM DUPLICATE_TABLE</td>
<td>There is already a class with this name</td>
</tr>
<tr>
<td>ORM CAST ERROR</td>
<td>Error in the type conversion</td>
</tr>
<tr>
<td>ORM UNIQUE VIOLATION</td>
<td>Not null constraint violated</td>
</tr>
<tr>
<td>ORM CONTAINS DATA</td>
<td>You can not delete classes or attributes of tables or domains containing data</td>
</tr>
<tr>
<td>ORM TYPE ERROR</td>
<td>Not corresponding type</td>
</tr>
<tr>
<td>ORM ERROR GETTING PK</td>
<td>The main key can not be determined</td>
</tr>
<tr>
<td>ORM ERROR LOOKUP CREATION</td>
<td>The lookup can not be created</td>
</tr>
<tr>
<td>ORM ERROR LOOKUP MODIFY</td>
<td>The lookup can not be modified</td>
</tr>
<tr>
<td>ORM ERROR LOOKUP DELETE</td>
<td>The lookup can not be deleted</td>
</tr>
<tr>
<td>ORM ERROR RELATION CREATE</td>
<td>The relation can not be deleted</td>
</tr>
<tr>
<td>ORM ERROR RELATION MODIFY</td>
<td>The relation can not be modified</td>
</tr>
<tr>
<td>ORM CHANGE LOOKUPTYPE ERROR</td>
<td>The lookup type can not be changed</td>
</tr>
<tr>
<td>ORM READ ONLY TABLE</td>
<td>Read-only table</td>
</tr>
<tr>
<td>ORM READ ONLY RELATION</td>
<td>Read-only relation</td>
</tr>
<tr>
<td>ORM DUPLICATE ATTRIBUTE</td>
<td>There is already an attribute with this name</td>
</tr>
<tr>
<td>ORM DOMAIN HAS REFERENCE</td>
<td>Domains with reference attributes can not be deleted</td>
</tr>
<tr>
<td>ORM FILTER CONFLICT</td>
<td>Conflict by defining the filter</td>
</tr>
<tr>
<td>ORM AMBIGUOUS DIRECTION</td>
<td>The direction relation can not be automatically determined</td>
</tr>
</tbody>
</table>
Description of support data structures

“Card” object

It represents a general typology of card configured in the system.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Type</th>
<th>Mandatory</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>className</td>
<td>string</td>
<td></td>
<td></td>
<td>Class name which includes the card. It corresponds to the table name in the database.</td>
</tr>
<tr>
<td>id</td>
<td>integer</td>
<td></td>
<td></td>
<td>Card identification, it is automatically assigned by the database.</td>
</tr>
<tr>
<td>attributeList</td>
<td>Attribute[]</td>
<td></td>
<td></td>
<td>Array of “Attribute” objects containing the values of additional custom attributes in the class. They correspond to additional attributes defined in the CMDBuild Administration Module and available in the card management. The list includes also the ClassId (not the className).</td>
</tr>
<tr>
<td>beginDate</td>
<td>date</td>
<td></td>
<td></td>
<td>It shows the creation date of the current card version (in response to the initial insertion or the last change. It is a date object in the TimeZone definition format of the standard XML Schema (YYYY-MM-DDThh:mm:ssZ).</td>
</tr>
<tr>
<td>user</td>
<td>string</td>
<td></td>
<td></td>
<td>Username of the person that performed the last operation on the card.</td>
</tr>
</tbody>
</table>

“Attribute” object

It represents a single "custom” attribute (which is additional if compared to the two default attributes “Code” and “Description”) belonging to a card typology configured in the system.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Type</th>
<th>Mandatory</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td></td>
<td>Attribute name, as created with the CMDBuild Administration Module. It corresponds to the table name in the database table.</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td></td>
<td></td>
<td>It corresponds to the attribute value.</td>
</tr>
<tr>
<td>code</td>
<td>string</td>
<td></td>
<td></td>
<td>It is developed only for &quot;Reference&quot; attributes with the id of the card.</td>
</tr>
</tbody>
</table>

“Lookup” object

It represents the value of a list with predefined values (Lookup list) used for the development of a Lookup attribute.
### Activity name

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Type</th>
<th>Mandatory</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>integer</td>
<td></td>
<td></td>
<td>Lookup identification, it is automatically assigned by the database</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td></td>
<td></td>
<td>Name of the Lookup list which includes the current heading.</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td></td>
<td></td>
<td>Description of the Lookup heading (one single heading of a Lookup list).</td>
</tr>
<tr>
<td>code</td>
<td>string</td>
<td></td>
<td></td>
<td>Code of the Lookup heading (one single heading of a Lookup list).</td>
</tr>
<tr>
<td>parent</td>
<td>Lookup</td>
<td></td>
<td></td>
<td>Lookup object corresponding to the parent heading of the current heading.</td>
</tr>
<tr>
<td>parentid</td>
<td>integer</td>
<td></td>
<td></td>
<td>Identification of the parent Lookup in the current heading (if applicable)</td>
</tr>
<tr>
<td>position</td>
<td>integer</td>
<td></td>
<td></td>
<td>Location of the Lookup heading in the related Lookup list</td>
</tr>
<tr>
<td>notes</td>
<td>string</td>
<td></td>
<td></td>
<td>Notes connected with the Lookup heading</td>
</tr>
</tbody>
</table>

### "Query" object

It represents a filter on the values to search. The filter can be unique (Filter) or a set of conditions with a common comparison operator (FilterOperator).

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Type</th>
<th>Mandatory</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filter</td>
<td>Filter</td>
<td></td>
<td></td>
<td>Atomic filter condition</td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>filterOperator</td>
<td>FilterOperator</td>
<td></td>
<td></td>
<td>Concatenation of filter conditions</td>
</tr>
</tbody>
</table>

### "Filter" object

It represents an atomic filter condition to select a card list.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Type</th>
<th>Mandatory</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
<td></td>
<td></td>
<td>Attribute which the filter condition is applied to.</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td></td>
<td></td>
<td>Value for the comparison with the attribute's content.</td>
</tr>
<tr>
<td>operator</td>
<td>string</td>
<td></td>
<td></td>
<td>Comparison operator (values such as EQUALS, LIKE are admitted).</td>
</tr>
</tbody>
</table>

### "FilterOperator" object

It represents a concatenation of atomic filter conditions connected with an operator.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Type</th>
<th>Mandatory</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operator</td>
<td>string</td>
<td></td>
<td></td>
<td>Concatenation operator to join filter conditions (values such as AND, OR are admitted)</td>
</tr>
</tbody>
</table>
Sample of operation to perform: creation of the filter “Supplier = Quasartek s.r.l.  OR Supplier = IBM Italia s.p.a.”

Filter1:
name: Supplier
domain: Quasartek s.r.l.
operator: EQUALS

Filter2:
name: Supplier
domain: IBM Italia s.p.a.
operator: EQUALS

FilterOperator:
subquery: [Filter1, Filter2]
operator: OR

Query:
filterOperator: FilterOperator

“Order” object
It represents the ordering standard among the cards drawn from the filter query.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Type</th>
<th>Mandatory</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>columnName</td>
<td>string</td>
<td></td>
<td></td>
<td>Attribute which the ordering is performed on.</td>
</tr>
<tr>
<td>type</td>
<td>string</td>
<td></td>
<td></td>
<td>Ordering typology applied (only ASC and DESC values are admitted).</td>
</tr>
</tbody>
</table>

“Relation” object
It represents a correlation between pairs of cards, filed in the system.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Type</th>
<th>Mandatory</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>domainName</td>
<td>string</td>
<td></td>
<td></td>
<td>Domain used for the relation.</td>
</tr>
<tr>
<td>class1Name</td>
<td>string</td>
<td></td>
<td></td>
<td>ClassName of the first card taking part in the relation.</td>
</tr>
<tr>
<td>card1Id</td>
<td>integer</td>
<td></td>
<td></td>
<td>Identifier of the first card which takes part in the relation.</td>
</tr>
<tr>
<td>class2Name</td>
<td>string</td>
<td></td>
<td></td>
<td>ClassName of the second card which takes part in the relation.</td>
</tr>
<tr>
<td>card2Id</td>
<td>integer</td>
<td></td>
<td></td>
<td>Identifier of the second card which takes part in the relation.</td>
</tr>
<tr>
<td>status</td>
<td>string</td>
<td>A</td>
<td></td>
<td>Relation status (‘A’ = active, ‘N’ = removed)</td>
</tr>
<tr>
<td>beginDate</td>
<td>date</td>
<td></td>
<td></td>
<td>Date when the relation was created (format</td>
</tr>
</tbody>
</table>
**“Attachment” object**

