Version

2.4

CMDBuild®

» Technical Manual

March 2016

Author Tecnoteca srl

www.tecnoteca.com

ENG

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

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

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

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

Contents

Introduction

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

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

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

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

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

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

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

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

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

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

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

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

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

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

CMDBuild modules

The CMDBuild application includes two main modules:

- the Administration Module, used to define the data model and set config options (classes and relations, users and permissions, reports and workflows, main options and preferences)
- the Management Module, used to manage cards and relations, add attachments, run workflow processes, visualize dashboards and execute reports

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

Available documentation

This guide is intended for software engineers interested in installing the system and to know the technical implementation of some of its components.

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

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

System configuration

In order to install the CMDBuild system you can use either one server or more. On these servers you install the components of the system:

- web server
- computing components
- database
- webservice
- documents archive

In the following paragraphs we present the software requirements needed by the CMDBuild system and how to install and configurate its components.

When planning the system configuration information security issues must be taken into account. The activation of a web application like CMDBuild demands the availability of hardware and network components with suitable levels of security. This is in order to avoid unwanted external accesses (firewall, DMZ) and to deliver good system on-line availability and suitable response times.

Hardware requirements

For the CMDBuild installation a physical or virtual server is required, with the following characteristics:

- recent generation CPU
- minimum RAM 4 GB; 8 GB for instances used by several users
- minimal disk storage 100 GB, it should be much higher if you need to manage extensive archives of documents (fed by the management of the attachments).

We also advise that:

- the disk storage should be in RAID configuration
- the CMDBuild system data should be backed up daily
- an UPS is employed in order to avoid sudden electric power failures

Software requirements

CMDBuild installation and use require the following software components.

1) Operating system

You can use any operating system supporting the software listed below (Linux operating system is best because CMDBuild is more extensively tested on it).

2) Database

PostgreSQL 9.0 or more recent (best PostgreSQL 9.3).

You should check whether "plpgsql" is active and whether the database is set with UTF-8 encoding.

CMDBuild uses the library "tomcat-dbcp" to connect to the database, this library is distributed with

Tomcat but is not included in some Linux distributions. In such cases the library can be found in the official Tomcat distribution or in the extras/tomcat-libs/{Tomcat version dir} folder inside the CMDBuild zip file; the library must be placed in /usr/share/{Tomcat version dir}/lib.

CMDBuild supports only the PostgreSQL database, because it is the only one that implements the functionality of "derivation" of tables in the "object oriented" meaning. This is used for managing the subclasses and for managing the historicizing of cards.

If you use the GIS extensions of CMDBuild you have to instal also the PostGIS spatial extension in the version 1.5.2 or 2.0. If you create a new database, you have to follow these operations (using for example the “psql” tool):

```
$ psql ... cmdbuild
cmdbuild=# CREATE SCHEMA gis;
cmdbuild=# SET SEARCH_PATH TO gis, public;
cmdbuild=# \i ${POSTGIS_DIR}/postgis.sql
cmdbuild=# \i ${POSTGIS_DIR}/spatial_ref_sys.sql
cmdbuild=# \i ${POSTGIS_DIR}/legacy.sql (if you use PostGIS 2.0)
cmdbuild=# ALTER DATABASE your_database_name_here SET
search_path="$user", public, gis;
```

If you have to restore an existing database (using for example the “psql” tool):

```
$ psql ... cmdbuild
cmdbuild=# CREATE SCHEMA gis;
cmdbuild=# SET SEARCH_PATH TO gis, public;
cmdbuild=# \i ${POSTGIS_DIR}/postgis.sql
cmdbuild=# \i ${POSTGIS_DIR}/legacy.sql (if you use PostGIS 2.0)
cmdbuild=# ALTER DATABASE ${DB_NAME} SET search_path="$user", public, gis;
cmdbuild=# DROP TABLE gis.geometry_columns;
cmdbuild=# DROP TABLE gis.spatial_ref_sys;
```

Web site of reference: http://www.postgresql.org/

3) Servlet Container / Web Server

CMDBuild needs the Apache Tomcat 7.0.32 or more recent (best Tomcat 8.0). At the moment, for issues derived from using Shark, you have to check whether Tomcat is carried out as "root" user and not as not privileged user (e.g. “tomcat”).

In order to support the UTF-8 characters when using attachments (see Installation of DMS system), edit the configuration file “server.xml” and specify the attribute URIEncoding="UTF-8" for the main "Connector" element.

You can use the web server Apache 2.2 in order to access many CMDBuild instances through virtual hosts supporting different domains.

Reference Web site for both: http://www.apache.org/

4) Document Management System (DMS)

It can be interfaced through the CMIS protocol (Alfresco in the last versions or other DMS systems that support that protocol) or through FTP protocol and proprietary webservice (Alfresco 3.4)

In the first case, in order to use the function of managing the documents attached to the cards in CMDBuild you need to install the document management system that supports the CMIS protocol, preferably version 1.1. As for CMIS in Alfresco, it supports the version 4.2 or more recent.

In the second case, you can use the document management offered by Alfresco 3.4.

Reference website: http://www.alfresco.com/

5) Java Libraries

The Java Libraries are required by Apache Tomcat.

CMDBuild requires JDK 1.8 Oracle.

