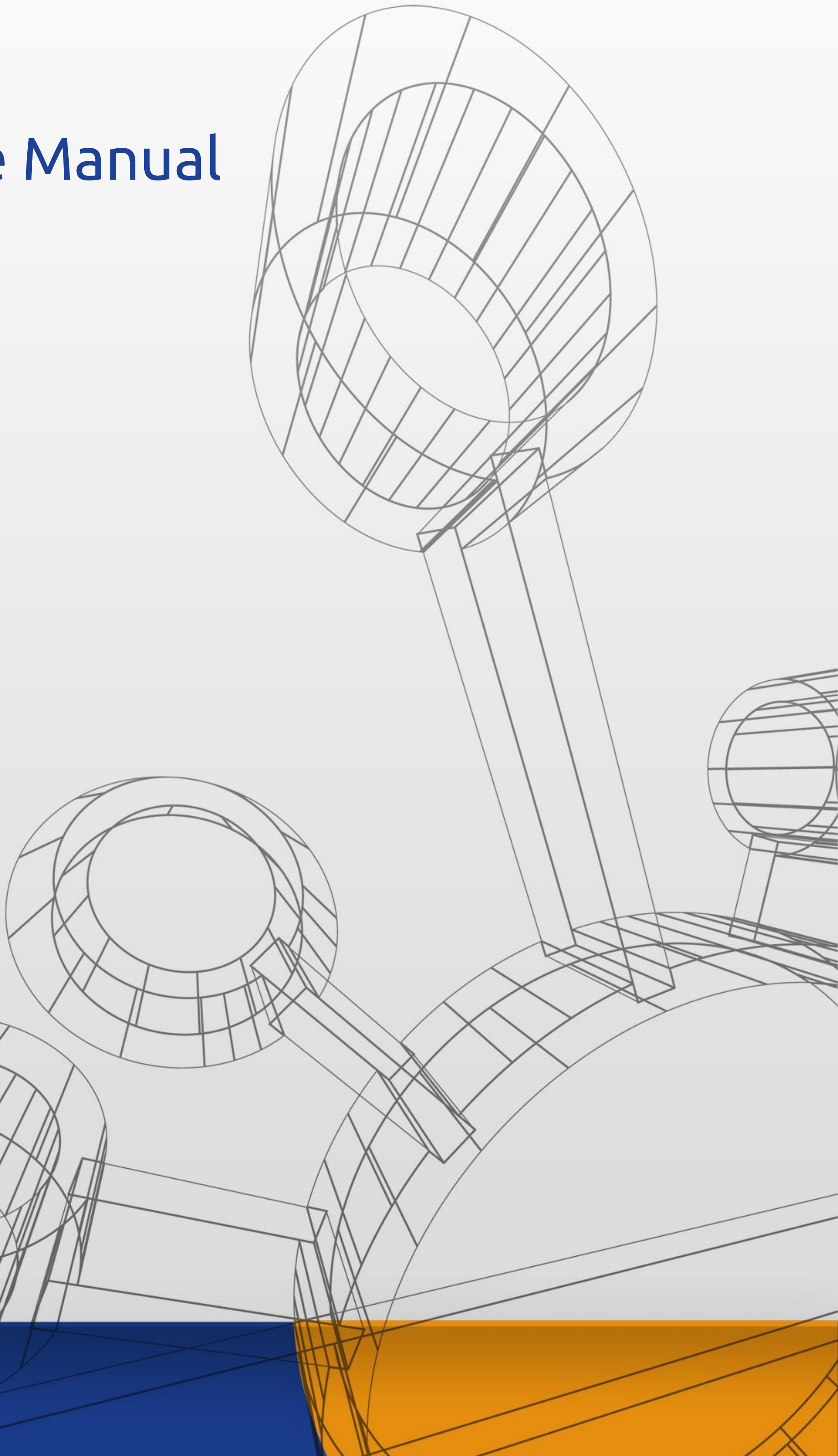
Version
3.4

» Webservice Manual

January 2022
Author Tecnoteca srl
www.tecnoteca.com

ENG



www.cmdbuild.org

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

Contents

1. Introduction

1.1. The application

CMDBuild is an open source web environment for the configuration of custom applications for the Asset Management.

On the one hand, it provides native mechanisms for the administrator, implemented in a "core" code which has been kept separated from the business logic, so that the system can be configured with all its features.

On the other hand, it generates dynamically a web interface for the operators, so that they can keep the asset situation under control and always know their composition, detachment, functional relations and how they update, in order to manage their life-cycle in a comprehensive way.

The system administrator can build and extend his/her own CMDB (hence the name of the project), modeling the CMDB according to the company needs; a proper interface 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.

Using external visual editors, the administrator can design workflows, import them into CMDBuild and put them at operators' disposal, so that they can execute them according to the configured automatisms.

In a similar way, using external visual editors, the administrator can design various reports on CMDB data (printouts, graphs, barcode labels, etc.), import them into the system and put them at operators' disposal.

The administrator can also configure some dashboards made up of charts which immediately show the situation of some indicators in the current system (KPI).

A task manager included in the user interface of the Administration Module allows you to schedule various operations (process starts, e-mail receiving and sending, connector executions) and to control CMDB data (synchronous and asynchronous events). Based on their findings, it sends notifications, starts workflows and executes scripts.

Thanks to document management systems that support the CMIS standard (Content Management Interoperability Services) - among which there is also the open source solution Alfresco - you will be able to attach documents, pictures, videos and other files.

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

The system also includes a REST webservice, so that CMDBuild users can implement custom interoperability solutions with external systems.

Furthermore, CMDBuild includes two external frameworks:

- the Advanced Connector CMDBuild, which is written in Java and can be configured in Groovy: it helps the implementation of connectors with external data sources, i.e automatic inventory systems, virtualization or monitoring ones (supplied with non-open source licence to the users that subscribe the annual Subscription with Tecnoteca)
- the GUI Framework CMDBuild, which helps the implementation of additional graphical interfaces, i.e. web pages (simplified for non technicians) that have to be published on external portals and that are able to interact with the CMDB through the REST webservice

CMDBuild includes a mobile interface (for smartphone and tablet). It is implemented as multi-platform app (iOS, Android) and is able to interact with the CMDB through the REST webservice (supplied with non-open source licence to the users that subscribe the annual Subscription with Tecnoteca).

CMDBuild is an enterprise system: server-side Java, web Ajax GUI, SOA architecture (Service Oriented Architecture), based on webservice and implemented by using the best open source technologies and following the sector standards.

CMDBuild is an ever-evolving system, which has been released for the first time in 2006 and updated several times a year in order to offer more features and to support new technologies.

1.2. Official website

CMDBuild has a dedicated website: http://www.cmdbuild.org

The website gathers a lot of documents on technical and functional features of the project: brochures, slides, manuals (see next paragraph), testimonials, case histories, newsletters, forums.

1.3. CMDBuild modules

The CMDBuild application includes two main modules:

- the Administration Module for the initial definition and the next changes of the data model and the base configuration (relation classes and typologies, users and authorization, dashboards, upload report and workflows, options and parameters)
- 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.

1.4. Available manuals

This manual is dedicated to the Administration Module, through which the administrator can configure data, define users and permissions, and perform other tasks.

You can find all the manuals on the official website (http://www.cmdbuild.org):

- system overview ("Overview Manual")
- system administration (“Administrator Manual”)
- installation and system management ("Technical Manual")
- workflow configuration (“Workflow Manual”)
- webservice details and configuration (“Webservice Manual”)
- connectors to sync data through external systems (“ConnectorsManual”)

1.5. Applications based on CMDBuild

Tecnoteca has used the CMDBuild environment in order to implement two different pre-configured solutions:

- CMDBuild READY2USE, for the management of assets and IT services, oriented to internal IT infrastructures or services for external clients (http://www.cmdbuild.org/it/prodotti/ready2use) according to the ITIL best practice (Information Technology Infrastructure Library)
- openMAINT, for the inventory management of assets, properties and related maintenance

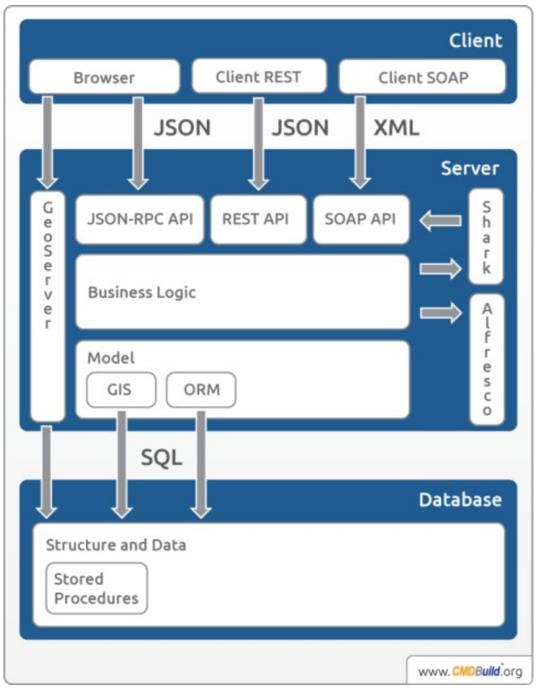
activities (http://www.openmaint.org)

Both applications are released with open source license, except for certain external components (data sync connectors, Self-Service portal, mobile APP, etc.), that are reserved to the users that subscribe the annual Subscription with Tecnoteca.

2. Interoperability standards

2.1. 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. REST protocol
2. SOAP 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.

3. Web services

3.1. Web Service introduction

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

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

Nowadays the two most used standards are:

- SOAP Web Services
- REST Web Services

In the following chapters both standards will be introduced, with a list of their differences and some examples.

3.2. SOAP Web Service introduction

SOAP (Simlpe Object Access Protocol) is a protocol based on XML language. Thanks to the XML usage SOAP, unlike other frameworks, provides a platform and language indipendent communication.

The structure of SOAP messages is divided in four parts:

- Envelope: a mandatory element that defines the beginning and the end of the message;
- Header: an optional element that contains amy optional attributes;
- Body: a mandatory element that provides the message that has to be sent;
- Fault: a mandatory element that can provide any error that occurs while processing the message;

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap="http://soap.services.cmdbuild.org">
   <soapenv:Header>
        ...
   <soapenv:Header/>
   <soapenv:Body>
        <soapenv:Fault>
        ...
        </soapenv:Fault>
   ...
   </soapenv:Body>
</soapenv:Envelope>
```

In the usage of SOAP for CMDBuild the header will be used mainly to authenticate the user, by adding a security field where the user can provide his username and password to access the service.

In the body field the user can provide the function that has to be called with the following syntax:

```
<soap:functionName/>
```

As an example of usage in CMDBuild, the following SOAP request will generate a session for the user specified in the header username field:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap="http://soap.services.cmdbuild.org">
   <soapenv:Header>
        <wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd">
        <wsse:UsernameToken>
        <wsse:Username>username</wsse:Username>
        <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-
200401-wss-username-token-profile-1.0#PasswordText">password</wsse:Password>
 </wsse:UsernameToken></wsse:Security>
   <soapenv:Header/>
   <soapenv:Body>
      <soap:createSession/>
   </soapenv:Body>
</soapenv:Envelope>
```

When executing a SOAP request, the target endpoint, in the CMDBuild case, has to be the Private.wsdl file in the CMDBuild installation. The WSDL file provides the user with a list of all available functions that can be called.