It represents a document enclosed in a card filed in the system.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Type</th>
<th>Mandatory</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>string</td>
<td></td>
<td></td>
<td>Category which the attachment belongs to (from proper Lookup list).</td>
</tr>
<tr>
<td>description</td>
<td>string</td>
<td></td>
<td></td>
<td>Description related to the attachment.</td>
</tr>
<tr>
<td>filename</td>
<td>string</td>
<td></td>
<td></td>
<td>Attachment name with extension</td>
</tr>
<tr>
<td>version</td>
<td>string</td>
<td></td>
<td></td>
<td>Document version in the DMS Alfresco</td>
</tr>
<tr>
<td>author</td>
<td>string</td>
<td></td>
<td></td>
<td>User that performs the file upload</td>
</tr>
<tr>
<td>created</td>
<td>date</td>
<td></td>
<td></td>
<td>Date when the document was inserted in the DMS</td>
</tr>
<tr>
<td>modified</td>
<td>date</td>
<td></td>
<td></td>
<td>Date of the last change to the document in the DMS</td>
</tr>
</tbody>
</table>

**Description of web service methods**

**Management area of cards**

Here’s the list of methods dedicated to the card management.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createCard</td>
<td>Card card</td>
<td>integer id</td>
<td>It creates in the database a new card, containing the information inserted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>in the &quot;Card&quot; object. It returns the &quot;id&quot; identification attribute.</td>
</tr>
<tr>
<td>deleteCard</td>
<td>string className integer cardId</td>
<td>boolean return</td>
<td>It deletes logically - in the identified class - the pre-existing card with</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>the identified &quot;id&quot;. It returns &quot;true&quot; if the operation went through.</td>
</tr>
<tr>
<td>updateCard</td>
<td>string className integer cardId Attribute[] attributeList</td>
<td>boolean return</td>
<td>It updates a pre-existing card. It returns &quot;true&quot; if the operation went</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>through.</td>
</tr>
<tr>
<td>getCard</td>
<td>string className integer cardId Attribute[] attributeList</td>
<td>Card card</td>
<td>It returns the required card with all attributes specified in &quot;attributeList&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(all card attributes if &quot;attributeList&quot; is null).</td>
</tr>
</tbody>
</table>
getCardList

string className
Attribute[] attributeList
Query queryType
Order[] orderType
Card[] cardList

It returns the card list resulting from the specified query, completed with all attributes specified in "attributeList" (all card attributes if “attributeList” is null). If the query is made on a superclass, the "className" attribute of the returned Card objects contains the name of the specific subclass the card belongs to, while in the attributeList it appears the ClassId of the same subclass.

getCardHistory

string className
integer cardId
Card[] cardList

It returns the list of the historicized versions of the specified card.

Management area of Lookup headings

Here's the list of methods dedicated to the management of Lookup headings (predefined values lists which a card attribute is held to).

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createLookup</td>
<td>Lookup lookup</td>
<td>integer id</td>
<td>It creates in the database a new heading of a data Lookup list containing information inserted in the “Lookup” object. It returns the &quot;id&quot; identification attribute.</td>
</tr>
<tr>
<td>deleteLookup</td>
<td>integer lookupId</td>
<td>boolean return</td>
<td>It deletes logically - in the identified class - the pre-existing card with the identified &quot;id&quot;. It returns “true” if the operation went through.</td>
</tr>
<tr>
<td>updateLookup</td>
<td>Lookup lookup</td>
<td>boolean return</td>
<td>It updates the pre-existing Lookup heading. It returns “true” if the operation went through.</td>
</tr>
<tr>
<td>getLookupById</td>
<td>integer id</td>
<td>Lookup lookup</td>
<td>It returns the Lookup heading which shows the specified &quot;Id&quot; identification.</td>
</tr>
</tbody>
</table>
| getLookupList    | string type
string value
boolean parentList | Lookup[] lookupList | It returns a complete list of Lookup values corresponding to the specified “type”. If the "value" parameter is specified, only the related heading is returned. If "parentList" takes the "True" value, it returns the complete hierarchy available for the multilevel Lookup lists. |

Management area of relations

Here's the list of methods dedicated to the relation management among cards.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createRelation</td>
<td>Relation relation</td>
<td>boolean return</td>
<td>It creates in the database a new relation between the pair of cards specified in the &quot;Relation&quot; object. It returns “true” if the operation went through.</td>
</tr>
<tr>
<td>Method</td>
<td>Input</td>
<td>Output</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------</td>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>deleteRelation</td>
<td>Relation relation</td>
<td>boolean return</td>
<td>It deletes the existing relation between the pair of cards specified in the &quot;Relation&quot; object. It returns &quot;true&quot; if the operation went through.</td>
</tr>
<tr>
<td>getRelationList</td>
<td>string domain, string className, integer cardId</td>
<td>Relation[] relationList</td>
<td>It returns the complete list of relations of the card specified for the specified domain.</td>
</tr>
<tr>
<td>getRelationHistory</td>
<td>Relation relation, Relation[] relationList</td>
<td></td>
<td>It returns the relation history of a card starting from a &quot;Relation&quot; object in which only &quot;Class1Name&quot; and &quot;Card1Id&quot; were defined.</td>
</tr>
</tbody>
</table>

**Management area of workflows**

Here's the list of methods dedicated to the management of the processes configurable in CMDBuild.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startWorkflow</td>
<td>Card card boolean CompleteTask</td>
<td>integer id</td>
<td>It starts a new instance of the workflow described in the specified &quot;Card&quot;. If the &quot;CompleteTask&quot; parameter takes the &quot;true&quot; value, the process is advanced to the following step. It returns the &quot;id&quot; identification attribute.</td>
</tr>
<tr>
<td>updateWorkflow</td>
<td>string processId Attribute[] attributeList boolean CompleteTask</td>
<td>boolean ret</td>
<td>It updates the information of the card in the specified process instance. If the &quot;CompleteTask&quot; parameter takes the &quot;true&quot; value, the process is advanced to the following step. It returns &quot;true&quot; if the operation went through.</td>
</tr>
</tbody>
</table>

**Management area of attachments**

Here's the list of methods dedicated to the management of documents enclosed in a card.

<table>
<thead>
<tr>
<th>Activity name</th>
<th>Input</th>
<th>Output</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uploadAttachment</td>
<td>string className integer cardId Base64Binary file string fileName string category string description</td>
<td>boolean return</td>
<td>It uploads the specified file in the DMS Alfresco and the relating connection to the CMDBuild card belonging to the &quot;className&quot; class and having the &quot;id&quot; identification. It returns &quot;true&quot; if the operation went through.</td>
</tr>
<tr>
<td>downloadAttachment</td>
<td>string className integer cardId string fileName</td>
<td>Base64Binary file</td>
<td>It returns the file enclosed in the specified card, which has the specified name.</td>
</tr>
<tr>
<td>deleteAttachment</td>
<td>string className integer cardId string fileName</td>
<td>boolean return</td>
<td>It removes from the DMS Alfresco the file enclosed in the specified card, which has the specified name. It returns &quot;true&quot; if the operation went through.</td>
</tr>
</tbody>
</table>
Web Service Manual

Web service and SOAP protocol

| updateAttachment | string className | integer cardId | string fileName | string description | boolean return | It updates the description of the file enclosed in the specified card, which has the specified name. It returns “true” if the operation went through. |

**Example of creating a “client”**

In this paragraph we propose two samples of creating SOAP "clients" enabled to interface with CMDBuild 1.0

**Java language**

**ATTENTION:** we advise you to use the wsdl2java tool. This tutorial implies the generation of the support classes with the above mentioned tool.

**Example of suggested operation:** obtain a list of all active cards of the "Computer" class

Once classes are generated, create the client as follows:

1. Create an instance in the ConfigurationContext class and indicate in it where the repository directory is. In the repository directory there are 2 directories: the modules directory which contains the rampart.mar file, and the conf directory which contains the file to define the safety policy which should be adopted.