Reference website: http://www.oracle.com/

6) Libraries included in the release

The CMDBuild file downloadable from the project website contains some libraries already inside the installation package, namely:

- the library for the JDBC connection to the PostgreSQL database
- the JasperReports libraries version 5.1.0 for the production of reports (http://www.jasperforge.org/)
- TWS Together Workflow Server 4.4 libraries which implement the workflow engine used by CMDBuild (http://www.together.at/prod/workflow/tws)
- the webservice available from the DMS Alfresco system in order to use its repository (http://www.alfresco.com/)
- the libraries ExtJS 4.1 version for the generation of the Ajax user interface (http://extjs.com/)
- the server and client components for the publication of georeferenced cartography (http://geoserver.org/ e http://openlayers.org/)

For designing custom reports you can use the visual editor iReport; it produces its descriptor in compatible format with the JaspertReports engine (http://jasperforge.org/projects/ireport).

For designing personalized workflows we suggest using the visual editor TWE Together Workflow Editor 4.4 (http://www.together.at/prod/workflow/twe). The editor produces in output a XPDL 2.0 file compatible with the Together Workflow Server 4.4 engine.

For integrating systems of automatic inventory we suggest using the OCS Inventory version 1.3.3 (http://www.ocsinventory-ng.org/).

Some functionalities of CMDBuild can be integrated as portlets within systems compatible with Portal JSR, among them Liferay version 6.0.6 or more recent (http://www.liferay.com/).

All software listed above are released with Open Source licence (the operating system is not included if you choose to use not the Linux operating system).

Client requirements

CMDBuild is a web-based application, so both modules are available using a standard web browser.

The system user has to arrange on his/her processor only an updated web browser (Mozilla Firefox, from version 3.6 to version 45, Chrome, from version 9 to version 49, Microsoft Explorer, from version 7 to version 10).

The web architecture ensures complete usability to any IT organization that operates in multiple locations (ie collaborative workflow); any entrusted client can connect and interact with the system using a standard web browser.

Installing CMDBuild using graphical interface

Getting started

The installation of CMDBuild requires that you have already installed the basic products needed for its operation, namely:

- PostgreSQL database (it must be started and accessible)
- Tomcat application server (not to be started)
- the DMS Alfresco (or others supporting the CMIS protocol, if you intend to use the management of attached documents)
- the Java environment

As a first step is therefore necessary to ensure downloading and installing these products, retrieving them from the links mentioned in the previous chapter.

Warning: you must to be careful to use directories not containing spaces within the entire path.

Then you start the PostgreSQL service and possibly (but it is not mandatory) the DMS service.

CMDBuild installation

After completion of the operations stated above, the installation and setup the standard CMDBuild is very simple.

Order to install CMDBuild is sufficient:

- downloading from the project site (http://www.cmdbuild.org/download) the compressed file (ZIP file) corresponding to the last version released
- copying the directory CMDBuild-{version}.war (it is in "root" of the ZIP file) in the directory "webapps" of Tomcat, renaming cmdbuild.war
- copying the directory CMDBuild-shark (found in the ZIP file in the directory "extras" of the ZIP file) in the directory "webapps" of Tomcat
- copying additional libraries for the chosen version of Tomcat (located in the directory "extras / tomcat-libs") in the "lib" directory of Tomcat

Once you do this, and only at this point, it will be necessary to also start the Tomcat application server.