For a more detailed description on what you can do with SOAP on CMDBuild read chapter 4.

3.3. REST Web Service introduction

REST (REpresentational State Transfer), unlike SOAP, is an architectural style that provides a stateless, simple and lightweight way of communicating with a system.

The format used to send and receive data with REST web services is JSON. This format is a simple text containing a series of key-value pairs, like the following:

```
{
 “Key1”:”value1”,
 “Key2”:”value2”,
 …
}
```

When REST is used, requests can be sent to various endpoints through GET, PUT, POST and DELETE HTTP requests.

Elements such as authentication tokens can be added in the header of the request, data that has to be sent through PUT or POST requests can be added as parameters of the request.

As an example, if we want to generate a session like we previously did with the SOAP web service, we would firstly need to get the session endpoint of CMDBuild:

http://hostname:port/cmdbuild/services/rest/v3/sessions

And than perform a POST request with the username and password of the user to authenticate.

The response would than contain a success key followed by a data key containing the response values (from version 3.2 due to security reasons an additional request parameter has to be set to true in order to obtain the sessionId in the response, more at chapter 5.3.1):

```
{
 “success”:”true”,
 “data”:[”sessionId”:”generatedSessionId”,
        “username”:”user”,
        “password”:”userPassword”]
}
```

For a more detailed description on what you can do with REST on CMDBuild read chapter 5.

4. SOAP Web Services

4.1. CMDBuild WSDL

To obtain a list of every possible function that can be called through SOAP web services CMDBuild provides a WSDL called Private.wsdl. It can be opened with a normal text editor to visualize an XML containing the definition of every element, but it can also be opened by a software like SoapUI to have a more clear view of its content. To access this file once CMDBuild is up and running you can use the following URL: http://host:port/cmdbuild/services/soap/Private?wsdl

In the next paragraph a list of every available function will be provided.

4.2. SOAP Functions

In the following table a list of available SOAP functions, divided by cathegory, will be provided, note that after future updates the functions might change, so it’s always better to verify the function format in the WSDL file.

4.2.1. Cards

Card data structure:

- className: a string that identifies the owner class
- id: a bigint to identify the card
- attributeList: an array of attributes of the card
- beginDate: a date that shows the creation date of the card
- user: a string that shows what user last modified the card

Function	Parameters	Description
getCard	-className -cardId -attributeList	Function used to obtain the information relative to a specific card owned by a specific class
getCardHistory	-className -cardId -limit -offset	Function used to obtain the history of a specific card owned by a specific class. With limit and offset the amount of results can be modified
getCardList	-className -attributeList -queryType -orderType -limit -offset -fullTextQuery -cqlQuery	Function used to obtain a full list of cards owned by a specific class, with the possibility of specifing a filter (with Query type, FullTextQuery or CqlQuery), with the possibility of controlling the number and order of results via Limit, Offset and Order list
getCardListExt	-className -attributeList -queryType -orderType -limit -offset	Function used to obtain a full extended list of cards owned by a specific class, with the possibility of specifing a filter (with Query type, FullTextQuery or CqlQuery), with the possibility of controlling the number and order of results via Limit, Offset and Order list

	-fullTextQuery -cqlQuery	
getCardListWithLongDateFormat	-className -attributeList -queryType -orderType -limit -offset -fullTextQuery -cqlQuery	Function used to obtain a full list of cards owned by a specific class with a long date format, with the possibility of specifing a filter (with Query type, FullTextQuery or CqlQuery), with the possibility of controlling the number and order of results via Limit, Offset and Order list
getCardMenuSchema		
createCard	-className -attributeList -beginDate -endDate -metadata	Create a card owned by the specified class with a list of attributes specified in the request
updateCard	-className -attributeList -beginDate -endDate -id -metadata	Update a card owned by the specified class with the values specified in the request
deleteCard	-className -cardId	Delete a specific card owned by a specific class

4.2.2. Sessions

Function	Parameters	Description
createSession		Create a session for the user with username and password specified in the header

4.2.3. Lookups

Lookup data structure:

- id: a bigint to identify the lookup
- type: a string to identify the name of the lookup list which includes the current heading
- description; a string to describe the lookup heading
- code
- parent: the parent of the current lookup
- parentId: the id of the parent of the current lookup
- position: the position of the lookup in the lookup list
- notes: a string containing the optional notes of the lookup

Function	Parameters	Description
getLookupById	-id	Get a specific lookup by specifing its Id
getLookList		Get the full list of lookups
getLookListByCode	-type -code	Get the list of lookups of a specific type with a specific code
createLookup	-code	Create a new lookup with the values defined in the

	-description -notes -parent -type	parameters. The paret parameters requires a parentId and a position
updateLookup	-code -description -id -notes -parent -type	Update a lookup with the id specified in the parameters with the values
deleteLookup	-id	Delete the lookup with the specified Id

4.2.4. Attributes

Attribute data structure:

- name: a string that defines the attribute name
- value: a string to identify the attribute value
- code

Function	Parameters	Description
getAttributeList	-className	Get a list of attributes of the class specified with the class name parameter

4.2.5. Relations

Relation data structure:

- domainName: a string that defines the domain used for the relation
- class1Name: a string to identify the first class of the relation
- card1Id: a bigint to identify the first card of the relation
- class2Name: a string to identify the second class of the relation
- card2Id: a bigint to identify the second card of the relation

Function	Parameters	Description
getRelationAttributes	-class1Name -class2Name -card1Id -card2Id -domainName	Get the attributes of a specific relation
getRelationHistory	-class1Name -class2Name -card1Id -card2Id -domainName	Get the history of a specific relation
getRelationList	-className -cardId	Get the full list of relations of a specific card owned by the class specified by Class name
getRelationListExt	-domain -className -cardId	Get the full extended list of relations of a specific card owned by the class specified by Class name
createRelation	-class1Name -class2Name	Create a relation between the two classes and two cards specified in the parameters

	-card1Id -card2Id -domainName	
createRelationWithAttributes	-class1Name -class2Name -card1Id -card2Id -domainName -attributes	Create a relation between the two classes and two cards specified in the parameters with the provided list f attributes
updateRelationAttributes	-class1Name -class2Name -card1Id -card2Id -domainName -attributes	Update the attributes owned by the relation between the specified cards and classes provided in the parameters
deleteRelation	-class1Name -class2Name -card1Id -card2Id -domainName	Delete the specified relation

4.2.6. Classes

Function	Parameters	Description
getClassSchema	-className	Get the schema of a the class with the name specified in the parameters
getClassSchemaById	-classId -includeAttributes	Get the schema of a the class with the id specified in the parameters and, optionally, include its attributes
getClassSchemaByName	-flassName	Get the schema of a the class with the name specified in the parameters

4.2.7. Functions

Function	Parameters	Description
callFunction	-functionName -code -name -value	Execute the specified function
getFunctionList		Obtain a list of all available functions

4.2.8. Attachments

Attachment data structure:

- category: a string that identifies the category of the attribute
- description: a string that reppresent the description of the attachment
- filename: a string that contains the name of the file with the extension
- version: a string containing the version of the attachment
- author: a string containing the author of the upload
- created: a date indicating when the file was firstly uploaed

- modified: a date indicating when the file was last modifies

Function	Parameters	Description
copyAttachment	-sourceClassName -sourceId -destinationClassName -destinationId	Copy an attachment from one class to another class
updateAttachmentDescription	-className -filename -description	Modify the description of the attachment with name Filename owned by the specified class
downloadAttachment	-className -objectId -filename	Download the specified attachment

4.2.9. Reports

Function	Parameters	Description
getBuiltInReport	-id -extension -params	Get a report with the specified id and extension
getReport	-id -extension -params	Get a report with the specified id and extension
getReportList	-type -limit -offset	Get a full list of reports of the specified type
getReportParameters	-id -extension	Obtain a list of all report parameters for the specified report with the specified extension

4.2.10. Other functions

Function	Parameters	Description
abortWorkflow	-card	Abort a specific workflow for the specified card in the parameters
generateDigest	-plainText -digestAlgorithm	Generate a digest with the specified algorithm
getActivityMenuSchema		Get the activity menu schema
getActivityObjects	-className -cardId	Get a list of activity objects for a specific card owned by the class with className
getMenuSchema		Get the menu schema
getProcessHelp	-className -cardId	Get the process help for the specified card owned by the class with name className
getReference	-className -query -orderType -limit -offset -fullTextQuery -cqlQuery -	Get the specified reference
getUserInfo		Get the information about the authenticated user

notify		
resumeWorkflow	-card	Resume the workflow of a card
suspendWorkflow	-card	Suspend the workflow of a card
updateWorkflow	-card	Update the workflow of a card
sync	-xml	

5. REST Web Services

5.1. CMDBuild REST web service

Unlike SOAP, for the REST webservices there isn’t a file containing every function that can be requested on the same endpoint, but there are specific endpoints for every action.

If we want to operate on a card the endpoint will be: http://hostname:port/cmdbuild/services/rest/v3/cards

And with GET, POST, PUT, DELETE requests and addition to the endpoint path we can get a list of cards, update a specific card, create new ones, etc.

5.2. CMDBuild REST Endpoints

In this paragraph a list of currently available endpoints and data structure will be presented.

Note that in some cases the enpoint can have some variations in the path, for example some endpoints are the same for cards or process instances, and when that occours the “|” symbol is used. (classes|processes means we can either use classes or processes)

The “Function” column indicated the name of the endpoint function in the source code of CMDBuild.

When an endpoint requires additional information in the path it will be specified in the Path column.

When an endpoint requires additional query params, those will be specified in the Parameters column.

A data structure is presented whenever a POST/PUT endpoint requires a custom data structure containing the required details in a json format;

The majority of endpoints will provide a function to read the entire list of objects, a function to get the details of a specific object, a function to create a new object, a function to update an existing object and a function to delete an existing object.

In the path of various endpoints, whenever there is “id”, “classId”, “cardId” or similar, that has to be substituted with the id (or code when talking about classes) of the required element.