   ```java
   ConfigurationContext configContext = ConfigurationContextFactory
   .createConfigurationContextFromFileSystem( //
       "/path/to/repository", //
       null //
   );
   ```

2. Instance the WebservicesStub class moving in it the ConfigurationContext just created

   ```java
   WebservicesStub stub = new WebservicesStub(configContext);
   ```

3. Set the authentication credentials

   ```java
   StAXOMBuilder builder = new StAXOMBuilder( //
       "/path/to/repository/conf/policy.xml" //
   );
   Options options = stub._getServiceClient().getOptions();
   options.setUserName("username");
   options.setPassword("password");
   options.setProperty( //
       RampartMessageData.KEY_RAMPART_POLICY, //
       PolicyEngine.getPolicy( builder.getDocumentElement() //
   
   //
   ```

4. Instance a GetCardList object and call the server

```java
GetCardList list = new GetCardList();
list.setClassName("Computer");
GetCardListResponse response = stub.getCardList(list);
Card[] card = response.get_return();
```

5. At this point you can iterate on the array card content and extract the most interesting values. For example, if you want to recover the description of every Computer, the following method will be enough:

```java
System.out.println(card[i].getDescription());
```

**PHP language**

ATTENTION: since PHP doesn't support in a native way the WS-Security standard and the MTOM specification to send SOAP messages with attachments, we advise you to use a framework which can add these functionalities. For this tutorial we used the framework WSO2/PHP1, released with Apache ver licence.

**Example of suggested operation:** upload in CMDBuild a file associated to the card with Id = 13 of the "Computer" class (filing in Alfresco system)

1. Request definition

```xml
$requestPayloadString = <<<XML
<?xml version="1.0" encoding="UTF-8" ?>
<ns1:upload xmlns:ns1="http://soap.services.cmdbuild.org">
    <className>Computer</className>
    <objectId>13</objectId>
    <file><xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include" href="cid:myid1"></xop:Include></file>
    <filename>$userfile_name</filename>
    <category>$category</category>
    <description>$description</description>
</ns1:upload>
XML;
```

2. Define a token security necessary for the authentication

```php
$security_options = array( "useUsernameToken" => TRUE );
$policy = new WSPolicy(array( "security" => $security_options ));
$security_token = new WSSecurityToken(array( "user" => "username", "password" => "password", "passwordType" => "Digest" ));
```

---

3 You can find the official website at the following address: http://wso2.org/projects/wsf/php
3. Instance a WSClient object to execute the request. You have to specify the MTOM use for the file transfer

```php
$client = new WSClient(array("useMTOM" => TRUE, "useWSA" => TRUE, "policy" => $policy,
"securityToken" => $security_token));
```

4. Store the file content in a single string using the function file_get_contents()

```php
$file = file_get_contents($_FILES["userfile"]["tmp_name"]);
```

5. Instance a WSMessage object and execute the request

```php
$url = "http://localhost:8080/cmdbuild/services/soap/FileTransfer";
$requestMessage = new WSMessage($requestPayloadString, array("to"=> $url, "attachments" =>
array("myid1" => $file));
$response = $client->request($requestMessage);
```

**Definition of authentication policy file**

**File structure**

The policy file - which must be passed to the Rampart module for client side - must be defined using the Web Services Security Policy Language (WS-SecurityPolicy).

This language, derived from XML, allows to define security rules respected by the application.

For example, in order to define the authentication use through Username Token with Password Digest, you have to define the policy as follows:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy wsu:Id="UTOverTransport"
 xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-ssecurity-utility-1.0.xsd"
  <wsp:ExactlyOne>
    <sp:SignedSupportingTokens
 xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitpolicy/200702">
      <wsp:Policy>
        <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
          <wsp:Policy>
            <sp:HashPassword/>
          </wsp:Policy>
        </sp:UsernameToken>
      </wsp:Policy>
    </sp:SignedSupportingTokens>
  </wsp:ExactlyOne>
</wsp:Policy>
```
You can also indicate in the policy file, certain configuration parameters of the Rampart module. For example, if you don't want to directly transfer the password in the code, but gather it from a database, you can suggest Rampart to use a custom class which implements the CallbackHandler function in which you can define how to find the password.

In that case you have to modify the policy file in the following way.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsp:Policy wsp:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-ssecurity-utility-1.0.xsd"
  <wsp:ExactlyOne>
    <sp:SignedSupportingTokens xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitpolicy/200702">
      <wsp:Policy>
        <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/AlwaysToRecipient">
          <wsp:Policy>
            <sp:HashPassword/>
            </wsp:Policy>
          </sp:UsernameToken>
        </wsp:Policy>
    </sp:SignedSupportingTokens>

    <ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
      <ramp:passwordCallbackClass>
        org.cmdbuild.services.soap.secure.MyPasswordHandler
      </ramp:passwordCallbackClass>
    </ramp:RampartConfig>
  </wsp:ExactlyOne>
</wsp:Policy>
```

For a deeper treatment of WS-SecurityPolicy, please refer to the related official document⁴.

REST web service

Services

Attention: all information in this section could be not updated or not detailed; it's a good habit to always refer to the WADL, looking at the desired endpoint (e.g. http://example.com/cmdbuild/services/rest/v1/_wadl).

If not differently specified, the content type of various HTTP methods will be “application/json”.

In order to prioritize resource description:

• JSON will be simplified omitting “ characters
• reported HTTP methods, references to the selected endpoint will be omitted

Version 1

Endpoint

http://hostname:port/cmdbuild/services/rest/v1

Dates format

All dates (time, timestamp and date attributes) uses the format “yyyy-MM-dd'T'HH:mm:ss”.

Session management

Access to the various resources is subject to the opening of a session that have associated some privileges. Each session is identified by an alphanumeric string or “token”.

Access to those resources requiring a session

After a session has been created or updated, its identifier must be used for all those resources that require it. The identifier should be sent within request's header.

GET .../classes/foo/cards/ HTTP/1.1
[various other headers]
CMDBuild-Authorization: 0123456789ABCDF...

If the resource requires a session and the token is not sent or is not correct then the answer will be:

HTTP/1.1 401 Unauthorized
[various headers]

Creation of a session

This is the only one method that does not require a session.
POST .../sessions/ HTTP/1.1
[various other headers]
{
    username : ...
    password : ...
}

HTTP/1.1 200 OK
[various headers]
{
    date : {
        _id : ${SESSION}
        username : ...
        password : null
        role : ...
    }
}

Reading a session

GET .../sessions/${SESSION}/ HTTP/1.1
[various other headers]

HTTP/1.1 200 OK
[various headers]
{
    username : ...
    password : null
    role : ...
}

Paying attention to the attribute role, if valued with "null" or absent then you will have some partial
authorizations, otherwise you will have full authorizations. If the user has only one role, then already after creating this will be present within the session. Alternatively you will need to update the credentials.

**Update of a session**

```
PUT .../sessions/$(SESSION)/ HTTP/1.1
[various other headers]
{
    role : ..., 
}
HTTP/1.1 204 No Content
[various headers]
```

You can change the role associated with a session (thus passing from a session with partial authorizations to complete and vice versa).

**Deletion of a session**

```
DELETE .../sessions/$(SESSION)/ HTTP/1.1
[various other headers]
HTTP/1.1 204 OK
[various headers]
```

**Reading of all roles associated to a session**

```
GET /sessions/$(SESSION)/roles/ HTTP/1.1
[various other headers]

HTTP/1.1 200 OK
[various headers]
{
    data : [foo, bar]
}
```
Lookup management

Reading of all types

GET .../lookup_types HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  },
  data : [
    {
      _id : ...,  
      name : ..., 
      parent : ...
    },
    ...
    {
      ...
    },
  ]
}

Read of the details of a type

GET .../lookup_types HTTP/1.1
[various headers]

HTTP/1.1 200 OK
Read of all values of a type

GET .../lookup_types/AlfrescoCategory/values HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    meta : {
        total : ...
    },
    data : [
        {
            _type" : AlfrescoCategory,
            _id" : 703,
            code" : null,
            description" : Document,
            parent_type" : null,
            active" : true,
            number" : 1,
            default" : false,
            parent_id" : null
        },
    ]
}
Web Service Manual

Read of the details of a value

GET .../lookup_types/AlfrescoCategory/values/703 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{

date : {
	_type" : AlfrescoCategory,
	_id" : 703,

code" : null,

description" : Document,

type" : null,

active" : true,

number" : 1,

default" : false,

parent_id" : null
}
}

Classes and cards management

Reading of all classes

GET .../classes HTTP/1.1
[various headers]
HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  }
  data : [
    {
      _id : Asset,
      name : Asset,
      description : Asset,
      parent : Class,
      prototype : true
    },
    ...
    {
      ...
    }
  ]
}