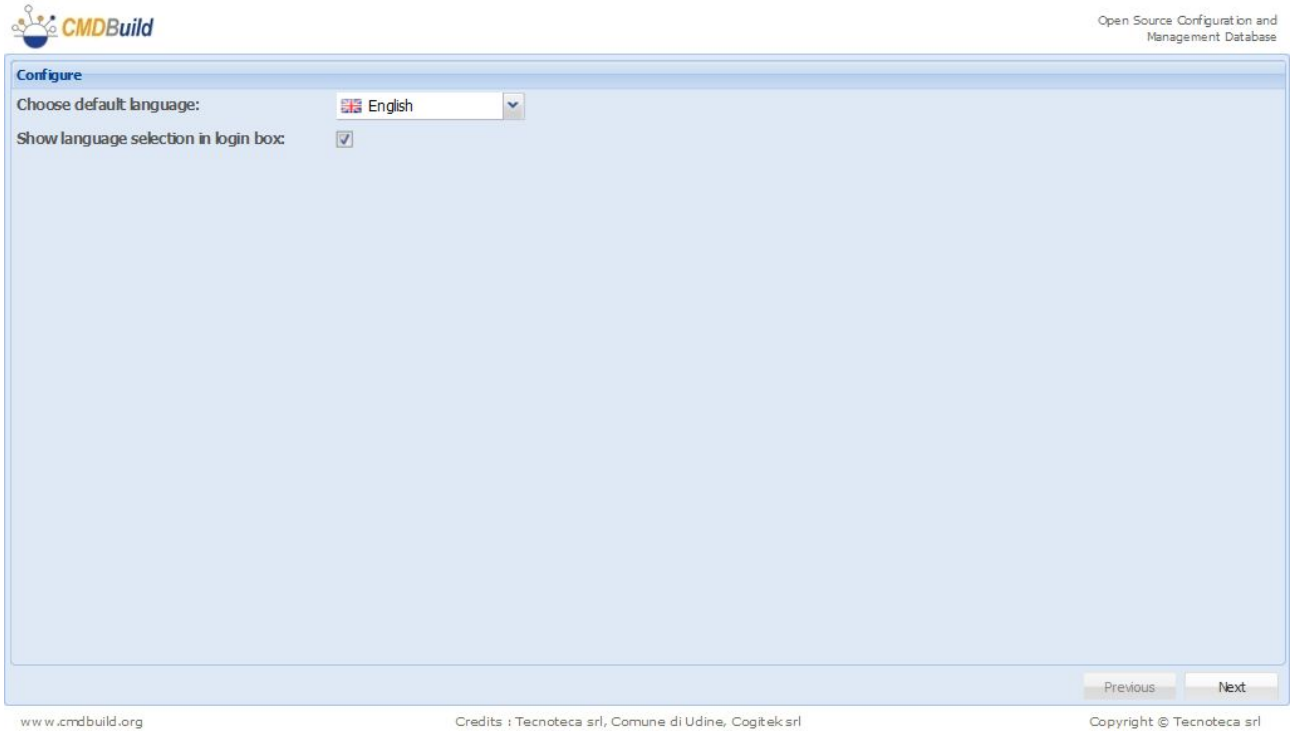
CMDBuild configuration

The basic configuration is done using a few setup pages that CMDBuild presents automatically at the first use.

To access the setup pages you need only to connect using your browser at http://localhost:8080/cmdbuild (the address may vary depending on the Tomcat configuration).

1) Language configuration

If the operations described above were carried out correctly it will appear the following screenshot:



From this screen you can set the language of the system.

Checking the "Show language choice in the login window" a language selection menu will be presented in the login CMDBuild interface.

Once you made your selection, click Next.

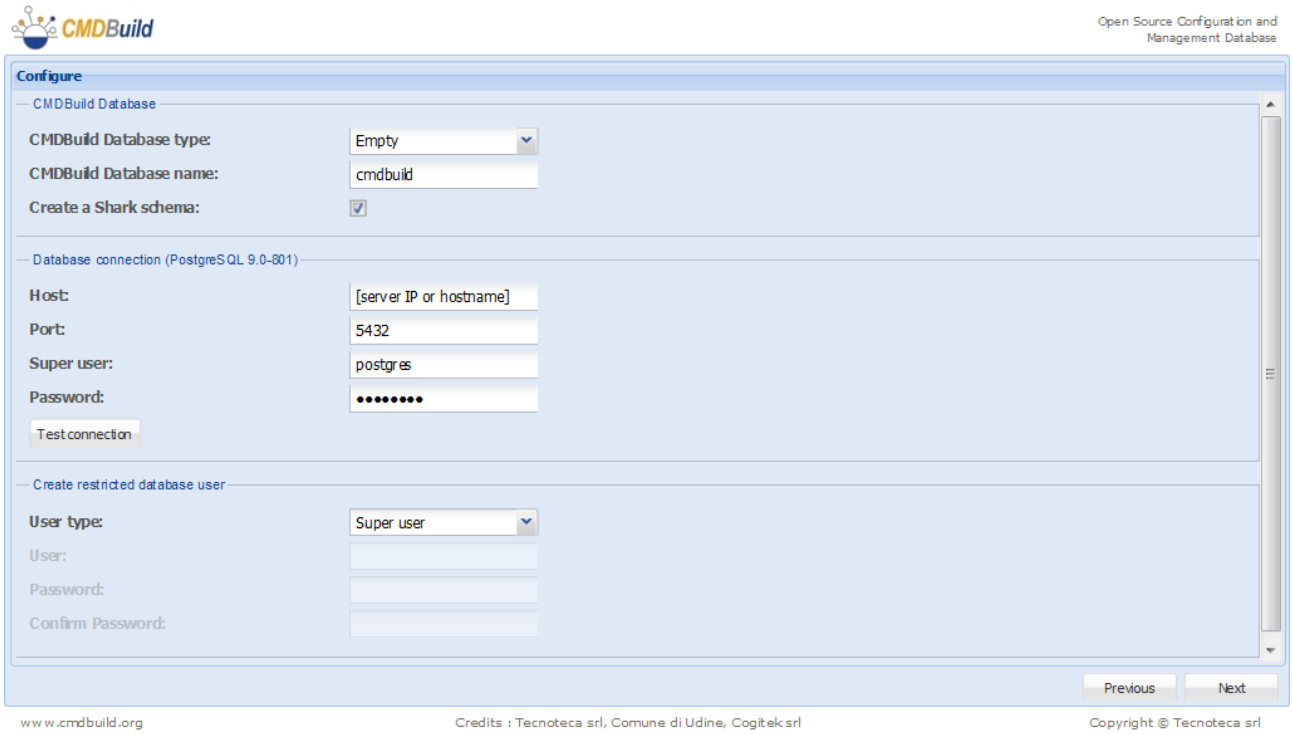
2) Database configuration

In the "Database connection" section you must specify:

- the host (host name or IP address) that is hosting the PostgreSQL database
- the PostgreSQL database port (the default port is 5432)
- the username for accessing the database PostgreSQL (for DBA activities)
- the password to access the PostgreSQL database (for DBA activities)