To avoid listing every time all the standard query parameters for endpoints that handle a list of items, a new object called StandardQueryParams has been created to always include those parameters. This is just to simplify the documentation, when the parameter StandardQueryParams is specified for an endpoint it is possible to use the following query parameters (this change doesn’t impact the usage, it’s just to avoid listing all the query parameters every time they are used):

• attrs	List of attributes to return
• filter	A string containing a filter
• sort	A string containing the sorting function
• limit	A long value to limit the amount of results
• offset	The pagination offset
• start	A long value to set an offset in the resultset
• detailed	A boolean to forse the server to return a detailed response
• positionOf	A long value containing an id to return in the meta field

5.2.1. Async Operation

Base url: http://hostname:port/cmdbuild/services/rest/v3/async

Path	Parameters	Type	Description
/jobs/{jobId}		GET	Obtain the status of an async job
/jobs/{jobId}/response		GET	Obtain the results of an async job

* {jobid}: Numeric id of the job

5.2.2. Audits

Base url: http://hostname:port/cmdbuild/services/rest/v3/system/audit

Path	Parameters	Type	Description
/mark		GET	Obtain the current date
/requests	-since String -limit Long	GET	Obtain a list of last requests since the date specified
/errors	-since String -limit Long	GET	Obtain a list of last errors since the date specified
/requests/{requestId}		GET	Get the details of a specific request

* {requestId}: String id of the request

5.2.3. Bim project

Base url: http://hostname:port/cmdbuild/services/rest/v3/bim/projects

Bim project data structure:

• name	String
• description	String
• importMapping	String
• projectId	String
• parentId	Long
• ownerClass	String
• ownerCard	String

Path	Parameters	Type	Description
		GET	Obtain the full list of Bim projects
/{projectId}/values/{globalId}	-if_exists boolean	GET	Obtain details about a bim value with the specified globalId
/{projectId}		GET	Obtain the specified Bim project
	-data json -file multipart dataHandler	POST	Create a new Bim project
/{projectId}	-data	PUT	Update the specified Bim project

	json -file multipart dataHandler		
/{projectId}/file	-ifcFormat String	GET	Download the specified Ifc file for the specified Bim project
/{projectId}/file	-file dataHandler -ifcFormat String	POST	Upload a new Ifc file for the specified Bim project
/{projectId}		DELETE	Delete the specified bim project
/{projectId}/convert/xkt		POST	Update the specified bim project from ifc to xkt

* {projectId}: Long id of the bim project
* {globalId}: String id of the bim value

5.2.4. Bim values

Base url: http://hostname:port/cmdbuild/services/rest/v3/bim/values

Path	Parameters	Type	Description
/globalId	-if_exists Boolean	GET	Obtain the Bim values of the specified Bim project

* {globalId}: String id of the bim value

5.2.5. Boot

Base url: http://hostname:port/cmdbuild/services/rest/v3/boot

Path	Parameters	Type	Description
/status		GET	Get the current status of the system
/database/check	-dbConfig Map<String,String>	POST	Verify the validity of the provided database configuration
/database/configure	-file multipart dataHandler -dbConfig Map<String,String>	POST	Configure the database with the provided configuration and provided database dump
/patches		GET	Obtain a list of currently available patches
/patches/apply		POST	Tell the system to perform the installation of every currently available patch

5.2.6. Calendar Event Attachments

Base url: http://hostname:port/cmdbuild/services/rest/v3/calendar/events/{eventId}/attachments

Attachment data structure:

• category	String
• fileName	String
• majorVersion	boolean
• cardValues	Map<String, Objects>

Path	Parameters	Type	Description
	-attachment attachmentData -file DataHandler -copyFrom_class String -copyFrom_card Long -copyFrom_id String	POST	Create a new attachment for the specified calendar event with the possibility of copying the attachment from another card
	-wsQueryOption	GET	Obtain a list of all the attachments for the specified event
/{attachmentId}		GET	Obtain a single attachment with the specified attachmentId
/{attachmentId}/{file}		GET	Download the file of a specific attachment
/{attachmentId}/preview		GET	Get the attachment preview if available
/{attachmentId}	-attachment attachmentData -file DataHandler	PUT	Update a specific attachment with the provided data
/{attachmentId}		DELETE	Delete an attachment with the specified id
/{attachmentId}/history		GET	Obtain the history of a specific attachment
/{attachmentId}/history/ version/file		GET	Get an older version of a specific attachment

* {eventId}: Long id of the event
* {attachmentId}: String id of the attachment

5.2.7. Calendar Event Email

Base url: http://hostname:port/cmdbuild/services/rest/v3/calendar/events/{eventId}/emails

Path	Parameters	Type	Description
	-applyTemplate Boolean -template_only Boolean -attachments List<String> -parts List<Attachment>	POST	Create a new email for the specified event with the possibility of adding a list of attachmentIds
/{emailId}	-emailData wsEmailData	PUT	Update an existing email
	-StandardQueryParams	GET	Obtain a list of every email associated with the specified event
/{emailId}		GET	Get the details of a specific email

			related to a specific event
/{emailId}		DELETE	Delete a sepcific email

* {eventId}: Long id of the event
***** {emailId}: Long id of the email

5.2.8. Calendar Event

Base url: http://hostname:port/cmdbuild/services/rest/v3/calendar/events

Event data structure:

• category	String
• priority	String
• card	Long
• sequence	Long
• content	String
• description	String
• timeZone	String
• eventEditMode	String
• partecipants	List<String>
• onCardDeleteAction	String
• type	String
• begin	String
• end	String
• completion	String
• owner	String
• status	String
• source	String
• notes	String
• values	Map<String, Object>

Path	Parameters	Type	Description
	-StandardQueryParams	GET	Obtain a list of every event
/{eventId}	-includeStats Boolean	GET	Get the details of a specific event
	-data json	POST	Create a new event with the provided data
/{eventId}	-data json	PUT	Update an existing event with the provided data
/{eventId}		DELETE	Remove an existing event
/{eventId}/history	-limit Long -start Long -detailed	GET	Obtain the history of a specific event

	Boolean		
/{eventId}/history/recordId		GET	Obtain the details of a specific record from the history of an event
/print/{file}	-StandardQueryParams -extension String -attributes String	GET	Obtain a report of all events (it is possible to filter the events with the standard query parameters)

* {eventId}: Long id of the event

5.2.9. Calendar Sequence

Base url: http://hostname:port/cmdbuild/services/rest/v3/calendar/sequences

Sequence data structure:

• category	String
• priority	String
• card	Long
• content	String
• description	String
• timeZone	String
• title	String
• eventCount	Integer
• frequencyMultiplier	Integer
• maxActiveEvents	Integer
• eventEditMode	String
• eventTime	String
• frequency	String
• partecipants	List<String>
• onCardDeleteAction	String
• sequenceParamsEditMode	String
• showGeneratedEventsPreview	Boolean
• eventType	String
• firstEvent	String
• lastEvent	String
• trigger	Long
• endType	String
• events	List<WsEventData>
• values	Map<String, Object>

Path	Parameters	Type	Description
/{sequenceId}	-includeEvents Boolean	GET	Obtain the details of a specific sequence, with the possibility of including the related events

/by-card/cardId	-detailed Boolean -includeEvents Boolean	GET	Obtain a list of sequences related to a specific card
	-data json	POST	Create a new sequence with the provided data
/{sequenceId}	-data json	PUT	Update an existing sequence with the provided data
/{sequenceId}		DELETE	Remove a specific sequence
/_ANY/generate-events	-data json	POST	Generate events based on the provided data

* {sequenceId}: Long id of the squence

5.2.10. Calendar Trigger

Base url: http://hostname:port/cmdbuild/services/rest/v3/calendar/triggers

Trigger data structure:

• category	String
• priority	String
• conditionScript	String
• content	String
• code	String
• description	String
• timeZone	String
• eventCount	Integer
• frequencyMultiplier	Integer
• maxActiveEvents	Integer
• delay	String
• eventEditMode	String
• eventTime	String
• frequency	String
• partecipants	List<String>
• onCardDeleteAction	String
• sequenceParamsEditMode	String
• showGeneratedEventsPreview	Boolean
• active	Boolean
• eventType	String
• ownerClass	String
• ownerAttr	String
• endType	String
• lastEvent	String
• scope	String

Path	Parameters	Type	Description
/{triggerId}		GET	Obtain the details of a specific trigger
/{triggerId}/generate-sqeuence	-dateValue String	GET	Generate a sequence based on the provided data
/{triggerId}/create-events		POST	Force the event creation for the specified trigger
	-StandardQueryParams	GET	Get a full list of calendar triggers
	-data json	POST	Create a new trigger with the provided data
/{triggerId}	-data json	PUT	Update an existing trigger with the provided data
/{triggerId}		DELETE	Delete an existing trigger

* {triggerId}: Long id of the squence

5.2.11. Calendar View Event

Base url: http://hostname:port/cmdbuild/services/rest/v3/calendar/views/viewId/events

Path	Parameters	Type	Description
/{eventId}		GET	Get the details of a specific event of a view
	-limit Long -start Long -detailed Boolean	GET	Obtain a list of events of a specific view

* {eventId}: Long id of the squence

5.2.12. Card Attachments

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/cards|instances/cardId/attachments

Attachment data structure:

• category	String
• fileName	String
• majorVersion	boolean
• cardValues	Map<String, Objects>

Path	Parameters	Type	Description
	-attachment attachmentData -file DataHandler -copyFrom_class String -copyFrom_card Long -copyFrom_id	POST	Create a new attachment for the specified card with the possibility of copying the attachment from another card

	String		
	-wsQueryOption	GET	Obtain a list of all the attachments for the specified card
/{attachmentId}		GET	Obtain a single attachment with the specified attachmentId
/{attachmentId}/download		GET	Download the file of a specific attachment
/_MANY/file	-attachmentId List<String>	GET	Obtain many attachments file based on the id list
/{attachmentId}/preview		GET	Get the attachment preview if available
/{attachmentId}	-attachment attachmentData -file DataHandler	PUT	Update a specific attachment with the provided data
/{attachmentId}		DELETE	Delete an attachment with the specified id
/{attachmentId}/history		GET	Obtain the history of a specific attachment
/{attachmentId}/history/ version/file		GET	Get an older version of a specific attachment

* {attachmentId}: String id of the attachment

5.2.13. Card Bim values

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/cards|instances/cardId/bimvalue

Path	Parameters	Type	Description
	-if_exists Boolean -include_related Boolean	GET	Obtain a list of every Bim value associated with the specified card

5.2.14. Card email attachments

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/cards|instances/cardId/emails/emailId/attachments

Attachment data structure:

• category	String
• fileName	String
• majorVersion	boolean
• cardValues	Map<String, Objects>

Path	Parameters	Type	Description
		POST	Create a new attachment for the specified email

		GET	Obtain a list of every available attachment for the specified email
/{attachmentId}		GET	Obtain the details of an attachment with id attachmentId of a specific email
/{attachmentId}/file		GET	Download the attachment with id attachmentId of the specified email
/{attachmentId}/preview		GET	Obtain a preview of the attachment with id attachmentId of the specified email
/{attachmentId}	-attachment multipart json -file multipart dataHandler	PUT	Update an existing attachment with id attachmentId of the specified email with the given data
/{attachmentId}		DELETE	Delete a specific attachment with if attachmentId of the email with id emailId
/{attachmentId}/history		GET	Obtain the history of a specific attachment
/{attachmentId}/history/ version/file:		GET	Download the specified previous version of the attachment with id attachmentId of the email with if emailId

* {attachmentId}: String id of the attachment

5.2.15. Card email

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/cards|instances/cardId/emails

Email data structure:

• delay	Long
• from	String
• replyTo	String
• to	String
• cc	String
• bcc	String
• subject	String
• body	String
• contentType	String
• account	Long
• template	Long
• autoReplyTemplate	Long
• keepSynchronization	boolean
• noSubjectPrefix	boolean

• promptSynchronization	boolean
• status	String
• _expr	String

Path	Parameters	Type	Description
	-limit Integer -start Integer -detailed Boolean	GET	Obtain a list of all available email associated with a card, with the possibility of changing the response to detailed or not and the possibility of change the number of results with limit
/{emailId}		GET	Obtain the details of a specific email
	-data List<json> -apply_template Boolean -template_only Boolean	POST	Create an email associated with a card with the data provided in emailData, with the possibility of extending a pre-existing template or using only a template
/{emailId}	-data json -apply_template Boolean -template_Only Boolean	PUT	Update an existing email associated with a card with the provided data in emailData
/{emailId}		DELETE	Delete a specific email

* {emailId}: Long id of the email

5.2.16. Card geo values

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/cards|instances/cardId/geovalues

Geo values data structure:

• _type	String
• x	Double
• y	Double
• points	List<WsPoint>

Point data structure:

• x	Double
• y	Double

Path	Parameters	Type	Description
		GET	Obtain every geovalue associated with a card
/{attributeId}		GET	Obtain the details of a specific geovalue
/{attributeId}	-data json	PUT	Update an existing geovalue with the provided data

/{attributeId}		DELETE	Delete a specific geovalue

* {attributeId}: Long id of the geo attribute

5.2.17. Card history

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/cards|instances/cardId/history

Path	Parameters	Type	Description
	-limit Long -start Long -types List<String>	GET	Get the history of a specific card, include cards, relations, system cards with the usage of the types parameter
/{recordId}		GET	Get the details of a specific card in the card history

* {recordId}: Long id of the history card

5.2.18. Card locks

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/cards|instances/cardId/lock

Path	Parameters	Type	Description
		GET	Get the lock of a specific card
		POST	Create a lock for a specific card
		DELETE	Release a lock for a specific card

5.2.19. Card print

Base url: http://hostname:port/cmdbuild/services/rest/v3/classes/classId/cards/cardId/print/file:

Path	Parameters	Type	Description
	-extension String	GET	Dwnload a file with an arbitrary extension, with the details of a specific card with cardId, owned by a class with classId

5.2.20. Card relations

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/cards|instances/cardId/relations

Relation data structure:

• _id	Long
• _type	String
• _sourceType	String
• _sourceId	Long
• _destinationType	String
• _destinationId	Long
• _is_direct	boolean

Path	Parameters	Type	Description
	-limit Long -start Long -detailed Boolean	GET	Get a list of relations for a specific card
	-relationData json	POST	Create a new relation for a specific card with the provided data
/{relationId}	-relationData json	PUT	Update an existing relation for a specific card with the provided data
/{relationId}		DELETE	Delete a specific relation

* {relationId}: Long id of the history card

5.2.21. Cards

Base url: http://hostname:port/cmdbuild/services/rest/v3/classes/classId/cards

Card data structure:

The data structure for a card is a map of string and objects that vary based on the class definition

Path	Parameters	Type	Description
/{cardId}	-includeModel Boolean -includeWidgets Boolean -includeStats Boolean	GET	Obtain the details of a specific card
	-StandardQueryParams -functionValue String -distinctIncludeNull Boolean -distinct String -count String	GET	Get a list of all available cards owned by a class, with the possibility of adding filters to limit the results
	-data json	POST	Create a new card with the provided data for a specific class
/{cardId}	-data json	PUT	Update an existing card with the provided data
/{cardId}		DELETE	Delete a specific card
	-StandardQueryParams -data json	PUT	Update many cards at once
	-StandardQueryParams	DELETE	Delete many cards at once

* {cardId}: Long id of the card

5.2.22. Charset

Base url: http://hostname:port/cmdbuild/services/rest/v3/syste,/charsets

Path	Parameters	Type	Description
		GET	Obtain a list of all available charsets

5.2.23. Chat messages

Base url: http://hostname:port/cmdbuild/services/rest/v3/session/current/messages

Message data structure:

• status	String
• target	String
• subject	String
• content	String
• thread	String
• meta	Map<String, String>

Path	Parameters	Type	Description
	-StandardQueryParams	GET	Obtain a list of all available messages for the current session
/{recordId}	-messageData json	PUT	Update the specified message
	-messageData json	POST	Create a new message with the specified data
/{recordId}		DELETE	Delete a message with the specified message id

* {recordId}: Long id of the message

5.2.24. Chat peers

Base url: http://hostname:port/cmdbuild/services/rest/v3/session/current/peers

Path	Parameters	Type	Description
	-StandardQueryParams	GET	Obtain a list of all available peers for the current session

5.2.25. Class attributes

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/attributes

Attribute data structure:

• formatPattern	String
• unitOfMeasure	String
• unitOfMeasureLocation	String
• visibleDecimals	Integer
• preselectIfUnique	boolean
• showThousandsSeparator	boolean
• showSeconds	boolean

• showSeparators	boolean
• type	String
• name	String
• description	String
• showInGrid	boolean
• showInReducedGrid	boolean
• domainKey	String
• domain	String
• direction	String
• unique	boolean
• mandatory	boolean
• active	boolean
• index	Integer
• defaultValue	String
• group	String
• precision	String
• scale	String
• targetClass	String
• maxLenght	Integer
• editorType	String
• lookupType	String
• filter	String
• help	String
• showIf	String
• validationRules	String
• mode	boolean
• autoValue	String
• metadata	map<String, String>
• classOrder	Integer
• isMasterDetail	Boolean
• masterDetailDescription	String
• ipType	String
• textContentSecurity	String

Path	Parameters	Type	Description
/{attributeId}		GET	Obtain the details of a specific attribute
	-limit Long -start	GET	Obtain a list of every available attribute for the specified class

	Long		
	-data json	POST	Create a new attribute for a specific class with the provided data
/{attributeId}	-data json	PUT	Update an existing attribute with the provided data
/{attributeId}		DELETE	Delete a specific attribute
/order	-attrOrder List<String>	POST	Reorder the list of attributes for a specific class with the new order set in the parameter attrOrder

* {attributeId}: Long id of the message

5.2.26. Class filters

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/filters

Filter data structure:

• name	String
• description	String
• target	String
• configuration	String
• active	Boolean
• shared	Boolean
• ownerType	String

Path	Parameters	Type	Description
	-limit Long -start Long -shared Boolean	GET	Obtain a list of all available filter for a specific class, with the possibility of filtering the result
/{filterId}		GET	Read a specific filter owned by a class with id classId
	-data json	POST	Create a new filter for a specific class with the provided data
/{filterId}	-data json	PUT	Update an existing filter with the data provided in element
/{filterId}		DELETE	Delete a specific filter
/{filterId}/defaultFor		GET	Obtain a list of roles for a specific filter
/{filterId}/defaultFor	-roles List<json>	POST	Update the list of roles for a specific filter

* {filterId}: Long id of the filter

5.2.27. Class or process domains

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/domains

Path	Parameters	Type	Description
	-detailed Boolean	GET	Obtain a list of all domains related to a class or process, with the possibility of specifying if obtaining the full response or a basic response with the boolean parameter includeFullDetails

5.2.28. Class print

Base url: http://hostname:port/cmdbuild/services/rest/v3/classes

Path	Parameters	Type	Description
/{classId}/print/file	-filter String -sort String -limit Long -start Long -extension String -attributes String	GET	Obtain the result of a specific report with the provided extension
/{classId}/print_schema/file	-file String -extenstion String	GET	Obtain a file with the schema of the specified class
/print_schema/file	-file String -extension String	GET	Obtain a file with the schema of the whole database

* {classId}: Long id of the class

5.2.29. Class stats

Base url: http://hostname:port/cmdbuild/services/rest/v3/classes/classId

Path	Parameters	Type	Description
/stats	WsQueryParams -Query String	GET	Obtain class stats
/relations	-StandardQueryParams	GET	Obtain class relations stats

5.2.30. Classes

Base url: http://hostname:port/cmdbuild/services/rest/v3/classes

Class data structure:

• name	String
• description	String
• defaultFilter	Long

• defaultImportTemplate	Long
• defaultExportTemplate	Long
• _icon	Long
• validationRule	String
• type	String
• allowedExtensions	String
• checkCount	String
• checkCountNumber	String
• maxFileSize	Int
• messageAttr	String
• flowStatusAttr	String
• engine	String
• parent	String
• active	Boolean
• prototype	Boolean
• noteInline	Boolean
• noteInlineClosed	Boolean
• attachmentsInline	Boolean
• attachmentsInlineClosed	Boolean
• enableSaveButton	Boolean
• dmsCategory	String
• multitenantMode	String
• stoppableByUser	Boolean
• formTriggers	List<ClassDataFormTrigger>
• contextMenuItems	List<ContextMenuItems>
• widgets	List<Widgets>
• attributeGroups	List<AttributeGroups>
• domainOrder	List<String>
• help	String
• formStructure	JsonNode

Path	Parameters	Type	Description
	-detailed Boolean -limit Long -start Long -filter String	GET	Obtain a list of every available class
/{classId}		GET	Get the details of a specific class

	-data json	POST	Create a new class with the provided data
/{classId}	-data json	PUT	Update an existing class with the provided data
/{classId}		DELETE	Delete a specific class

* {classId}: Long id of the class

5.2.31. Configurations

Base url: http://hostname:port/cmdbuild/services/rest/v3/configuration

Path	Parameters	Type	Description
/public		GET	Obtain the public configuration
/system		GET	Obtain the system configuration

5.2.32. Context menu component

Base url: http://hostname:port/cmdbuild/services/rest/v3/components/contextmenu

Custom menu component data structure:

• description	String
• active	boolean

Path	Parameters	Type	Description
		GET	Obtain a full list of the available custom menu components
/{contextMenuId}		GET	Obtain the details of a specific custom menu component
/{contextMenuId}		DELETE	Delete a specific custom menu component
/{contextMenuId}/ {targetDevice}		DELETE	Delete a specific custom menu component for the specified target device
/{contextMenuId}/ {targetDevice}/file	-extension String -parameters String	GET	Download the custom menu component for the specified target device
	-file dataHandler -data json -merge Boolean	POST	Create a new custom menu component with the provided data
/{contextMenuId}	-file dataHandler -data json	PUT	Update a specific custom menu component with the provided file

* {contextMenuId}: Long id of the context menu component
* {targetDevice}: Target device of the context menu component

5.2.33. Core components

Base url: http://hostname:port/cmdbuild/services/rest/v3/components/core/{type}

Core components data structure:

• name	String
• description	String
• data	String
• active	Boolean

Path	Parameters	Type	Description
		GET	Obtain a full list of the available core components
/{componentId}		GET	Obtain the details of a specific core component
/{componentId}		DELETE	Delete a specific core component
	-coreComponent json	POST	Create a new core component with the provided data
/{componentId}	-coreComponent json	PUT	Update a specific core component with the provided data

* {componentId}: Long id of the core component

5.2.34. Custom pages

Base url: http://hostname:port/cmdbuild/services/rest/v3/custompages

Custom page data structure:

• description	String
• active	boolean

Path	Parameters	Type	Description
		GET	Obtain a full list of the available custom pages
/{customPageId}		GET	Obtain the details of a specific custom page
/{customPageId}		DELETE	Delete a specific custom page
/{customPageId}/ {targetDevice}		DELETE	Delete a specific custom page for the specified target device
	-file dataHandler -customPageData json -merge Boolean	POST	Create a new custom page with the provided data
/{customPageId}	-file dataHandler -data json	PUT	Update a specific custom page with the provided data
/{customPageId}/	-extension	GET	Download the custom page file

{targetDevice}/file	String -parameters String		

* {customPageId}: Long id of the custom page
* {targetDevice}: Target device of the custom page

5.2.35. Dashboard

Base url: http://hostname:port/cmdbuild/services/rest/v3/dashboards

Dashboard data structure:

• name	String
• description	String
• active	boolean
• charts	Object
• layout	Object

Path	Parameters	Type	Description
	-detailed Boolean -limit Integer -start Integer	GET	List all available dashboars
/{dashboardId}		GET	Obtain a specific dashboard
	-dashboardData json	POST	Create a new dashboard with the provided data
/{dashboardId}	-dashboardData json	PUT	Update an existing dashboard with the provided data
/{dashboardId}		DELETE	Delete a specific dashboard

* {dashboardId}: Long id of the dashboard

5.2.36. Dms category values

Base url: http://hostname:port/cmdbuild/services/rest/v3/dms/categories/lookupTypeId/values

Same data structure as the LookUp values one.