Read of the details of a class

GET .../classes/Asset HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  date : {

Read of all attributes of a class

GET .../classes/Asset/attributes HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta: {
    total: ...
  },
  data: [
    {
      lookupType: null,
      index: 1,
      group: General data,
      unique: false,
      scale: null,
      editorType: null,
      targetClass: null,
      active: true,
      description: Code,
      type: string,
    }
  ]
}
Creation of a card

POST .../classes/Building/cards HTTP/1.1

[varying headers]

{
Code = test,
...                  
}

HTTP/1.1 200 OK
[varying headers]

{
  data : 3217
}
## Reading of all cards

GET .../classes/Asset/cards HTTP/1.1

[various headers]

HTTP/1.1 200 OK

[various headers]

{
    meta : {
        total : ...
    },
    data : [
        {
            Room : null,
            Description : Acer - AL1716,
            Notes : null,
            Workplace : null,
            ScreenSize : null,
            Brand : 138,
            PurchaseDate : null,
            SerialNumber : null,
            AcceptanceNotes : null,
            Code : MON0001,
            TechnicalReference" : null,
            FinalCost : null,
            AcceptanceDate : null,
            Model : AL1716,
            _id : 550,
            _type : Asset,
            Assignee : 134,
            Type" : null,
        }
    ]
}
Reading of the details of a card

GET .../classes/Asset/cards/550 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]

{ 
  date: { 
    Room: null,
    Description: "Acer - AL1716",
    Notes: null,
    Workplace: null,
    ScreenSize: null,
    Brand: 138,
    PurchaseDate: null,
    SerialNumber: null,
    AcceptanceNotes: null,
    Code: "MON0001",
    TechnicalReference: null,
    FinalCost: null,
    AcceptanceDate: null,
    Model: "AL1716",
    _id: 550,
  }
}
Update of a card

PUT .../classes/Building/cards/3217 HTTP/1.1
[various headers]
{
  Code = test,
  ...
}

HTTP/1.1 204 No Content

Deletion of a card

DELETE .../classes/Building/cards/3217 HTTP/1.1
[various headers]

HTTP/1.1 204 No Content

Domains and relations management

Reading of all domains

GET .../domains HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{  
  meta : {  
    total : ...  
  }  
  data : [  
    {  
      _id : WorkplaceComposition,  
      name : WorkplaceComposition,  
      description : Workplace composition  
    },  
    ...  
    {  
      ...  
    }  
  ]  
}

Read of the details of a domain

GET ../domains/WorkplaceComposition HTTP/1.1
[various headers]  

HTTP/1.1 200 OK  
[various headers]  

{  
  data" : {  
    _id : WorkplaceComposition,  
    descriptionMasterDetail : Asset,  
    sourceProcess : false,  
    descriptionInverse : belongs to workplace,  
    descriptionDirect : includes assets,  
    destinationProcess : false,  
  }  
}
name : WorkplaceComposition,
source : Workplace,
destination : Asset,
cardinality : 1:N,
description : Workplace composition

Read of all attributes of a domain

GET .../domains/UserRole/attributes HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : 1
  },
  data : [
    {
      targetClass : null,
      editorType : null,
      unique : false,
      description : Default Group,
      index : 1,
      defaultValue : null,
      lookupType : null,
      active : true,
      precision : null,
      inherited : false,
      displayableInList : true,
    }
  ]
}
Creation of a relation

POST .../domains/UserRole/relations/ HTTP/1.1
[various headers]
{
  date : {
    _sourceType : User,
    _sourceId : 123,
    _destinationType : Role,
    _destinationId : 456,
    DefaultGroup : false
    }
}

HTTP/1.1 200 OK
[various headers]
{
  data : 789
}
Reading of all relations

GET .../UserRole/relations HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta: {
    total: ...
  },
  data: [
    {
      destinationDescription: 'Helpdesk',
      sourceDescription: 'Jones Patricia',
      sourceType: 'User',
      sourceld: 678,
      destinationType: 'Role',
      _type: 'UserRole',
      destinationId: 677,
      _id: 681
    },
    ...
    {
      ...
    }
  ]
}

Read of the details of a relation

GET .../domains/UserRole/relations/681 HTTP/1.1
Update of a relation

POST .../domains/UserRole/relations/123 HTTP/1.1
[various headers]
{
  date : {
    DefaultGroup : true
  }
}

HTTP/1.1 204 No Content
Deletion of a relation

```
DELETE .../domains/UserRole/relations/123 HTTP/1.1
[various headers]

HTTP/1.1 204 No Content
```

Processes, instances and activities management

Read of all processes

```
GET .../processes HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  }
  data : [
    {
      _id : RequestForChange,
      name : RequestForChange,
      prototype : false,
      parent : Activity,
      description : Request for change
    }
    ...
    {
    }
  ...
}
```
Read of the details of a process

GET .../processes/RequestForChange HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    date : {
        name : RequestForChange,
        statuses : [
            6,
            7,
            8,
            10
        ],
        description : Request for change,
        description_attribute_name : Description,
        prototype : false,
        _id : RequestForChange,
        parent : Activity,
        defaultStatus : 6
    }
}

Read of all attributes of a process

GET .../processes/RequestForChange/attributes HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
```json
{
    "meta": {
        "total": ...
    },
    "data": [
        {
            "name": "Dummy",
            "scale": null,
            "_id": "Dummy",
            "inherited": false,
            "precision": null,
            "description": "Dummy",
            "editorType": null,
            "type": "string",
            "unique": false,
            "targetClass": null,
            "defaultValue": null,
            "values": [],
            "filter": null,
            "group": null,
            "lookupType": null,
            "length": 100,
            "mandatory": false,
            "index": 0,
            "active": true,
            "displayableInList": false
        },
        ...
        {
            ...
        }
    ]
}
```
Read of all starting activities of a process

GET .../processes/RequestForChange/start_activities HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : 1
  }
  data : [
    {
      _id : RegisterRFC,
      description : Register RFC,
      writable : true
    }
  ]
}

Read of the details of a starting activity

GET .../processes/RequestForChange/start_activities/RegisterRFC HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  date : {
    attributes : [
"
Read of all the activities of an instance

GET .../processes/RequestForChange/instances/1460/activities HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    meta: {
        total: ...
    },
    data: [

Read of the details of an activity

GET .../processes/RequestForChange/instances/1460/activities/foo HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  date : {
    widgets : [
      {
        type : "OpenAttachment",
        active : true,
        _id : widget-57a9a9d9,
        required : false,
        date : {
          ... 
        },
        label : Attachments
      },
      ...
    ]
  }
}
Creation of a process instance

POST .../processes/RequestForChange/instances HTTP/1.1
[various headers]
{
    Requester : 123,
    RFCDescription : test,
    _advance = true
}
According with the value of the "_advance" attribute, an instance can be advanced or just saved.

**Reading of all process instances**

```
GET .../processes/RequestForChange/instances HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  }
  data : [
    {
      Category : null,
      FinalResult : null,
      RFCEndDate : null,
      ExecutionEndDate : null,
      RFCDescription : test,
      Requester : 123,
      ...
      _type : RequestForChange,
      _name : null,
      _id : 1460,
    },
    ...
  ]
}
```
Read of the details of an instance

GET .../processes/RequestForChange/instances/1460 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  date : {
    FinalResult : null,
    RFCEndDate : null,
    ExecutionEndDate : null,
    RFCDescription : test,
    Requester : 123,
    ...
    _type : RequestForChange,
    _name : null,
    _id : 1460,
    _status : 6
  }
}
Update of an instance

PUT .../processes/RequestForChange/instances/1460 HTTP/1.1
[various headers]
{
    Requester: 123,
    RFCDescription: test,
    _advance = true
}

HTTP/1.1 204 No Content

According with the value of the "_advance" attribute, an instance can be advanced or just saved.