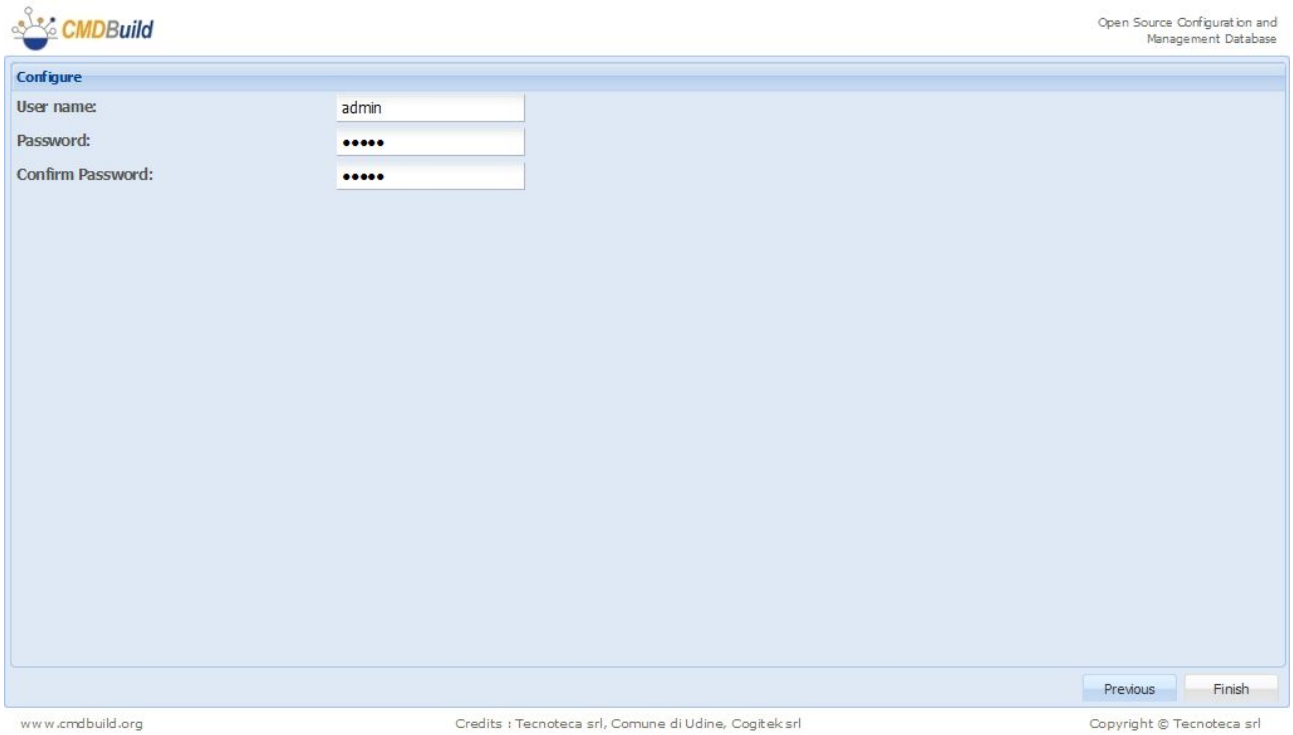
In the CMDBuild Database section you must specify:

- the type of database to configure CMDBuild, choosing from:
  - creating an empty database
  - selecting an existing database compatible with CMDBuild 1.0
  - creation of a database with test data
- the database name



Selecting the checkbox "Create database user with limited privileges," a new PostgreSQL user is created with all privileges as DBA, but only on the database that will be used / created in the CMDBuild instance that you are currently configuring.

3) Configuring access credentials



From the previous form you can specify the credentials that the user administrator (superuser) will use to access CMDBuild (either to the Management Module and to the Administration Module). Clicking on "Finish" you will be redirected to the interface system login.

Installing CMDBuild in manual mode

Getting started

Before you begin installing CMDBuild you need to unpack the directory cmdbuild.war.

To do this you must copy it in the webapps directory of Tomcat and wait for the application server to create the cmdbuild directory.

Verify that in Tomcat is present the JDBC PostgreSQL driver.[1]

Note: In the following we denote:

- **{CMDBUILD}**, the CMDBuild directory in webapps
- **{ALFRESCO}**, the alfresco directory in webapps
- **{SHARK}**, the Together Workflow Server in webapps

Warning: The following installation involves the creation of a new empty database. If you have a pre-existing database go directly to "Configure the application" paragraph. If you already have a CMDBuild database please move to "Application configuration" section.

Database installation

To manually install you must do the following:

- using a tool with a graphical interface (for example pgAdmin3, a native PostgreSQL) or from the command line, create the database by specifying a name, for example cmdbuild:

```
CREATE DATABASE cmdbuild
    WITH OWNER cmdbuilduser
    ENCODING = 'UTF8';
```

- access the new database cmdbuild and create the language plpgsql:

```
CREATE LANGUAGE plpgsql;
```

- to create a database with demo data, run the scripts in alphabetical order that you will find in the directory **{CMDBUILD}/WEB-INF/sql/base_schema**, or alternatively the file

```
{CMDBUILD}/WEB-INF/sql/sample_schemas/demo_schema.sql
```

- if you choose the empty database, you have to check the last available patch analysing the directory

  {CMDBUILD}/WEB-INF/patches

  recording the name of the last patch (in alphabetical order and without extension, for example “1.2.3”) and execute these orders:

```
SELECT cm_create_class('Patch', 'Class', 'DESCR: |MODE: reserved|STATUS: active|SUPERCLASS: false|
TYPE: class');
INSERT INTO "Patch" ("Code") VALUES ('1.2.3');
```

1 The JDBC driver can be downloaded at http://jdbc.postgresql.org/download.html

- execute the following SQL commands to create the "Superuser" user (in the example with username **admin** and password **admin**):

```
INSERT INTO "User" ("Status", "Username", "IdClass", "Password", "Description")
    VALUES ('A', 'admin', '"User"', 'DQdKW32Mlms=', 'Administrator');
INSERT INTO "Role" ("Status", "IdClass", "Administrator", "Code", "Description")
    VALUES ('A', '"Role"', true, 'SuperUser', 'SuperUser');
INSERT INTO "Map_UserRole" ("Status", "IdClass1", "IdClass2", "IdObj1", "IdObj2", "IdDomain")
    VALUES ('A', '"User"'::regclass,'"Role"'::regclass, currval('class_seq')-2, currval('class_seq'), '"Map_UserRole"'::regclass);
```

At this point you have an empty database compatible with CMDBuild system.