Path	Parameters	Type	Description
/{lookupValueId}		GET	Obtain a dms category value
		GET	Obtain a list of all the dms category values for the specific category
	-data json	POST	Create a new dms category value
/{lookupValueId}	-data json	PUT	Update an existing dms category value
/{lookupValueId}		DELETE	Delete an existing dms category value

/order	-lookupValueIds List<Long>	POST	Reorder the dms category values

* {lookupValueId}: Long id of the dms lookupValue

5.2.37. Dms categories

Base url: http://hostname:port/cmdbuild/services/rest/v3/dms/categories

Same data structure as the LookUp type one

Path	Parameters	Type	Description
/{lookupTypeId}		GET	Obtain a specific dms category
		GET	Obtain a list of all the dms categories
	-data json	POST	Create a new dms category
		DELETE	Delete a dms category

* {lookupTypeId}: Id of the dms lookup type

5.2.38. Dms models

Base url: http://hostname:port/cmdbuild/services/rest/v3/dms/models

Same data structure as the class one

Path	Parameters	Type	Description
	-StandardQueryParams	GET	Obtain all dms models
/{classId}		GET	Obtain a specific dms model
	-data json	POST	Create a new dms model
/{classId}	-data json	PUT	Update an existing dms model
/{classId}		DELETE	Delete an existing dms model
/{classId}/attributes/ {attributeId}		GET	Read a specific attribute of the specified dms model
/{classId}/attributes	-StandardQueryParams	GET	Read all attributes of a specific dms model
/{classId}/attributes	-data json	POST	Create a new attribute for a specific dms model
/{classId}/attributes/ {attributeId}	-data json	PUT	Update a specific attribute for a dms model
/{classId}/attributes/ {attributeId}		DELETE	Delete a specific attribute
/{classId}/attributes/order	-attrOrder List<String>	POST	Reorder the dms model attributes
/{classId}/print_schema/file	-extension String	GET	Print the schema of a specific dms model

* {classId}: String Id of the dms model class
* {attributeId}: Long Id of the dms model attribute

5.2.39. Domain attributes

Base url: http://hostname:port/cmdbuild/services/rest/v3/domains/domainId/attributes

Attribute data structure:

formatPattern	String
unitOfMeasure	String
unitOfMeasureLocation	String
visibleDecimals	Integer
preselectIfUnique	boolean
showThousandsSeparator	boolean
showSeconds	boolean
showSeparators	boolean
type	String
name	String
description	String
showInGrid	boolean
showInReducedGrid	boolean
domainKey	String
domain	String
direction	String
unique	boolean
mandatory	boolean
active	boolean
index	Integer
defaultValue	String
group	String
precision	String
scale	String
targetClass	String
maxLenght	Integer
editorType	String
lookupType	String
filter	String
help	String
showIf	String
validationRules	String
mode	boolean
autoValue	String
metadata	map<String, String>

• classOrder	Integer
• isMasterDetail	Boolean
• masterDetailDescription	String
• ipType	String
• textContentSecurity	String

Path	Parameters	Type	Description
	-limit Integer -start Integer	GET	Obtain the full attributes list of a specific domain, with the possibility of filtering the result with limit and offset
/{attributeId}		GET	Get the details of a specific domain attribute
	-data json	POST	Create a new attribute for a specific domain with the provided data
/{attributeId}	-data json	PUT	Update an existing attribute with the provided data
/{attributeId}		DELETE	Delete a specific attribute owned by a domain
/{attributeId}	-attrOrder List <String>	POST	Reorder the list of domain attributes of a specific domain with the porovided order provided in attrOrder

* {attributeId}: Long Id of the domain attribute

5.2.40. Domains

Base url: http://hostname:port/cmdbuild/services/rest/v3/domains

Domain data structure:

• source	String
• name	String
• description	String
• destination	String
• cardinality	String
• descriptionDirect	String
• descriptionInverse	String
• indexDirect	int
• indexInverse	int
• descripttionMasterDetail	String
• filterMasterDetail	String
• disabledSourceDescendants	List<String>
• disabledDestinationDescendants	List<String>
• masterDetailAggregateAttrs	List<String

• active	boolean
• isMasterDetail	boolean
• sourceInline	Boolean
• sourceDefaultClosed	Boolean
• destinationInline	Boolean
• destinationDefaultClosed	Boolean
• cascadeActionDirect_askConfirm	Boolean
• cascadeActionInverse_askConfirm	Boolean
• cascadeActionDirect	String
• cascadeActionInverse	String

Path	Parameters	Type	Description
	-filter String -limit Integer -start Integer	GET	Obtain a complete list of all available domains with the possibility of filtering the results with limit and offset
/{domainId}		GET	Get the details of a specific domain
	-domainData json	POST	Create a new domain with the provided data
/{domainId}	-domainData json	PUT	Update an existing domain with the provided data
/{domainId}		DELETE	Delete an existing domain

* {domainId}: Long Id of the domain

5.2.41. Email accounts

Base url: http://hostname:port/cmdbuild/services/rest/v3/email/accounts

Email account data structure:

• name	String
• username	String
• password	String
• address	String
• smtp_server	String
• smtp_port	Integer
• smtp_ssl	boolean
• smtp_starttls	boolean
• imap_output_folder	String
• imap_server	String
• imap_port	Integer
• imap_ssl	boolean

- imap_starttls boolean

Path	Parameters	Type	Description
	-limit Long -offset Long -detailed Boolean	GET	Obtain the full list of available email accounts with the possibility of filtering the results with limit and offset
/{accountId}		GET	Get the details of a specific email account
	-emailAccountData json	POST	Create a new email account with the provided data
/{accountId}	-emailAccountData json	PUT	Update an existing email account with the provided data
/{accountId}			Delete an existing email account
/_NEW/test	-emailAccountData json	POST	Verify the configuration of a new accout
/{accountId}/test	-emailAccountData json	POST	Verify the configuration of an existing account with the additional data

* {accountId}: Long Id of the email account

5.2.42. Email queue

http://hostname:port/cmdbuild/services/rest/v3/email/queue

Path	Parameters	Type	Description
/trigger		POST	Trigger the email queue to send outgoing emails
/outgoing		GET	Get a list of every email with outgoing status
/outgoing/{emailId}/trigger		POST	Trigger a specific email

* {emailId}: Long Id of the email to send

5.2.43. Email signatures

Base url: http://hostname:port/cmdbuild/services/rest/v3/email/signatures

Email signature data structure:

active	Boolean
code	String
description	String
content_html	String

Path	Parameters	Type	Description
	-filter String -sort String -limit	GET	Obtain a full list of available email signatures with the possibility of filtering the results with limit offset and sort

	Long -start Long -detailed Boolean		
/{signatureId}		GET	Get the details of a specific email signature
	-emailSignatureData json	POST	Create a new email signature with the provided data
/{signatureId}	-emailSignatureData json	PUT	Update an existing email signature with the provided data
/{signatureId}		DELETE	Delete a specific email signature

* {signatureId}: Long Id of the email signature

5.2.44. Email templates

Base url: http://hostname:port/cmdbuild/services/rest/v3/email/templates

Email template data structure:

• name	String
• description	String
• delay	Long
• from	String
• to	String
• cc	String
• bcc	String
• subject	String
• contentType	String
• body	String
• account	String
• signature	Long
• keepSynchronization	boolean
• promptSynchronization	boolean
• provider	String
• data	Map<String, String>

Path	Parameters	Type	Description
	-filter String -sort String -limit Long -start Long -detailed Boolean	GET	Obtain a full list of available email templates with the possibility of filtering the results with limit offset and sort

	-includeBindings Boolean		
/{templateId}		GET	Get the details of a specific template
	-data json	POST	Create a new email template with the provided data
/{templateId}	-data json	PUT	Update an existing template with the provided data
/{templateId}		DELETE	Delete an existing email template

* {templateId}: Long Id of the email template

5.2.45. Etl Config

Base url:http://hostname:port/cmdbuild/services/rest/v3/etl/configs

Etl gate data structure:

• disabled	String
• enabled	Boolean
• params	Map<String, String>
• data	String

Path	Parameters	Type	Description
	-StandardQueryParams -includeMeta boolean	GET	Obtain a list of all Etl configs
/{configCode}	-includeMeta boolean -if_exists boolean	GET	Obtain the details of a specific etl config
/{configCode}/items	-StandardQueryParams	GET	Obtain a list of items of a specific etl config, with the possibility of filtering the results with the standard query parameters
	-etlConfigData json	POST	Create a new etl config with the provided data
/{configCode}	-etlConfigData json -file dataHandler	PUT	Update an existing elt config
/{configCode}		DELETE	Delete a specific etl config

* {configCode}: String Id of the etl config

5.2.46. Etl Gate

Base url:http://hostname:port/cmdbuild/services/rest/v3/etl/gates

Etl gate data structure:

• code	String
• description	String
• processingMode	String

• allowPublicAccess	Boolean
• enabled	Boolean
• config	Map<String, String>
• handlers	List<Map<String, String>>

Path	Parameters	Type	Description
	-limit Long -offset Long -detailed boolean	GET	Obtain a list of all Etl gates
by-class/{classId}	-StandardQueryParams -include_etl_templates boolean	GET	Obtain a list of etl gates for a class
/{gateId}		GET	Get the details of a specific etl gate
	-date json	POST	Create a new Etl gate with the provided data
/{gateId}	-data json	PUT	Update an existing Etl gate with the provided data
/{gateId}		DELETE	Remove an existing Etl gate

* {gateId}: Long Id of the etl gate
* {classId}: String Id of the class

5.2.47. Etl messages

Base url:http://hostname:port/cmdbuild/services/rest/v3/etl/messages

Path	Parameters	Type	Description
	-detailed Boolean	GET	Obtain a list of etl messages
/{messageReference}		GET	Obtain a specific message with the messageReference provided
/{messageReference}/ attachments/{attachmentId}		GET	Obtain an attachment of the specific message

* {messageReference}: String Id of the message

5.2.48. Etl templates

Base url:http://hostname:port/cmdbuild/services/rest/v3/etl/templates

Etl template data structure:

• errorTemplate	String
• notificationTemplate	String
• errorAccount	String
• notificationAccount	String
• fileFormat	String
• code	String

• description	String
• targetName	String
• targetType	String
• source	String
• exportFilter	String
• mergeMode	String
• mergeMode_when_missing_update_attr	String
• mergeMode_when_missing_update_value	String
• active	Boolean
• enableCreate	Boolean
• type	String
• useHeader	Boolean
• ignoreColumnOrder	Boolean
• dataRow	Integer
• firstCol	Integer
• charset	String
• csv_separator	String
• importKeyAttributes	Object
• filter	JsonNode
• columns	List<EtlColumnData>
• dateFormat	String
• timeFormat	String
• decimalSeparator	String
• thousandsSeparator	String
• handleMissingRecordOnError	Boolean
• attributes	List<AttributeData>

Path	Parameters	Type	Description
	-StandardQueryParams	GET	Obtain all etl templates
/by-class/{classId}	-StandardQueryParams -include_related_domains boolean	GET	Obtain all etl templates for a specific class
/by-process/{classId}	-StandardQueryParams -include_related_domains boolean	GET	Obtain all etl templates for a specific process
/by-view/{viewId}	-StandardQueryParams -include_related_domains boolean	GET	Obtain all etl templates for a specific view
/{templateId}		GET	Get the details of a specific template
/{templateId}/export	-filterStr String	GET	Execute an export using a template