Deletion of an instance

DELETE .../processes/RequestForChange/instances/1460 HTTP/1.1
[various headers]

HTTP/1.1 204 No Content

Menu management

Reading of the menu

GET .../menu HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    date: {
        menuType: root,
        children: [
            ...
        ]
    }
}
Configuration management

Reading of all attachments categories

GET .../configuration/attachments/categories HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta: {
    total: ...
  }
}
Read of all attributes of a category

> GET .../configuration/attachments/categories/Document/attributes HTTP/1.1

[various headers]

HTTP/1.1 200 OK
[various headers]

{  
  meta : {  
      total : ...
  },
  data : [
      
      {  
        length : null,
        index : 0,
        editorType : null,
        targetClass : null,
        inherited : null,
      }
  ]
}
Read of all process statuses

GET .../configuration/processes/statuses HTTP/1.1

[various headers]

HTTP/1.1 200 OK

[various headers]

{  
  meta : {  
    total : ...  
  },  
  data : [  
    {  
      _id : 6,  
      description : Open,  
      value : open  
    },  
    ...  
    {  
      ...  
    }  
  ]}

}
**Privileges management**

**Reading of all privileges for a role**

```
GET .../roles/Foo/classes_privileges HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    meta : {
        total : ...
    },
    data : [
        {
            name : UPS,
            description : UPS,
            mode : w,
            _id : UPS
        },
        ...
        {
            name : License,
            _id : License,
            description : License,
            mode" : r
        }
    ]
}
```

**Attachments management**

It is applied in the same way as reported for process instances. So, by replacing

```
.../classes/${CLASS_ID}/cards/${CARD_ID}
```

with
you will have the same capabilities.

Upload of an attachment

POST .../classes/Building/cards/64/attachments HTTP/1.1
[various headers]
Content-Type : multipart/form-data

== Part, name : "attachment", content-type : "application/json" ==
{
  _description : this is a test,
  _category" : Image,
}
====

== Part, name : "file", content-type : "*/" ==
  binary
====

HTTP/1.1 200 OK
[various headers]
{
  data : abc123
}

Notes:
- the method uses “multipart/form-data” content-type
- uploads file and sets properies

Reading of all attachments

GET .../classes/Building/cards/64/attachments HTTP/1.1
[various headers]
HTTP/1.1 200 OK
[various headers]
{
    meta : {
        total : ...
    }
    data : [
        {
            _name : test.jpg,
            _id" : "abc123,
            _version : 1.3,
            _description : this is a test,
            _modified" : 2014-12-16T12:35:12,
            _category" : Image,
            _created" : 2014-11-20T14:20:04,
            _author" : admin
        },
        ...
        {
            ...
        }
    ]
}

Reading of the details of an attachment
GET .../classes/Building/cards/64/attachments/abc123 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
```json
{
  date: {
    _category: Image,
    _created: 2014-11-20T14:20:04,
    _description: this is a test,
    _author: admin,
    _id: abc123,
    _version: 1.3,
    _modified: 2014-12-16T12:35:12,
    _name: test.jpg
  }
}
```

Download of an attachment

```
GET .../classes/Building/cards/64/attachments/abc123/download.jpg HTTP/1.1
[various headers]
```

```
HTTP/1.1 200 OK
[various headers]
Content-Type: application/octet-stream
```

Update of an attachment

```
PUT .../classes/Building/cards/64/attachments/abc123 HTTP/1.1
[various headers]
Content-Type: multipart/form-data
```

```
== Part, name: "attachment", content-type: "application/json" ==
{
  _description: this is a test,
  _category": Image,
}
```
== Part, name: "file", content-type: "*/*" ==
binary

HTTP/1.1 204 No Content

Notes:
- the method uses “multipart/form-data” content-type
- it uploads file and sets properties

Deletion of an attachment

DELETE .../classes/Building/cards/64/attachments/abc123 HTTP/1.1
[various headers]

HTTP/1.1 204 No Content

Version 2

Endpoint
http://hostname:port/cmdbuild/services/rest/v2

Dates format
All dates (time, timestamp and date attributes) uses the format "yyyy-MM-dd'T'HH:mm:ss".

Session management
Access to the various resources is subject to the opening of a session that have associated some privileges. Each session is identified by an alphanumeric string or "token".

Access to those resources requiring a session
After a session has been created or updated, its identifier must be used for all those resources that require it. The identifier should be sent within request’s header.

GET .../classes/foo/cards/ HTTP/1.1
If the resource requires a session and the token is not sent or is not correct then the answer will be:

HTTP/1.1 401 Unauthorized
[various headers]

**Creation of a session**

This is the only one method that does not require a session.

```plaintext
POST .../sessions/ HTTP/1.1
[various other headers]
{
    username : ..., password : ...
}
```

HTTP/1.1 200 OK
[various headers]
{
    date : {
        _id : ${SESSION}
        username : ...
        password : null
        role : ...
    }
}

**Reading a session**

```plaintext
GET .../sessions/${SESSION}/ HTTP/1.1
[various other headers]
```
HTTP/1.1 200 OK
[various headers]
{
  username : ...
  password : null
  role : ...
}

Paying attention to the attribute role, if valued with "null" or absent then you will have some partial authorizations, otherwise you will have full authorizations. If the user has only one role, then already after creating this will be present within the session. Alternatively you will need to update the credentials.

**Update of a session**

PUT .../sessions/$(SESSION)/ HTTP/1.1
[various other headers]
{
  role : ..., 
}

HTTP/1.1 204 No Content
[various headers]

You can change the role associated with a session (thus passing from a session with partial authorizations to complete and vice versa).

**Deletion of a session**

DELETE .../sessions/$(SESSION)/ HTTP/1.1
[various other headers]

HTTP/1.1 204 OK
[various headers]
Reading of all roles associated to a session

GET /sessions/${SESSION}/roles HTTP/1.1
[various other headers]

HTTP/1.1 200 OK
[various headers]
{
  data : [foo, bar]
}

Lookup management

Reading of all types

GET .../lookup_types HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  },
  data : [
    {
      _id : ..., 
      name : ..., 
      parent : ...
    },
    ...
  }
}
Read of the details of a type

GET .../lookup_types HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  date:
    _id: ..., 
    name: ..., 
    parent: ...
}

Read of all values of a type

GET .../lookup_types/AlfrescoCategory/values HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta:
    total: ...
  ,
  data: [
    

Read of the details of a value

GET .../lookup_types/AlfrescoCategory/values/703 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{

date: {
    _type": AlfrescoCategory,
    _id": 703,
    code": null,
    description": Document,
    parent_type": null,
    active": true,
    number": 1,
    default": false,
    parent_id": null
}
...
Classes and cards management

Reading of all classes

GET .../classes HTTP/1.1
[...]
HTTP/1.1 200 OK
[...]
{
  meta: {
    total: ...
  }
  data: [
    {
      _id: Asset,
      name: Asset,
      description: Asset,
      parent: Class,
      prototype: true
    },
    ...
    {
      ...
    }
  ]
}
Read of the details of a class

GET .../classes/Asset HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]

{
  date : {
    _id : Asset,
    name : Asset,
    description : Asset,
    description_attribute_name : Description,
    parent : Class,
    prototype : true
  }
}

Read of all attributes of a class

GET .../classes/Asset/attributes HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]

{
  meta : {
    total : ...
  },
  data : [
    {
```
lookupType: null,
index: 1,
group: General data,
unique: false,
scale: null,
editorType: null,
targetClass: null,
active: true,
description: Code,
type: string,
displayableInList: true,
values: [],
length: 100,
name: Code,
_id: Code,
filter: null,
mandatory: false,
precision: null,
defaultValue": null,
inherited": true
}
...
{
  ...
}
]

Creation of a card

POST .../classes/Building/cards HTTP/1.1
[various headers]
{

Code = test,
...
}

HTTP/1.1 200 OK
[various headers]
{
  data : 3217
}

Reading of all cards

GET .../classes/Asset/cards HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  },
  data : [
    {
      Room : null,
      Description : Acer - AL1716,
      Notes : null,
      Workplace : null,
      ScreenSize : null,
      Brand : 138,
      PurchaseDate : null,
      SerialNumber : null,
    }
  ]
AcceptanceNotes : null,
Code : MON0001,
TechnicalReference" : null,
FinalCost : null,
AcceptanceDate : null,
Model : AL1716,
_id : 550,
_type : Asset,
Assignee : 134,
Type" : null,
Supplier" : null

Reading of the details of a card

GET .../classes/Asset/cards/550 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{

    Date : {

        Room : null,
        Description : Acer - AL1716,
        Notes : null,
        Workplace : null,
        ScreenSize : null,
    }

}
Update of a card

```
PUT .../classes/Building/cards/3217 HTTP/1.1
[ various headers ]