Application configuration

You can manually configure the application modifying certain files. If Tomcat was already started, you need to stop it and enter the directory **{CMDBUILD}/WEB-INF/conf**.

Then you have to do the following:

- file **cmdbuild.conf**
  - open it with a text editor
  - select the default language of the system by modifying the "language" (the values are IT and EN; selecting "true" the choice "languageprompt" in the login interface you can select the language)
  - save and close the file
- file **database.conf**
  - uncomment the three rows
  - indicate in "**db.url**" the name of the database to use
  - under "**db.username**" and "**db.password**" indicate the credentials to access the database
  - save and close the file

The installation is now finished, restart Tomcat and login into CMDBuild.

Configuration of the interface between CMDBuild and Together Workflow Server

You can configure Together Workflow Server modifying certain configuration files. If Tomcat was already started, you need to stop it and execute the following operations:

1) Database address

In the file of the shark webapp **{SHARK}/META-INF/context.xml** you have to configure the address of the database modifying the following row:

```
url="jdbc:postgresql://localhost/${cmdbuild}"
```

For example, if the database has the default name "**cmdbuild**" and the postgres server is working on the same computer and uses the standard port, you will have:

```
url="jdbc:postgresql://localhost/cmdbuild"
```

2) URL and service user

In the file **{SHARK}/conf/Shark.conf** set correctly the following parameters:

```
# CMDBUild connection settings
org.cmdbuild.ws.url=http://localhost:8080/cmdbuild/
org.cmdbuild.ws.username=workflow
org.cmdbuild.ws.password=changeme
```

inserting the url related to CMDBuild, the username and the password of the ServiceUser used by Together Workflow Server to use the CMDbuild services.

3) Setup of ServiceUser for Together Workflow Server

With a text editor, open the file **{CMDBUILD}/WEB-INF/conf/auth.conf** and uncomment the row

```
serviceusers.prigileged=workflow
```

If necessary, replacing to workflow the value inserted for the parameter org.cmdbuild.ws.username at the previous point.

4) Starting the services

The Shark Tomcat service must already be active when you start operating across the workflows using the CMDBuild user interface.

On Windows the path should be expressed with the bar reversed:

```
DatabaseManager.ConfigurationDir=C:/srv/tomcat/webapps/shark/conf/dods
```

or double

```
DatabaseManager.ConfigurationDir=C:\\srv\\tomcat\\webapps\\shark\\conf\\dods
```

One way to check the proper functioning of the Shark instance is to refer to the its log file.

Before pursuing the configuration, reboot the tomcat instance of CMDBuild.

5) Authorizations

From the Administration Module CMDBuild you have to enter the Setup menu and:

- enable the workflow (with the appropriate checkbox)
- set the URL to which the Shark service responds

6) Create the user in CMDBuild and grant privileges

From the Administration Module CMDBuild you have to enter the “Users and Groups” and:

- under the heading Users, create a new user with user Name and Password corresponding to the values defined with the parameters org.cmdbuild.ws.username and org.cmdbuild.ws.password in item 2)
- Add the user just created to a Group with administrative privileges (e.g. SuperUsers or a group with the admin flag enabled)

Update of CMDBuild and Together Workflow Server

CMDBuild update

To update CMDBuild please follow the following steps:

1. turn off tomcat and backup the application and its database
2. save the configurations existing in ${tomcat_home_cmdbuild}/webapps/${cmdbuild_instance}/WEB-INF/conf
3. if you use the gis, you have to save the gis icons existing in: ${tomcat_home_cmdbuild}/webapps/cmdbuild_instance}/upload/images/gis
4. delete the directory ${tomcat_home_cmdbuild}/webapps/${cmdbuild_instance}/
5. delete the content of the work directory in Tomcat
6. move the supplied war into the webapps directory of Tomcat and run Tomcat
7. connect CMDBuild and start the configuration specifying that the database already exists and what its name is
8. if necessary, when the configuration ends, the system will warn that the database has to be updated and, clicking on "confirm", the system will run the necessary patches.
9. when connecting with the updated CMDBuild, you are suggested to clean the browser cache

Carried out these operations, the system will be updated. You are suggested to re-activate other configurations (e.g. alfresco, workflow). We suggest you to copy the configuration files previously saved.

It is not possible to overwrite files between this and the previous versions due to the variation in the number of parameters contained.

You have to restart Tomcat every time you edit manually the configuration (from filesystem).

Update of Together Workflow Server

To update Shark please follow the following steps. The Shark version (Together Workflow Server) and the CMDBuild one should always be coherent.