/{templateId}/export/fileName			
/{templateId}/import	-dile dataHandler -detailed_report boolean	POST	Execute an import using a template
	-data json	POST	Create a new etl template
/{templateId}	-data json	PUT	Update an existing etl template
/{templateId}		DELETE	Delete an existing template
/inline/export /inline/export/fileName	-data String -config json	POST	
/inline/import	-file dataHandler -config json	POST	

* {classId}: String Id of the class
* {viewId}: Long Id of the view
* {templateId}: Long Id of the template

5.2.49. Fk Domain

Base url: http://hostname:port/cmdbuild/services/rest/v3/fkdomains

Path	Parameters	Type	Description
	-filter String -limit Long -start Long	GET	Obtain a list of all FKs for every domain, with the possibility of filtering the results

5.2.50. Functions

Base url:http://hostname:port/cmdbuild/services/rest/v3/functions

Path	Parameters	Type	Description
	-limit Integer -start integer -filter String -detailed Boolean	GET	Obtain a list of all available functions with the possibility of filtering the results wuth limit offset and filter
/{functionId}		GET	Get the details of a specific functions
/{functionId}/parameters	-limit Integer -start Integer	GET	Get a list of input parameters of a specific function

/{functionId}/attributes	-limit Integer -start Integer	GET	Get a list of output parameters of a specific function
/{functionId}/outputs	-parameters String -model String	GET	Call a specific function
/outputs	-model String	POST	

* {functionId}: Long Id of the function

5.2.51. Geo attributes

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/geoattributes

Geo attribute data structure:

• _icon	Long
• name	String
• description	String
• active	Boolean
• type	String
• subtype	String
• index	int
• zoomMin	int
• zoomDef	int
• zoomMax	int
• visibility	List<String>
• style	Map<String, Object>
• infoWindowEnabled	Boolean
• infoWindowContent	String
• infoWindowImage	String

Path	Parameters	Type	Description
	-limit Integer -start Integer -detailed Boolean -visible Boolean	GET	Obtain a full list of all available atrtibutes with the possibility of filtering the results with offset, limit and visible
/order	-attrOrder List<Long>	POST	Reorder the list of attributes with the details provided in attrOrder
/{attributeId}		GET	Get the details of a specific attribute
	-data	POST	Create a new attribute for a

	json		specific class with the provided data
/visibility	-geoAttributes List<Long>	POST	Update the visibility of the attributes owned by a specific class
/{attributeId}	-data json	PUT	Update a specific attribute with the provided data
/{attributeId}		DELETE	Delete a specific attribute

* {attributeId}: Long Id of the attribute

5.2.52. Geo style rules

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/geostylerules

Geo style rules data structure:

• name	String
• description	String
• function	String
• attribute	String
• alanalysistype	String
• classattribute	String
• segments	Integer
• owner	String
• rules	JsonNode

Path	Parameters	Type	Description
	-limit Integer -start Integer	GET	Obtain a full list of geo style rules
/{rulesetId}		GET	Get the details of a specific geo style rule owned by the class with id classId
	-data json	POST	Create a new geo style ruleset with the provided data for the class with id classId
/{rulesetId}	-data json	PUT	Update a specific geo style rule with the provided data
/{rulesetId}		DELETE	Delete an existing ruleset with the id rulesetId and class with id classId
/{rulesetId}/result	-cards String	GET	Get the results of the application of a ruleset with id rulesetId
/tryRules	-data json -cards String	POST	Get the results of the application of a ruleset with the provided data

* {rulesetId}: Long Id of the attribute

5.2.53. Geo values

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/_ANY/cards|instances/_ANY/geovalues

Path	Parameters	Type	Description
	-attrs Set<Long> -area String -filter String -forOwner String -attach_nav_tree Boolean	GET	Obtain a list of all currently available geo values in the specified area
/area	-attribute Set<Long> -filter String	GET	
/center	-attribute Set<Long> -filter String	GET	

5.2.54. Geo server layers

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes|classes/classId/cards|instances/cardId/geolayers

Geo server layer data structure:

• name	String
• type	String
• active	Boolean
• index	Integer
• geoserver_name	String
• description	String
• zoomMin	int
• zoomMax	int
• zoomDef	int
• visibility	List<String>

Path	Parameters	Type	Description
/{layerId}		GET	Get the details of a spefic geoserver layer for a specific card
	-visible Boolean	GET	Obtain a full list of geo server layers for a specific card
/{layerId}	-data json -file	PUT	Create a new attribute with the provided in data for a specific card

	DataHandler		
/{layerId}		DELETE	Delete an existing geo server layer with the provided data

* {layerId}: Long Id of the layer

5.2.55. Grants

Base url: http://hostname:port/cmdbuild/services/rest/v3/roles/roleId/grants

Grant data structure:

• mode	String
• objectType	String
• objectTypeName	String
• filter	String
• attributePrivileges	Map<String, String>

Path	Parameters	Type	Description
	-filter String -limit Long -start Long -include ObjectDescription Boolean -include RecordsWithoutGrant Boolean	GET	Obtain a list of available grants for a specific role
/by-target/objectType/ objectTypeName		GET	
/_ANY	-data List<json>	POST	Update the list of available grants for a specific role with the provided data

5.2.56. Impersonation

Base url: http://hostname:port/cmdbuild/services/rest/v3/sessions/current/impersonate

Path	Parameters	Type	Description
/username		POST	Impersonate a specific user with the provided username
		DELETE	Stop the current impersonation

5.2.57. Jobs

Base url: http://hostname:port/cmdbuild/services/rest/v3/jobs

Job data structure:

• code	String
• description	String
• type	String

• enabled	boolean
• config	Map<String, Object>

Path	Parameters	Type	Description
	-limit Long -start Long	GET	Obtain a list of every Job available
/{jobId}		GET	Get the details of a specific job
	-data json	POST	Create a new job with the provided data
/{jobId}	-data json	PUT	Update an existing job with the provided data
/{jobId}		DELETE	Delete a specific existing job
/{jobId}/run		POST	Run a specific job
/{jobId}/runs	-limit Long -start Long	GET	Get a list of every execution of a specific job
/{jobId}/errors	-limit Long -start Long	GET	Get a list of all the errors generated by a specific job
/_ANY/runs	-limit Long -start Long	GET	Get a list of every run of every job
/_ANY/errors	-limit Long -start Long	GET	Get a list of all the errors generated by all jobs
/{jobId}/runs/{runId]		GET	Get the details of a specific run of a specific job

5.2.58. Language configurations

Base url: http://hostname:port/cmdbuild/services/rest/v3/configuration/languages

Path	Parameters	Type	Description
		GET	Obtain a list of all available languages

5.2.59. Languages

Base url: http://hostname:port/cmdbuild/services/rest/v3/languages

Path	Parameters	Type	Description
	-active Boolean	GET	Obtain a list of all available languages

5.2.60. Locks

Base url: http://hostname:port/cmdbuild/services/rest/v3/locks

Path	Parameters	Type	Description
		GET	Get all available locks
/{lockId}		GET	Get details of a specific lock
/{lockId}		DELETE	Delete a specific lock
/_ANY		DELETE	Delete all locks

* {lockId}: Long Id of the lock

5.2.61. Lookup types

Base url: http://hostname:port/cmdbuild/services/rest/v3/lookup_types

Lookup type data structure:

• name	String
• parent	String

Path	Parameters	Type	Description
/{lookupTypeId}		GET	Get the details of a specific lookup type
	-limit Long -start Long -filter String	GET	Obtain a list of all available lookup types
	-data json	POST	Create a new lookup type with the provided data
/{lookupTypeId}	-data json	PUT	Update an existing lookup type with the values provided in wsLookupType
/{lookupTypeId}		DELETE	Delete an existing lookup type

5.2.62. Lookup values

Base url: http://hostname:port/cmdbuild/services/rest/v3/lookup_types/lookupTypeId/values

Lookup data structure:

• code	String
• description	String
• index	Integer
• active	boolean
• parent_id	Long
• default	boolean
• note	String
• text_color	String
• icon_type	String
• icon_image	String
• icon_font	String

- icon_color String

Path	Parameters	Type	Description
/{lookupValueId}		GET	Obtain a list of the lookup values for a specific lookup type
	-limit Long -start Long -filter String -active Boolean	GET	Obtain a full list of the lookup values available
	-data json	POST	Create a new lookup value for a specific lookup type with the provided data
/{lookupValueId}	-data json	PUT	Update an existing lookup value with the provided data
/{lookupValueId}		DELETE	Delete an existing lookup value
/order	-lookupValueIds List<Long>	POST	Reorder the lookup value list of a specific lookup type with the provided data

* {lookupValueId}: Long Id of the lookup value

5.2.63. Menu

Base url: http://hostname:port/cmdbuild/services/rest/v3/menu

Menu Node data structure:

• _id	String
• menuType	MenuItemType
• objectTypeName	String
• objectDescription	String
• children	List<MenuNodes>

Menu Root Node data structure:

• group	String
• children	List<MenuNodes>
• type	String
• device	String

Path	Parameters	Type	Description
	-detailed Boolean	GET	Obtain a full list of all available menus
/{menuId}		GET	Get the details of a specific menu
/gismenu		GET	Get the details of the gismenu is it exists
	-data	POST	Create a new menu with the

	json		provided data
/{menuId}	-data json	PUT	Update an existing menu with the provided data
/{menuId}		DELETE	Delete an existing menu

* {menuId}: Long Id of the menu

5.2.64. Minions

Base url: http://hostname:port/cmdbuild/services/rest/v3/system_services

Path	Parameters	Type	Description
		GET	Obtain a list of the status of every available service
/{serviceId}		GET	Get the status details of a specific service
/{serviceId}/start		POST	Start a specific service
/{serviceId}/stop		POST	Stop a specific service

* {serviceId}: Long Id of the service

5.2.65. Nav trees

Base url: http://hostname:port/cmdbuild/services/rest/v3/domainTrees

Tree data structure:

• name	String
• description	String
• nodes	List<TreeNodes>
• active	boolean
• type	String

Tree node data structure:

• _id	String
• filter	String
• targetClass	String
• description	String
• domain	String
• direction	String
• recursionEnabled	Boolean
• showOnlyOne	Boolean
• nodes	List<TreeNodes>

Path	Parameters	Type	Description
	-filter String -limit Long -start Long	GET	Get a list of every available nav tree

/{treeId}	-treeMode String	GET	Obtain the details of a specific nav tree
	-data json	POST	Create a new nav tree with the provided data
/{treeId}	-data json	PUT	Update an existing nav tree with the provided data
/{treeId}		DELETE	Delete a specific nav tree

* {treeId}: Long Id of the tree

5.2.66. Process configuration

Base url: http://hostname:port/cmdbuild/services/rest/v3/configuration/processes

Path	Parameters	Type	Description
/statuses		GET	Obtain a list with the statuses of every available process

5.2.67. Process instance activity email

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes/processId/instances/instanceId/activities/activityId/emails

Path	Parameters	Type	Description
/sync	-flowData	POST	Update a specific email with a specific card data

5.2.68. Process instance activity

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes/processId/instances/processInstanceId/activities