{ 
  Code = test,
  ...
}
```

HTTP/1.1 204 No Content

Deletion of a card

```
DELETE .../classes/Building/cards/3217 HTTP/1.1
[ various headers ]
```
HTTP/1.1 204 No Content

Domains and relations management

Reading of all domains

GET .../domains HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]

{ 
  meta: { 
    total: ...
  } 
  data: [
    { 
      _id: WorkplaceComposition, 
      name: WorkplaceComposition, 
      description: Workplace composition 
    }, ...
    { 
      ...
    }
  ]
}

Read of the details of a domain

GET .../domains/WorkplaceComposition HTTP/1.1
[various headers]
HTTP/1.1 200 OK
[various headers]
{
    data" : {
        _id : WorkplaceComposition,
        descriptionMasterDetail : Asset,
        sourceProcess : false,
        descriptionInverse : belongs to workplace,
        descriptionDirect : includes assets,
        destinationProcess : false,
        name : WorkplaceComposition,
        source : Workplace,
        destination : Asset,
        cardinality : 1:N,
        description : Workplace composition
    }
}

Read of all attributes of a domain

GET .../domains/UserRole/attributes HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    meta : {
        total : 1
    },
    data : [
        
    ]
}
##### Creation of a relation

```
POST .../domains/UserRole/relations/ HTTP/1.1
[various headers]
{
  date : {
    _sourceType : User,
    _sourceId : 123,
    _destinationType : Role,
  }
}
```
Reading of all relations

GET .../UserRole/relations HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]

{  
data : 789
}

_{destinationId} : 456,
DefaultGroup : false

}
Read of the details of a relation

GET .../domains/UserRole/relations/681 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  date : {
    _id : 681,
    _destinationId : 677,
    _destinationType : Role,
    _destinationDescription : Helpdesk,
    _sourceId : 678,
    _sourceDescription : Jones Patricia,
    _type : UserRole,
    DefaultGroup : null,
    _sourceType : User
  }
}

Update of a relation

POST .../domains/UserRole/relations/123 HTTP/1.1
Deletion of a relation

DELETE ../domains/UserRole/relations/123 HTTP/1.1
[various headers]

HTTP/1.1 204 No Content

Processes, instances and activities management

Read of all processes

GET ../processes HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]

{ meta : { total : ... } data : [ { _id : RequestForChange, ... } ] }
Read of the details of a process

GET .../processes/RequestForChange HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  date: {
    name: RequestForChange,
    statuses: [
      6,
      7,
      8,
      10,
    ],
    description: Request for change,
    description_attribute_name: Description,
    prototype: false,
    _id: RequestForChange,
    parent: Activity,
  }
}
defaultStatus : 6
}

Read of all attributes of a process

GET .../processes/RequestForChange/attributes HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  },
  data : [
    {
      name : Dummy,
      scale : null,
      _id : Dummy,
      inherited : false,
      precision" : null,
      description : Dummy,
      editorType : null,
      type : string,
      unique : false,
      targetClass : null,
      defaultValue : null,
      values : [],
      filter : null,
      group : null,
      lookupType : null,
    }
  ]
}
Read of all starting activities of a process

GET .../processes/RequestForChange/start_activities HTTP/1.1

[various headers]

HTTP/1.1 200 OK
[various headers]

{  
  meta: {  
    total: 1  
  }  
  data: [  
    {  
      _id: RegisterRFC,  
      description: Register RFC,  
      writable: true  
    }  
  ]
}
Read of the details of a starting activity

GET .../processes/RequestForChange/start_activities/RegisterRFC HTTP/1.1
[variuous headers]

HTTP/1.1 200 OK
[variuous headers]
{
date : {
    attributes : [
    {
        mandatory : true,
        _id : Requester,
        index : 0,
        writable : true
    },
    {
        writable : true,
        index : 1,
        _id : RFCDescription,
        mandatory : true
    }
    ],
    instructions : ...,
    description : Register RFC,
    widgets : [],
    _id : RegisterRFC
}
Read of all the activities of an instance

GET .../processes/RequestForChange/instances/1460/activities HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  },
  data : [
    {
      description : Formal evaluation,
      _id : foo,
      writable : true
    },
    ...
    {
      ...
    }
  ]
}

Read of the details of an activity

GET .../processes/RequestForChange/instances/1460/activities/foo HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{

```json

date: {
    widgets: [
        {
            type: 'OpenAttachment',
            active: true,
            _id: 'widget-57a9a9d9',
            required: false,
            date: {
                ...,
            },
            label: 'Attachments',
        },
        ...
    ],
    _id: 'FormalEvaluation',
    attributes: [
        {
            mandatory: false,
            writable: false,
            _id: 'RFCStartDate',
            index: 0
        },
        ...
    ],
    instructions: '...',
    description: 'Formal evaluation'
}
```
Creation of a process instance

POST .../processes/RequestForChange/instances HTTP/1.1
[various headers]
{
  Requester : 123,
  RFCDescription : test,
  _advance = true
}

HTTP/1.1 200 OK
[various headers]
{
  data : 1460
}

According with the value of the "_advance" attribute, an instance can be advanced or just saved.

Reading of all process instances

GET .../processes/RequestForChange/instances HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  }
  data : [}
Read of the details of an instance

GET .../processes/RequestForChange/instances/1460 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]

{
    date: {
        FinalResult: null,
        RFCEndDate: null,
        ExecutionEndDate: null,
    }
}
RFCDescription : test,
Requester : 123,
...
_type : RequestForChange,
_name : null,
_id : 1460,
_status : 6
}

Update of an instance

PUT .../processes/RequestForChange/instances/1460 HTTP/1.1
[various headers]
{
    Requester : 123,
    RFCDescription : test,
    _advance = true
}

HTTP/1.1 204 No Content

According with the value of the "_advance" attribute, an instance can be advanced or just saved.

Deletion of an instance

DELETE .../processes/RequestForChange/instances/1460 HTTP/1.1
[various headers]

HTTP/1.1 204 No Content
Menu management

Reading of the menu

GET .../menu HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  date: {
    menuType: 'root',
    children: [
      {
        menuType: 'folder',
        objectId: 0,
        children: [
          ...
        ],
        ...
      },
      ...
    ],
    objectDescription: 'Employee',
    children: [],
    objectType: 'Employee',
    objectId: 0,
    index: 34,
    menuType: 'class'
  }
}
**Configuration management**

**Reading of all attachments categories**

```
GET .../configuration/attachments/categories HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    meta: {
        total: ...
    },
    data: [
        {
            _id: Document,
            description: "Document"
        },
        ...
    ]
}
```

**Read of all attributes of a category**

```
> GET .../configuration/attachments/categories/Document/attributes HTTP/1.1
[various headers]

HTTP/1.1 200 OK
```
Read of all process statuses

GET .../configuration/processes/statuses HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]

{  
  meta : {  
    total : ...  
  },  
  data : [  
    {  
      length : null,  
      index : 0,  
      editorType : null,  
      targetClass : null,  
      inherited : null,  
      ...  
    },  
    ...  
    {  
      ...  
    }  
  ]
}
Privileges management

Reading of all privileges for a role

GET .../roles/Foo/classes_privileges HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]

{  
    meta: {  
        total: ...
    },  
    data: [  
        {  
            name: UPS,  
            description: UPS,  
            mode: w,  
            _id: UPS  
        },  
        ...  
    ]
}
Attachments management

It is applied in the same way as reported for process instances. So, by replacing

```plaintext
.../classes/${CLASS_ID}/cards/${CARD_ID}
```

with

```plaintext
.../processes/${PROCESS_ID}/instances/${PROCESS_ID}
```

you will have the same capabilities.

Upload of an attachment

```plaintext
POST .../classes/Building/cards/64/attachments HTTP/1.1
[various headers]
Content-Type : multipart/form-data

== Part, name : "attachment", content-type : "application/json" ==
{
    _description : this is a test,
    _category" : Image,
}
====