1. switch off Tomcat
2. save the files ${tomcat_home_shark}/webapps/${shark_instance}/META-INF/context.xml and ${tomcat_home_shark}/webapps/${shark_instance}/conf/Shark.conf
3. remove the directory {tomcat_home_shark}/webapps/${shark_instance}/
4. delete the content of the work directory in Tomcat
5. copy the new war into the Tomcat webapps
6. start Tomcat and - when the server starts (check in the file catalina.out) - switch it off so that the shark directory is created in the webapps directory. Otherwise you can simply create a shark directory and decompress in it the war like any other file.
7. once tomcat is stopped, edit ${tomcat_home_shark}/webapps/${shark_instance}/META-INF/context.xml e ${tomcat_home_shark}/webapps/${shark_instance}/conf/Shark.conf so

that they refer to the cmdbuild database and to the application, respectively (you can copy the values from previously saved files without overwriting them)

8. if it exists, remove the file ${tomcat_home_shark}/conf/Catalina/localhost/{nome_istanza_shark}.xml
9. restart Tomcat

We suggest you to remove or rename the extension of any war file existing in the webapps directory of tomcat once the installation is completed.

Run this operation when Tomcat is switched off.

Configuration to access a DMS through CMIS

CMDBuild allows the integration with other document systems through CMIS, an open standard to exchange document data through the web protocol, supported by all main products of the sector (e.g. Liferay, Sharepoint)

The acceptance of such protocol allows to support base features:

- to attach any kind of file to any data card
- to assign a category to every file
- to display the uploaded files

N.B.: the management of categories and metadata is supported at the moment only for the protocol CMIS 1.1. SO, you will be able to use even DMS based on CMIS 1.0, but you will not be able to save information into CMDBuild, such as the author, the description and the category.

Configuration for the categories management

In this section you will learn how to configure a DMS that can support the protocol CMIS 1.1, in order to manage the classification of attachments when updating and refreshing them on CMDBuild cards.

The management of categories is completely included in the aspect files, defined according to the requirements of the DMS.

Here's a sample of configuring Alfresco 5.0.

N.B.: the names used here are unbinding.

1) cmdbuild-model-context.xml

This file includes the definition of the aspect for saving the categories and it is structured as follows:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE beans PUBLIC '-//SPRING//DTD BEAN//EN' 'http://www.springframework.org/dtd/spring-
beans.dtd'>
<beans>
   <!-- Registration of new models -->
   <bean id="example.dictionaryBootstrap" parent="dictionaryModelBootstrap" depends-on="dictionaryBootstrap">
      <property name="models">
         <list>
            <value>alfresco/extension/cmdbuildModel.xml</value>
         </list>
      </property>
   </bean>
</beans>
```

2) cmdbuildModel.xml

This file defines the usable categories

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- The important part here is the name - Note: the use of the my: namespace which is defined further on in the document -->
<model name="cmdbuild:module" xmlns="http://www.alfresco.org/model/dictionary/1.0">

	<description>Custom Model for CMDBuild</description>
	<author>CMDBuild Team</author>
	<version>1.0</version>

	<imports>
		<!-- Import Alfresco Dictionary Definitions -->
		<import uri="http://www.alfresco.org/model/dictionary/1.0" prefix="d" />
		<!-- Import Alfresco Content Domain Model Definitions -->
		<import uri="http://www.alfresco.org/model/content/1.0" prefix="cm" />
	</imports>

	<!-- Introduction of new namespaces defined by this model -->
	<namespaces>
		<namespace uri="org.cmdbuild.dms.alfresco" prefix="cmdbuild" />
	</namespaces>

	<aspects>
		<aspect name="cmdbuild:classifiable">
			<title>Classification</title>
			<properties>
				<property name="cmdbuild:classification">
					<type>d:text</type>
				</property>
			</properties>
		</aspect>
	</aspects>

</model>
```

Both files should be copied into the directory of the used DMS, for instance for the DMS Alfresco:

${ALFRESCO_HOME}/tomcat/shared/classes/alfresco/extension

As for Alfresco, the system should be restarted so that the changes can work.

Configuration for the metadata management

In this section you will learn how to configure the management of custom metadata in order to manage additional information when updating and refreshing attachments on CMDBuild cards.

Even in this case, the aspects have to be defined according to the requirements of the DMS in use. Here's a sample related to Alfresco.

1) Custom model definition

As provided for in Alfresco, you have to create a custom model where you will define metadata and their groups.

You can find a sample of that in the CMDBuild files (dms/alfresco module).

The default name for this file is cmdbuildCustomModel.xml, but you can customize it specifying the new name in the configuration file dms.conf of CMDBuild:

```
alfresco.custom.model.filename=anotherNameForCustomModel.xml
```

Sample of the file:

```
<?xml version="1.0" encoding="UTF-8"?>
<model
    name="cmdbuild:customModule"
    xmlns="http://www.alfresco.org/model/dictionary/1.0">
    ...
    <namespaces>
        <namespace uri="org.cmdbuild.dms.alfresco" prefix="cmdbuild" />
    </namespaces>
    <constraints>
        <constraint name="cmdbuild:imageFormat" type="LIST">
            <parameter name="allowedValues">
                <list>
                    <value>bmp</value>
                    ...
                </list>
            </parameter>
        </constraint>
        ...
```

```
    </constraints>
    <types>
        <type name="cmdbuild:Document">
            <title>Document</title>
            <mandatory-aspects>
                <aspect>cmdbuild:documentStatistics</aspect>
                ...
            </mandatory-aspects>
        </type>
        ...
    </types>
    <aspects>
        <aspect name="cmdbuild:documentStatistics">
            <title>Document Statistics</title>
            <properties>
                <property name="cmdbuild:characters">
                    <title>Number of characters</title>
                    <type>d:long</type>
                </property>
                ...
            </properties>
        </aspect>
        ...
    </aspects>
</model>
```