Path	Parameters	Type	Description
		GET	Obtain a list of every available task for a specific class
/{processActivityId}		GET	Get all the details of a specific task

5.2.69. Process instance history

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes/processId/instances/instanceId/history

Path	Parameters	Type	Description
	-limit Long -start Long	GET	Obtain a list of the history of processes for a specific card
/{recordId}		GET	Get the details of a specific record in the history

5.2.70. Process instances

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes/processId/instances

Process instance data structure:

• values	Map<String, Object>
• _advance	boolean

- _activity String

Path	Parameters	Type	Description
	-data json	POST	Create a new process instance with the provided data
/{processInstanceId}	-data json	PUT	Update a process instance with the provided data
/{processInstanceId}	-include_tasklist Boolean	GET	Get the details of a specific process
/{processInstanceId} /graph	-simplified Boolean	GET	
	-filter String -sort String -limit Long -start Long -positionOf Long -positionOf_goToPage Boolean -include_tasklist Boolean	GET	Obtain the list of all available process instances with the possibility of filtering the results with filter, sort, limit, offset, positionOfCard, goToPage
/{processInstanceId}		DELETE	Delete an existing process instance
/{processInstanceId}/ suspend		POST	Suspend the specific process instance
/{processInstanceId}/resume		POST	Resume the specific process instance
		DELETE	Delete the specific process instance

5.2.71. Process start activities

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes/processId/start_activities

Path	Parameters	Type	Description
		GET	Get the start activities of a specific process

5.2.72. Process task definition

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes/processId/activities

Task definition data structure:

- formStructure JsonNode

Path	Parameters	Type	Description
	-limit Long -start Long	GET	Get all task definitions for a specific process

/{taskId}		GET	Obtain the details of a specific task
/{taskId}	-data json	PUT	Update an existing task

* {taskId}: String Id of the task

5.2.73. Process task

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes/processId/instance_activities

Path	Parameters	Type	Description
	-limit Long -start Long	GET	Get all activities of a specific process

5.2.74. Processes

Base url: http://hostname:port/cmdbuild/services/rest/v3/processes

Path	Parameters	Type	Description
	-activeOnly Boolean -limit Long -start Long -detailed Boolean	GET	Obtain a list of every available process
/{processId}		GET	Get the details of a specific process
	-data json	POST	Create a new process with the provided data
/{processId}	-data json	PUT	Update an existing process with the provided data
/{processId}		DELETE	Delete an existing process
/{processId}/versions	-file DataHandler	POST	Upload a new xpdl
/{processId}/migration	-provider String -file DataHandler	POST	Upload a new xpdl and force the process to use the new xpdl version
/{processId}/versions		GET	Obtain a list of all xpdl version
/{processId}/versions/planId/file		GET	Obtain an xpdl file
/{processId}/template		GET	Obtain an xpdl template file

* {processId}: Long Id of the process

5.2.75. Relation history

Base url: http://hostname:port/cmdbuild/services/rest/v3/domains/domainId/relations

Path	Parameters	Type	Description
/history/{relationId}		GET	Obtain a history record for the specific relation

5.2.76. Relations

Base url: http://hostname:port/cmdbuild/services/rest/v3/domains/domainId/relations

Path	Parameters	Type	Description
	-limit Long -start Long -detailed Boolean	GET	Obtain a list of all available relations for a specific domain
/{relationId}		GET	Get the details of a specific relation
	-data json	POST	Create a new relation with the provided data
/{relationId}	-data json	PUT	Update an existing relation with the provided data
/{relationId}		DELETE	Delete a specific relation
/_ANY/move	-data json	POST	Move the specified relation
/_ANY/copy	-data json	POST	Copy a specific relation

* {relationId}: Long Id of the relation

5.2.77. Reports

Base url: http://hostname:port/cmdbuild/services/rest/v3/reports

Report data structure:

• _id	Long
• code	String
• description	String
• _description_translation	String
• active	boolean
• title	String
• query	String

Path	Parameters	Type	Description
	-filter String -limit Long -start Long -detailed Boolean	GET	Obtain a list of all available reports

/{reportId}		GET	Get the details of a specific report
/{reportId}/attributes	-limit Long -start Long	GET	Get all attributes of a specific report
/{reportId}/file:	-extension String -parameters String	GET	Download a report on a specified file with a specified extension
	-data json -attachments List<Attachment>	POST	Create a new report with the provided attachments
/{reportId}	-attachments List<Attachment>	PUT	Update an existing report with the provided data
/{reportId}/template-data json -attachments List<Attachment>	-data json -attachments List<Attachment>	PUT	Update an existing report template with the provided data
/{reportId}/template:		GET	Download a specific report template
/{reportId}		DELETE	Delete an existing report

* {reportId}: Long Id of the relation

5.2.78. Resources

Base url: http://hostname:port/cmdbuild/services/rest/v3/resources

Path	Parameters	Type	Description
/company_logo/file	-roleId	GET	Obtain the logo of the company

5.2.79. Role class filters

Base url: http://hostname:port/cmdbuild/services/rest/v3/roles/roleId/filters

Path	Parameters	Type	Description
		GET	Obtain a list of all available filter for a specific role
	-data json	POST	Update the list of filters available for a specific role with the provided data

5.2.80. Roles

Base url: http://hostname:port/cmdbuild/services/rest/v3/roles

Role data structure:

• type	String
• name	String
• description	String
• email	String

•	active	boolean
•	processWidgetAlwaysEnabled	boolean
•	startingClass	String

Path	Parameters	Type	Description
/roleId		GET	Get the details of a specific role
	-limit Long -offset Long	GET	Obtain a list of all available roles
/roleId/users	-filter String -sort String -limit Long -start Long -assigned Boolean	GET	Obtain a list of all available roles for a specific user
/roleId/users	-users json	POST	Update the roles of a specific user
	-jsonData String	POST	Create a new role with the provided data
/roleId	-jsonData String	PUT	Update an existing role with the provided data

5.2.81. Search

Base url: http://hostname:port/cmdbuild/services/rest/v3/search

Path	Parameters	Type	Description
/itemType	-StandardQueryOptions	GET	
/itemType1/itemType2	-StandardQueryOptions	GET	

5.2.82. Session menu

Base url: http://hostname:port/cmdbuild/services/rest/v3/sessions/sessionId/menu

Path	Parameters	Type	Description
	-flat Boolean	GET	Obtain a list of every session menu available

5.2.83. Session preferences

Base url: http://hostname:port/cmdbuild/services/rest/v3/sessions/sessionId/preferences

Path	Parameters	Type	Description
		GET	Obtain a list of all available preferences for a specific session
/{key}	-key String	GET	Get the value of a specific user configuration

/{key}	-key String -value String	PUT	Update the value of a specific user configuration
	-data Map<String, String>	POST	Update the vlue of multiple user configurations
/{key}		DELETE	Delete a specific configuration key

* {key}: String session preference key

5.2.84. Sessions

Base url: http://hostname:port/cmdbuild/services/rest/v3/sessions

Session data structure:

• username	String
• password	String
• role	String
• scope	String
• device	String
• tenant	Long
• ignoreTenants	Boolean
• activeTenants	List<Long>

Path	Parameters	Type	Description
	-data json -includeExtendedData Boolean -scopeStr String -returnId Boolean	POST	Create a new session with the provided data
/{sessionId}	-include ExtendedData Boolean	GET	Get the details of a specific session
/{sessionId}/privileges		GET	Get the details of the privileges of a specific session
		GET	Obtain a list of all available sessions
/{sessionId}	-data json -includeExtendedData Boolean	PUT	Update an existing session with the provided data
/{sessionId}		DELETE	Delete an existing session
/all		DELETE	Delete all existing sessions
/{sessionId}/keepalive		POST	Keep a session alive

* {sessionId}: String session id

5.2.85. System configuration

Base url: http://hostname:port/cmdbuild/services/rest/v3/system/config

Path	Parameters	Type	Description
	-detailed Boolean	GET	Get the full system configuration
/{key}	-includeDefault Boolean	GET	Get a specific configuration of the system
/{key}	-value String -encrypt Boolean	PUT	Update a specific configuration of the system
/_MANY	-data Map<String, String>	PUT	Update the system configuration with the provided data
/{key}		DELETE	Delete a specific ststem configuration
/reload		POST	Reload the system configuration

* {key}: String system configuration

5.2.86. System

Base url: http://hostname:port/cmdbuild/services/rest/v3/system

Path	Parameters	Type	Description
/status		GET	Obtain a list of the system services status
/cluster/status		GET	Get the status of the cluster system
/cache/drop		POST	Invalidate all system cache
/cache/cacheId/drop		POST	Invalidate a specific cache
/cache/stats		GET	Obtain a list of the cache statuses
/stop		POST	Stop the CMDBuild system
/reload		POST	Reload the CMDBuild system
/restart		POST	Restart the CMDBuild system
/upgrade	-file DataHandler	POST	Upgrade the CMDBuild system with the provided file
/audit/drop		POST	Drop the audits of the system
/audit/cleanup		POST	Cleanup the audits of the system
/patches		GET	Obtain a list of all patches
/tenants		GET	Obtain a list of all available tenants
/scheduler/jobs		GET	Obtain a list of all the jobs available in the scheduler
/scheduler/job/jobId/trigger		POST	Trigger the execution of a job
/loggers		GET	Obtain a list of all available loggers

/loggers/key	-loggerCategory String -loggerLevel String	POST	Update the level of a specific logger
/loggers/key	-loggerCategory String -loggerLevel String	PUT	Add a logger level to an existing logger
/loggers/key	-loggerCategory String	DELETE	Delete a level of a specific logger
/logger/stream		POST	Enable the stream of the active loggers
/logger/stream		DELETE	Stop the stream of the active loggers
/database/dump		GET	Create a dump of the current database used by the system
/database/reconfigure	-dbConfig Map<String, String>	POST	Reconfigure the database with the configuration provided by dbConfig
/database/import	-file DataHandler	POST	Reconfigure a database with the dump file provided by dataHandler
/debuginfo/download		GET	Generate a bug report
/debuginfo/send	-message String	POST	Send a bug report with the provided message
/messages/broadcast	-message String	POST	Send a broadcast alert with the provided message

5.2.87. Tenants

Base url: http://hostname:port/cmdbuild/services/rest/v3/tenants

Path	Parameters	Type	Description
	-limit Long -start Long	GET	Obtain a list of all available tenants
/configure	-configData json	POST	Set the multitenant configuration with the provided data

5.2.88. Timezones

Base url: http://hostname:port/cmdbuild/services/rest/v3/timezones

Path	Parameters	Type	Description
		GET	Obtain a list of all available timezones

5.2.89. Translations

Base url: http://hostname:port/cmdbuild/services/rest/v3/translations

Path	Parameters	Type	Description

	-limit Long -start Long -filter String	GET	Obtain a list of all available translations
/by-code	-StandardQueryParams -lang String - includeRecordsWithoutTransla tion Boolean -section String	GET	Obtain a list of specific translations, divided by language and section
/code	-lang String	GET	Get the translation for a specific key in a specific language
/code	-data jdon	PUT	Set a specific translation with the provided data
/code	-lang String	DELETE	Remove a specific translation for a specific language
/export	-language String -format String -filter String -separator String	GET	Obtain a file of the specified format containing the translations for a specific language
	-separator String -file dataHandler	POST	Upload a file containing translations

5.2.90. Uploads

Base url: http://hostname:port/cmdbuild/services/rest/v3/uploads

Upload item data structure:

• path	String
• description	String

Path	Parameters	Type	Description
	-dir String	GET	Obtain a list of all available files in a specific directory
/fileId		GET	Get the details of a specific file
/fileId/file:		GET	Download a specific file
/_MANY/file:.zip	-dir	GET	Download multiple specified files
/_ANY/file:.zip		GET	Download all available files
	-file DataHandler -directoryPath	POST	Load a new file in the system

	String		
/_MANY	-dataHandler DataHandler	POST	Load a zip file containing one or more files
/fileId	-file DataHandler	PUT	Update an existing file with the provided data
/fileId		DELETE	Delete a specific file

5.2.91. Users

Base url: http://hostname:port/cmdbuild/services/rest/v3/users

User data structure:

• username	String
• description	String
• email	String
• password	String
• initialPage	String
• changePasswordRequired	Boolean
• active	boolean
• service	boolean
• language	String
• multiGroup	boolean
• multiTenant	boolean
• multiTenantActivationPrivileges	String
• defaultUserGroup	Long
• userTenants	List<TenantInfo>
• userGroups	List<UserRole>

Path	Parameters	Type	Description
	-filter String -sort String -limit Long -start Long -detailed Boolean	GET	Obtain a list of all available users with the possibility of filtering the results with filter, sort, limit and offset
/{userId}		GET	Get the details of a specific user
	-data json	POST	Create a new user with the provided data
/{userId}	-data json	PUT	Update an existing user with the provided data
/current/password	-data json	PUT	Update the password of the current user with the provided data

/{userId}/password		POST	
/{userId}/password/recovery		POST	

* {userId}: String user id

5.2.92. Card views

Base url: http://hostname:port/cmdbuild/services/rest/v3/views/viewId/cards

Path	Parameters	Type	Description
	-data Map<String,Object>	POST	Create a new view with the provided data
/{cardId}		GET	Get the card details with the specified view
	-filter String -sort String -limit Long -start Long -positionOf Long -forDomainName String -forDomainDirection String -forDomainOriginId Long	GET	Obtain a list of information with the specified view
/{cardId}	-data Map<String,Object>	PUT	Update the information of a specific view with the information contained in data
/{cardId}		DELETE	Delete a specific card view

* {cardId}: Long card id

5.2.93. Views

Base url: http://hostname:port/cmdbuild/services/rest/v3/views

View data structure:

• name	String
• description	String
• sourceClassName	String
• sourceFunction	String
• filter	String
• active	Boolean
• shared	Boolean
• type	String
• masterClass	String
• masterClassAlias	String

• sorter	JsonNode
• join	List<JoinElement>
• attributes	List<JoinAttribute>
• attributeGroups	List<JoinAttributeGroup>
• formStructure	JsonNode
• contxtMenuItems	List<ClassDataContextMenuItem>

Path	Parameters	Type	Description
		GET	Obtain a list of all available views
/{viewId}		GET	Get the details of a specified view
	-data json	POST	Create a new view with the provided data
/{viewId}	-data json	PUT	Update an existing view with the provided data
/{viewId}		DELETE	Delete an existing view

* {viewId}: Long view id

5.2.94. Widget

Base url: http://hostname:port/cmdbuild/services/rest/v3/components/widget

Widget data structure:

• description	String
• active	Boolean

Path	Parameters	Type	Description
		GET	Obtain the list of all widgets
/{widgetId}		GET	Obtain the details of a specific widget
/{widgetId}		DELETE	Delete the specific widget
/{widgetId}/{targetDevice}		DELETE	Delete the specific widget for the provided targetDevice
/{widgetId}/version/file		GET	Obtain the file of the specified widget
	-widgetData json -file dataHandler	POST	Create a new widget with the specified data
/{widgetId}	-widgetData json -file dataHandler	PUT	Update the specific widget with the provided data

5.3. REST Examples

In this paragraph various examples of REST calls will be presented. Note that CMDBuild in this scenario is configured with the database: demo.dump.xz, so if you want to replicate the same examples with the same data you must load that dump first.

To perform the following examples various tools can be used, via terminal with curl on linux operating systems, or with the support of a graphical interface with programs like Postman, or any other software that can perform HTTP requests.

Every request requires the user to specify in the header the field “Cmdbuild-authorization”, that field is a session token generated when creating a session, the first example request will show how to obtain that through a specific request.

5.3.1. Generating a session token

The endpoint to use for generating a session token is:

http://hostname:port/cmdbuild/services/rest/v3/sessions

The request type has to be POST, because username and password will be provided to create a new session. If in the response we want to obtain the session token, from version 3.2 a query parameter has to be set to true in the request, the parameter is ‘returnId’. If this parameter is not set to true the sessionId will be hidden and the value ‘current’ will be returned instead

The request will be like the following:

```
POST http://hostname:port/cmdbuild/services/rest/v3/sessions?
scope=service&returnId=true HTTP/1.1
Content-Type:application/json
{
 username : admin,
 password : admin
}
```

The response will be like the following, where the _id will be the generated session id and after the list of available roles information like multigroup and role priviledges will be displayed:

```
HTTP/1.1 200 OK
Content-Type:application/json
{
 “success”: true,
 “data” : {
       “_id”:“sessionId”,
       “username”:”admin”,
       “userDescription”:”Administrator”,
       “role”:”SuperUser”,
       “availableRoles”:[
       “SuperUser”
```

```
      ],
 ...
 }
}
```

An inactive session will be deleted after a certain amount of time, causing the user to re-create the session every once in a while.

With futher requests users can provide the generated id (the value in the field _id) in the header to obtain access to every rest endpoint.

5.3.2. Obtaining a list of every class

If the user wants to obtain a list of the available classes the endpoint that will be used is:

http://hostname:port/cmdbuild/services/rest/v3/classes

The request type has to be GET, and will look like the following:

```
GET http://hostname:port/cmdbuild/services/rest/v3/classes?scope=service
HTTP/1.1
Cmdbuild-authorization:sessionId
```

The response will be like the following, the results should be 24, but for the documentation purpose only the first results are displayed.

```
HTTP/1.1 200 OK
Content-Type:application/json
{
 “success”: true,
 “data” : {
   “_id”:“Invoice”,
   "name": "Invoice",
   "description": "Invoice",
   "_description_translation": "Invoice",
   "prototype": false,
   "parent": "Class",
   "active": true,
   "type": "standard",
   "_can_read": true,
   "_can_create": true,
   "_can_update": true,
   "_can_clone": true,
   "_can_delete": true,
   "_can_modify": true,
   "defaultFilter": null,
    "description_attribute_name": "Description",
    "metadata": {},
    "_icon": null
  },
. . .
,
 "meta": {
  "total": 24
 }
}
```

In the response of multiple items at the bottom of the response a “meta” field will always be provided, various information such as the number of total results can be found here.

Note that the parameters previously described in the documentation can be provided (in this case the available parameters are activeOnly, detailed, limit and offset). If, for example, we wanted the amount of results to be limited to two the request would look the same with the addition of the parameter in the endpoint like:

```
http://hostname:port/cmdbuild/services/rest/v3/classes?
scope=service&limit=2
```

The same with the addition of other parameters.

5.3.3. Obtaining the information of a specific class

If instead of a class list, the user wants to obtainthe information of a specific class only, the endpoint will be the same as the full list with the addition of the classId in the path:

http://hostname:port/cmdbuild/services/rest/v3/classes/classId

Where the value of classId will be the value of the class that we want to obtain, for the example the class we use will be the one previously returned in the response (the class with _id=invoice), so the new request endpoint will be:

http://hostname:port/cmdbuild/services/rest/v3/classes/Invoice

The request type has to be GET, and will look like the following:

```
GET http://hostname:port/cmdbuild/services/rest/v3/classes/Invoice?
scope=service HTTP/1.1
Cmdbuild-authorization:sessionId
```

The response will contain the information of only that class like shown in the response with every class:

```
HTTP/1.1 200 OK
Content-Type:application/json
{
 “success”: true,
 “data” : {
   “_id”:“Invoice”,
   "name": "Invoice",
   "description": "Invoice",
   "_description_translation": "Invoice",
   "prototype": false,
   "parent": "Class",
   "active": true,
   "type": "standard",
   . . .
```

5.3.4. Creating a new class

If the objective of the request has to be the creation of a new class, the endpoint is:

http://hostname:port/cmdbuild/services/rest/v3/classes

The new class information have to be provided, in the header it is also required to add the content-type, as in the session creation and the request type will be POST:

```
POST http://hostname:port/cmdbuild/services/rest/v3/classes HTTP/1.1
Cmdbuild-authorization:sessionId
Content-Type:application/json
{
 “name”:”testClass”,
 “type”:”standard”
}
```

In this case the class created has only the two basic information, the name and the type, in the request we can add whatever information we want that is supported by the class.

When the request is made the response will contain the newly added class:

```
HTTP/1.1 200 OK
Content-Type:application/json
 {
 "success": true,
 "data": {
  "_id": "testClass",
  "name": "testClass",
  "description": "",
  "_description_translation": "",
  "prototype": false,
  "parent": "Class",
  "active": true,
  "type": "standard",
   . . .
```

So that if we would perform a get request for that specific class:

```
GET http://hostname:port/cmdbuild/services/rest/v3/classes/testClass?
scope=service HTTP/1.1
Cmdbuild-authorization:sessionId
```

We would obtain those information that just got added.

5.3.5. Update an existing class

If a class has been already created, there is the possibility of updating the information of this class via a PUT request, the endpoint will be

http://hostname:port/cmdbuild/services/rest/v3/classes/classId

And in the request every element that needs changing can be included, for example if the objective is change the description of our previously created class (testClass) to “test description” this will be the request:

```
PUT http://hostname:port/cmdbuild/services/rest/v3/classes/testClass
HTTP/1.1
Cmdbuild-authorization:sessionId
Content-Type:application/json
{
 “name”:”testClass”,
 “type”:”standard”,
 “description”:”test description”
}
```

The request will contain the information about the updated class:

```
HTTP/1.1 200 OK
Content-Type:application/json
 {
 "success": true,
 "data": {
  "_id": "testClass",
  "name": "testClass",
  "description": "test description",
  "_description_translation": "",
  "prototype": false,
  "parent": "Class",
  "active": true,
  "type": "standard",
   . . .
```

So that if we would perform a get request for that specific class:

```
GET http://hostname:port/cmdbuild/services/rest/v3/classes/testClass?
scope=service HTTP/1.1

Cmdbuild-authorization:sessionId
```

We would obtain those information that just got added.

6. Appendix: Glossary

6.1.1. ATTACHMENT

An attachment is a file associated to a card.

In order to manage the attachments, CMDBuild uses in embedded mode any document system which is compatible with the standard protocol CMIS (or the DMS Alfresco until the version 3 through its native webservice).

The management of the attachments supports the versioning of those files that have been uploaded a few times, with automatic numbering.

6.1.2. 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

6.1.3. 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

6.1.4. 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

6.1.5. 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

6.1.6. 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

6.1.7. 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

6.1.8. 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

6.1.9. 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.

6.1.10. 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

6.1.11. 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

6.1.12. 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

6.1.13. 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

6.1.14. 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

6.1.15. 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.

6.1.16. 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

6.1.17. 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

6.1.18. 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

6.1.19. 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

6.1.20. 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

6.1.21. 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

6.1.22. 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

6.1.23. 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

6.1.24. 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.

6.1.25. 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.