== Part, name : "file", content-type : "*/" ==
binary
====
```
HTTP/1.1 200 OK
[

data : abc123
]

Notes:
- the method uses "multipart/form-data" content-type
- uploads file and sets properties

Reading of all attachments

GET .../classes/Building/cards/64/attachments HTTP/1.1
[

data : [
        {
            _name : test.jpg,
            _id" : "abc123,
            _version : 1.3,
            _description : this is a test,
            _modified" : 2014-12-16T12:35:12,
            _category" : Image,
            _created" : 2014-11-20T14:20:04,
            _author" : admin
        },
        ...
    ]
Reading of the details of an attachment

GET ...
/Classes/Building/cards/64/attachments/abc123
HTTP/1.1

[various headers]

HTTP/1.1 200 OK

[various headers]

{
  date : {
    _category : Image,
    _created : "2014-11-20T14:20:04",
    _description : this is a test,
    _author : admin,
    _id : abc123,
    _version : 1.3,
    _modified : "2014-12-16T12:35:12",
    _name : test.jpg
  }
}

Download of an attachment

GET ...
/Classes/Building/cards/64/attachments/abc123/download.jpg
HTTP/1.1

[various headers]

HTTP/1.1 200 OK
Update of an attachment

PUT .../classes/Building/cards/64/attachments/abc123 HTTP/1.1
[various headers]
Content-Type : multipart/form-data

== Part, name : "attachment", content-type : "application/json" ==
{
   _description : this is a test,
   _category" : Image,
}
====

== Part, name : "file", content-type : "/*/*" ==
binary
====

HTTP/1.1 204 No Content

Notes:
- the method uses “multipart/form-data” content-type
- it uploads file and sets properties

Deletion of an attachment

DELETE .../classes/Building/cards/64/attachments/abc123 HTTP/1.1
[various headers]

HTTP/1.1 204 No Content
CQL query

Query execution

GET .../cql?... HTTP/1.1
[various headers]

Parameters:

- **filter** includes the expression

```json
{
  CQL: ...
}
```

- **sort** includes the organized list of the attributes to execute the order

```json
[
  {
    property: Description
    direction: ASC
  },
  {
    property: Code
    direction: DESC
  }
]
```

- **limit** the number of returned results
- **start** the offset from the beginning of the list

HTTP/1.1 200 OK
[various headers]

```json
{
  meta: {
    total: ...
  }
  data: [
    {
      _id: Asset,
      name: Asset,
    }
  ]
}
```
E-mail template management

Reading of all e-mail templates

GET .../email_templates?start=...&limit=... HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    meta: {
        total: ...
    }
    data: [ foo, bar, baz ]
}

Reading of the details of a template

GET .../email_templates/foo HTTP/1.1
[various headers]

HTTP/1.1 200 OK
E-mail management

It is applied in the same way as reported for process instances. So, by replacing

```
.../classes/${CLASS_ID}/cards/${CARD_ID}/emails
```

with

```
.../processes/${PROCESS_ID}/instances/${PROCESS_ID}/emails
```

you will have the same capabilities.

Reading of e-mail statuses

```
GET .../classes/Building/cards/64.getEmails/statuses HTTP/1.1
[various headers]
```
HTTP/1.1 200 OK
[various headers]
{
  meta : {
      total : ...
  }
  data : [ received, draft, outgoing, sent ]
}

In order to send an e-mail, it should be set as "outgoing". At that point you cannot change it anymore.

**Creation of an e-mail**

```
POST .../classes/Building/cards/64/emails HTTP/1.1
[various headers]
{
  from: ...
  to: ...
  cc: ...
  bcc: ...
  subject: ...
  body: ...
  status: ...
  notifyWith: ...
  noSubjectPrefix: ...
  account: ...
  template: ...
  keepSynchronization: ...
  promptSynchronization: ...
  delay: ...
}
```
HTTP/1.1 200 OK
{date: 123}

**Reading of all e-mails**

GET .../classes/Building/cards/64/emails HTTP/1.1
{various headers}

HTTP/1.1 200 OK
{various headers}
{meta: {
total: ...
},
data: [
{
from: ...
to: ...
cc: ...
bcc: ...
subject: ...
body: ...
status: ...
notifyWith: ...
noSubjectPrefix: ...
account: ...
template: ...
keepSynchronization: ...
}
promptSynchronization: ...
    delay: ...
},
{
    ...
}
}

Reading of the details of an e-mail

GET .../classes/Building/cards/64/emails/123 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    date : {
        from: ...
        to: ...
        cc: ...
        bcc: ...
        subject: ...
        body: ...
        status: ...
        notifyWith: ...
        noSubjectPrefix: ...
        account: ...
        template: ...
        keepSynchronization: ...
        promptSynchronization: ...
        delay: ...
    }
}
E-mail update

PUT .../classes/Building/cards/64/emails/123 HTTP/1.1
[various headers]
{
    from: ...
    to: ...
    cc: ...
    bcc: ...
    subject: ...
    body: ...
    status: ...
    notifyWith: ...
    noSubjectPrefix: ...
    account: ...
    template: ...
    keepSynchronization: ...
    promptSynchronization: ...
    delay: ...
}

HTTP/1.1 204 No Content

Deletion of an e-mail

DELETE .../classes/Building/cards/64/emails/123 HTTP/1.1
[various headers]

HTTP/1.1 204 No Content
Management of functions

Reading of all functions

GET .../functions?start=...&limit=... HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  },
  data : [
    {
      name: ...
      description: ...
    },
    {
      ...
    }
  ]
}

Reading of the details of a function

GET .../functions/foo HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  date : {
}}
Reading of input values/attributes of a function

GET .../functions/foo/parameters?start=...&limit=... HTTP/1.1

[various headers]

HTTP/1.1 200 OK
[various headers]

{
    meta:
        total: ...,
    data:
        {
            type: ...
            name: ...
            description: ...
            ...,
        },
        {
            ...
        }
}

Reading of output values/attributes of a function

GET .../functions/foo(attributes?start=...&limit=... HTTP/1.1

[various headers]
HTTP/1.1 200 OK
[various headers]
{
    meta : {
        total : ...
    },
    data : [
        {
            type: ...
            name: ...
            description: ...
            ...
        },
        {
        ...
        }
    ]
}

Execution of a function in order to get its results

GET .../functions/foo/outputs HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    meta : {
        total : ...
    },

Report management

Reading of all reports

GET .../reports?start=...&limit=... HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    meta : {
        total : ...
    },
    data : [
        {
            _id: ...
            description: ...
        },
        {
            ...
        }
    ]
}
Reading of the details of a report

GET .../reports/123 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  date : {
    _id: ...
    title: ...
    description: ...
  }
}

Reading of input values/attributes of a report

GET .../reports/123/attributes?start=...&limit=... HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  } ,
  data : [
    {
      type: ...
      name: ...
      description: ...
    }
  ]
}
Download of a report

GET .../reports/123/somename.someextension?extension=...&parameters={foo: FOO, bar: BAR, ...}
HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
Content-Type : application/octet-stream

Management of domain trees

Reading of all trees

GET .../domainTrees?start=...&limit=... HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    meta : {
        total : ...
    },
    data : [
    {
        _id: ...
    }}
Reading of the details of a tree

GET .../domainTrees/123 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  date: {
    _id: ...
    description: ...
    nodes: [
      {
        _id: ...
        parent: ...
        metadata: {
          ...
        }
      },
      ...
    ]
  }
}
Management of images

Reading of all folders

GET .../filestores/images/folders HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
    total : ...
  },
  data : [
    {
      _id: ...
      name: ...
      parent: ...
    },
    {
      ...
    }
  ]
}

Reading of the details of a folder

GET .../filestores/images/folders/foo HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{

Upload of a file into a folder

POST .../filestores/images/folders/foo/files HTTP/1.1
[various headers]
Content-Type : multipart/form-data

== Part, name : "file", content-type : "/*/*" ==
binary
=====

HTTP/1.1 200 OK
[various headers]
{
  data : bar
}

Reading of all files into a folder

GET .../filestores/images/folders/foo/files HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
  meta : {
  
}
total: ...
}
,

data: [
{
    _id: ...
    name: ...
    parent: ...
},
{
    ...
},
]
]

Reading of the details of a file

GET .../filestores/images/folders/foo/files/bar HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    date: {
        _id: ...
        name: ...
        parent: ...
    }
}

Download of a file

GET .../filestores/images/folders/foo/files/bar/download HTTP/1.1
[various headers]
HTTP/1.1 200 OK
[various headers]
Content-Type: application/octet-stream

Deletion of a file
DELETE .../filestores/images/folders/foo/files/bar HTTP/1.1
[various headers]

HTTP/1.1 204 No Content

Icons management

JSON model of the icon:

```json
{
  _id: ...
  type: ...
  details: {
    ...
  }
  image: {
    type: ...
    details: {
      ...
    }
  }
}
```

type: it represents the kind of icon to upload. The supported types are
- class
- process
details: according to type
- **type:** *class*
  id has to include the class name

- **type:** *process*
  id has to include the process name

**image:** it represents the associated image

- **type:** "filestore"

  - **details:** according to the type, but "filestore" is the only one that is supported, so:
    - **store:** store id (see Methods for the file management)
    - **folder:** folder id (see Methods for the file management)
    - **file:** file id (see Methods for the file management)

**Creation of an icon**

POST .../icons HTTP/1.1
[various headers]
{
  type: ...
  details: {
    ...
  }
  image: {
    type: ...
    details: {
      ...
    }
  }
}
HTTP/1.1 200 OK
[various headers]
{
  date: 123
}
**Reading of all icons**

GET .../icons HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    meta : {
        total : ...
    },
    data : [
        {
            _id: ...
            ...
        },
        {
            ...
        }
    ]
}

**Reading of the details of an icon**

GET .../icons/123 HTTP/1.1
[various headers]

HTTP/1.1 200 OK
[various headers]
{
    date : {
        ...
    }
}
Update of an icon

```
PUT ../../../icons/123 HTTP/1.1
[various headers]
{
  ...
}

HTTP/1.1 204 No Content
```

Deletion of a file

```
DELETE ../../../icons/123 HTTP/1.1
[various headers]

HTTP/1.1 204 No Content
```
APPENDIX: Glossary

ATTACHMENT
An attachment is a file associated to a card.
Attachments containing text (PDF, Open Office, Microsoft Word, etc.) are indexed in full text mode so that they can appear in search results.

WORKFLOW STEP
"Activity" means one of the steps of which the process consists.
An activity has a name, an executor, a type, possible attributes and methods with statements (CMDBuild API) to be executed.
A process instance is a single process that has been activated automatically by the application or manually by an operator.
See also: Process

ATTRIBUTE
The term refers to an attribute of a CMDBuild class.
CMDBuild allows you to create new attributes (in classes and domains) or edit existing ones.
For example, in "supplier" class the attributes are: name, address, phone number, etc.
Each attribute corresponds, in the Management Module, to a form field and to a column in the database.
See also: Class, Domain, Report, Superclass, Attribute Type

BIM
Method with the aim to support the whole life cycle of a building: from its construction, use and maintenance, to its demolition, if any.
The BIM method (Building Information Modeling) is supported by several IT programs that can interact through an open format for data exchange, called IFC (Industry Foundation Classes).
See also: GIS

CI
We define CI (Configuration Item) each item that provides IT service to the user and has a sufficient detail level for its technical management.
CI examples include: server, workstation, software, operating system, printer, etc.
See also: Configuration

CLASS
A Class is a complex data type having a set of attributes that describe that kind of data.
A Class models an object that has to be managed in the CMDB, such as a computer, a software, a service provider, etc.
CMDBuild allows the administrator - with the Administration Module - to define new classes or delete / edit existing ones.

Classes are represented by cards and, in the database, by tables automatically created at the definition time.

See also: Card, Attribute

**CONFIGURATION**

The configuration management process is designed to keep updated and available to other processes the items (CI) information, their relations and their history.

It is one of the major ITIL processes managed by the application.

See also: CI, ITIL

**DASHBOARD**

In CMDBuild, a dashboard corresponds to a collection of different charts, in this way you can immediately hold in evidence some key parameters (KPI) related to a particular management aspect of the IT service.

See also: Report

**DATABASE**

The term refers to a structured collection of information, hosted on a server, as well as utility software that handle this information for tasks such as initialization, allocation, optimization, backup, etc.

CMDBuild relies on PostgreSQL, the most powerful, reliable, professional and open source database, and uses its advanced features and object-oriented structure.

**DOMAIN**

A domain is a relation between two classes.

A domain has a name, two descriptions (direct and inverse), classes codes, cardinality and attributes.

The system administrator, using the Administration Module, is able to define new domains or delete / edit existing ones.

It is possible to define custom attributes for each domain.

See also: Class, Relation

**DATA FILTER**

A data filter is a restriction of the list of those elements contained in a class, obtained by specifying boolean conditions (equal, not equal, contains, begins with, etc.) on those possible values that can be accepted by every class attribute.

Data filters can be defined and used exceptionally, otherwise they can be stored by the operator and then recalled (by the same operator or by operators of other user groups, which get the permission to use them by the system Administrator).

See also: Class, View

**GIS**

A GIS is a system able to produce, manage and analyse spatial data by associating geographic
elements to one or more alphanumeric descriptions.

GIS functionalities in CMDBuild allow you to create geometric attributes (in addition to standard attributes) that represent, on plans / maps, markers position (assets), polylines (cable lines) and polygons (floors, rooms, etc.).

See also: BIM

GUI FRAMEWORK

It is a user interface you can completely customise. It is advised to supply a simplified access to the application. It can be issued onto any webportals and can be used with CMDBuild through the standard REST webservice.

See also: Mobile, Webservice

ITIL

"Best practices" system that established a "standard de facto"; it is a nonproprietary system for the management of IT services, following a process-oriented schema (Information Technology Infrastructure Library).

ITIL processes include: Service Support, Incident Management, Problem Management, Change Management, Configuration Management and Release Management.

For each process, ITIL handles description, basic components, criteria and tools for quality management, roles and responsibilities of the resources involved, integration points with other processes (to avoid duplications and inefficiencies).

See also: Configuration

LOOKUP

The term "Lookup" refers to a pair of values (Code, Description) set by the administrator in the Administration Module.

These values are used to bind the user's choice (at the form filling time) to one of the preset values.

With the Administration Module it is possible to define new "LookUp" tables according to organization needs.

MOBILE

It is a user interface for mobile tools (smartphones and tablets). It is implemented as multi-platform app (iOS, Android) and can be used with the CMDB through the REST webservice.

See also: GUI Framework, Webservice

PROCESS

The term "process" (or workflow) refers to a sequence of steps that realize an action.

Each process will take place on specific assets and will be performed by specific users.

A process is activated by starting a new process (filling related form) and ends when the last workflow step is executed.

See also: Workflow step

RELATION

A relation is a link between two CMDBuild cards or, in other words, an instance of a given domain.
A relation is defined by a pair of unique card identifiers, a domain and attributes (if any).

CMDBuild allows users, through the Management Module, to define new relations among the cards stored in the database.

See also: Class, Domain

REPORT
The term refers to a document (PDF or CSV) containing information extracted from one or more classes and related domains.

CMDBuild users run reports by using the Management Module; reports definitions are stored in the database.

See also: Class, Domain, Database

CARD
The term "card" refers to an element stored in a class.

A card is defined by a set of values, i.e. the attributes defined for its class.

CMDBuild users, through the Management Module, are able to store new cards and update/delete existing ones.

Card information is stored in the database and, more exactly, in the table/columns created for that class (Administration Module).

See also: Class, Attribute

SUPERCLASS
A superclass is an abstract class used to define attributes shared between classes. From the abstract class you can derive real classes that contain data and include both shared attributes (specified in the superclass) and specific subclass attributes.

For example, you can define the superclass "Computer" with some basic attributes (RAM, HD, etc.) and then define derived subclasses "Desktop", "Notebook", "Server", each one with some specific attributes.

See also: Class, Attribute

ATTRIBUTE TYPE
Each attribute has a data type that represents attribute information and management.

The attribute type is defined using the Administration Module and can be modified within some limitations, depending on the data already stored in the system.

CMDBuild manages the following attribute types: "Boolean", "Date", "Decimal", "Double", "Inet" (IP address), "Integer", "Lookup" (lists set in "Settings" / "LookUp"), "Reference" (foreign key), "String", "Text", "Timestamp".

See also: Attribute

VIEW
A view not only includes the whole content of a CMDB class, it is a group of cards defined in a logical way.

In particular, a view can be defined in CMDBuild by applying a filter to a class (so it will contain a reduced set of the same rows) or specifying an SQL function which extracts attributes from one or
more related classes.
The first view type maintains all functionalities available for a class, the second one allows the sole display and search with fast filter.
See also: Class, Filter

**WEBSERVICE**
A webservice is an interface that describes a collection of methods, available over a network and working using XML messages.
With webservices, an application allows other applications to interact with its methods.
CMDBuild includes a SOAP and a REST webservice.

**WIDGET**
A widget is a component of a GUI that improves user interaction with the application.
CMDBuild uses widgets (presented as "buttons") that can be placed on cards or processes. The buttons open popup windows that allow you to insert additional information, and then display the output of the selected function.