Namespace definition

You have to define a namespace as follows:

```
<namespaces>
    <namespace uri="org.cmdbuild.dms.alfresco" prefix="cmdbuild" />
</namespaces>
```

The above mentioned values are defaults, but you can customize them specifying the following headings in the CMDBuild configuration file dms.conf:

```
alfresco.custom.uri=another.unique.uri

alfresco.custom.prefix=another_prefix
```

Please, pay attention to elements of the .xml document: most of their names are preceded by the prefix here defined.

List of values definition

For certain aspect features (see "Aspect definition" in the next page), you can specify predefined values that can be selected using a combobox.

```
<constraints>
    <constraint name="cmdbuild:imageFormat" type="LIST">
        <parameter name="allowedValues">
            <list>
                <value>bmp</value>
                ...
            </list>
        </parameter>
    </constraint>
    ...
</constraints>
```

Please note: the name attribute of the constraint element must match with the name attribute of the property element defined within the aspect (see "Aspect definition" in the next page).

Types definition

The types below are combined with categories defined and managed on the CMDBuild's side.

```
<types>
    <type name="cmdbuild:Document">
        <title>Document</title>
        <mandatory-aspects>
            <aspect>cmdbuild:documentStatistics</aspect>
            ...
        </mandatory-aspects>
    </type>
```

```
    ...
</types>
```

In particular:

- the definition of a type is necessary only if you have to match aspects (and therefore also metadata) to documents managed with CMDBuild/Alfresco
- the content of the title element must match with the Description attribute of a CMDBuild Lookup (e.g. “Document”).

Aspect definition

Aspects group together several features, and different aspects can be applied to one single document.

```
<aspects>
    <aspect name="cmdbuild:documentStatistics">
        <title>Document Statistics</title>
        <properties>
            <property name="cmdbuild:characters">
                <title>Number of characters</title>
                <type>d:long</type>
            </property>
            ...
        </properties>
    </aspect>
    ...
</aspects>
```

In particular:

- if you want to to apply aspects, they must match with a document type (see "Types definition" in the next page)
- features types that are currently managed are as follows: “text”, “int”, “long”, “float”, “date”, “datetime”, “boolean”, as defined by Alfresco
- you can manage the list of “text” values (see "List of values definition" in the previous page).

2) Definition of autocompletion rules

You have to create an .xml file that will be interpreted by CMDBuild. Its information must allow the metadata autocompletion, starting from the attributes of the card the document is attached to.

The default name for this file is metadataAutocompletion.xml, but you can customize it specifying the new name in the configuration file dms.conf of CMDBuild:

```
metadata.autocompletion.filename=anotherNameForAutocompletionRules.xml
```

Here is a sample:

```
<?xml version="1.0" encoding="UTF-8"?>
<autocompletion>
    <metadataGroups>
        <metadataGroup name="documentStatistics">
            <metadata name="character">
                <rules>
                    <rule classname="Employee" value="123" />
                    <rule classname="Asset" value="client:..." />
                </rules>
            </metadata>
            ...
        </metadataGroup>
    </metadataGroups>
</autocompletion>
```

In particular:

- the file can be omitted if no autocompletion is required
- the name attribute of the metadataGroup element represents the name (without the prefix) of an aspect (see "Aspect definition" in the previous pages)
- the attribute name of the metadata element represents the name (without the prefix) of an aspect feature (see Aspect definition)
- every rule element shows the way to complete the metadatum value, this value can be a fix value or an expression managed with the Template Resolver (see the specific documents).

3) Installation

Given the following files:

- cmdbuildCustomModel.xml
- metadataAutocompletion.xml

Within the directory of the Alfresco extensions (e.g. $ALFRESCO_HOME/tomcat/shared/classes/alfresco/extension):

- copy the file cmdbuildCustomModel.xml
- create a file cmdbuild-custom-model-context.xml built in this way (remember to specify the file name of the right custom module)

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE beans PUBLIC '-//SPRING//DTD BEAN//EN' 'http://www.springframework.org/dtd/spring-beans.dtd'>
<beans>
    <!-- Registration of new models -->
    <bean
        id="extension.dictionaryBootstrap"
        parent="dictionaryModelBootstrap"
        depends-on="dictionaryBootstrap">
        <property name="models">
            <list>
                <value>
                    alfresco/extension/cmdbuildCustomModel.xml
                </value>
            </list>
        </property>
    </bean>
</beans>
```

- create a file web-client-config-custom.xml containing definitions that allow to view aspect features within the web interface of Alfresco (see official documents)
- start/reboot Alfresco so that changes have effects, check in the log any error definition of the module
- Within the CMDBuild configuration directory:
- copy the file cmdbuildCustomModel.xml
- copy the file metadataAutocompletion.xml
- clear the server cache and then the client one

N.B.: the file names are unbinding.

Configuration of the interface between CMDBuild and Alfresco

1) Configuring the CMDBuild side

In order to configure the use of a DMS through CMIS with CMDBuild, enter the directory {CMDBUILD}/WEB-INF/conf and open the file **dms.conf** with a text editor.

It is then necessary to perform the following operations:

- set **enabled**=true

- set **dms.service.type**=cmis:
- verify that in "**dms.service.cmis.url**" there is the url to reach the dms through cmis
- set the authentication credentials to "**dms.service.cmis.user**" "**dms.service.cmis.password**"
- set the filesystem path to "**dms.service.cmis.path**"
- set the model name to "**dms.service.cmis.model**"
- specify the used CMDBuild category to "**category.lookup**"

2) Alfresco side configuration

Refer to the chapter "Installing and using the system DMS Alfresco 3.4"

Installing and using the system DMS Alfresco 3.4

If you do not have a DMS supporting the protocol CMIS 1.1, you can upload the attachments and their metadata and categories through FTP services and webservices offered by Alfresco 3.4.

Here you can learn how to install Alfresco 3.4 and how to upload and download files, in order to classify them and to upload any custom metadata.

In Alfresco there is a wizard, both graphic and text, used to simplify installation and configuration of basic settings.

In particular, you have to specify the following:

- ports used by Tomcat
- port used by the FTP server
- settings related to the database

If you decide to proceed using the manual installation, you can refer to the following installation notes, related to the installation of Alfresco 3.4.

Note: in the following we denoted **{ALFRESCO}** the alfresco installation directory (for example /opt/alfresco).

Basic configuration

1) Configuring Tomcat

Configure the start of the Tomcat server used by Alfresco on ports different from the Tomcat server used by CMDBuild.

Go to the directory

```
{ALFRESCO}/tomcat/conf
```

Open to edit the **server.xml** file and change the properties:

```
<Server port="8005" shutdown="SHUTDOWN">
...
<Connector port="8080" protocol="HTTP/1.1" ... />
...
<Connector port="8009" protocol="AJP/1.3" ... />
```

so as not to conflict with the instance of Tomcat used by CMDBuild.

For example you can use the following (always taking care to other instances of Tomcat in your system):

```
<Server port="9005" shutdown="SHUTDOWN">
...
<Connector port="9080" protocol="HTTP/1.1" ... />
```

...

<Connector port="9009" protocol="AJP/1.3" ... />

2) Configuring the repository and the database

Go to the directory **{ALFRESCO}/tomcat/shared/classes/** and edit the file **alfresco-global.properties** (if not present you may use a copy of the file **alfresco-global.properties.sample**).

Change the following properties:

dir.root={REPOSITORY DIRECTORY}

to define the location of the repository where documents are stored (for example: C:/alfresco/repository for Windows systems or /var/alfresco/repository per sistemi *nix).

Alfresco supports the following databases: HSQL (default), MySQL, Oracle, Sybase, SQLServer, and PostgreSQL. We suggest addressing Alfresco to a new database managed by the same PostgreSQL server used for CMDBuild. Since during installation it is not possible to make this choice, you have to do it manually.

Set the following properties files in the same file **alfresco-global.properties**:

db.driver=org.postgresql.Driver

db.username=postgres

db.password=postgres

db.url=jdbc:postgresql://localhost:5432/alfresco

Copy the drivers to access the Postgres database (interface library in the directory /extras/tomcat-libs/[TomcatVersion] of the CMDBuild distribution) into **{ALFRESCO}/tomcat/lib**.

Using a GUI tool (for example pgAdmin3 of PostgreSQL) or from the command line to create the database using the name specified in the property "**db.url**" above referred (in the "alfresco" example).

Also in the file **alfresco-global.properties**, edit the following lines to enable the FTP server:

ftp.enabled=true

ftp.port=1121

ftp.ipv6.enabled=false

Be careful if there is another FTP server on the same host, in this case you have to change the port that the FTP server will run Alfresco.

The specified port must be equal to that specified in the file:

{CMDBUILD}\WEB-INF\conf\dms.conf

At the end, once the initial database creation is successful, stop the Alfresco and check if in the file **alfresco-global.properties** the property "**db.schema.update**" is set to *false* (or possibly commented out):

db.schema.update=false

Additional configurations

Then you have to create in Alfresco the "space" intended to contain CMDBuild attachments and the category associated with these documents.

At the end, you have to start Alfresco and, if there are no problems, after a few minutes you can log in to Alfresco via web browser at:

http://{SERVER}:{PORT}/alfresco

The default credentials are "admin" both as the user name and the password.

In the "Navigator" go to "Company Home", then "User Homes" and create a space with the same name as that specified in the file:

{CMDBUILD}\WEB-INF\conf\dms.conf (for example "cmdbuild")

Go now in the administration console of Alfresco (first icon on the upper bar), go to "Category Management" and create a new category with the same name and description as specified in the file:

{CMDBUILD}\WEB-INF\conf\dms.conf (for example "AlfrescoCategory")

At this point the configuration is complete. When you restart Tomcat, you can also handle attachments CMDBuild.

Configuration for the metadata management

See the similar section in the chapter “Configuration to access a DMS through CMIS ”

Configuration of the interface between CMDBuild and Alfresco

1) Configuring the CMDBuild side

To configure the use of Alfresco Community 3.4 with CMDBuild enter the directory {CMDBUILD}/ WEB-INF/conf and open with a text editor the file **dms.conf** (**legacydms.conf** for CMDBuild versions prior to 1.4).

It is then necessary to perform the following operations:

- set **enabled**=true
- set **dms.service.type**=cmis:
- verify that in "**server.url**" there is the url to reach Alfresco through webservices
- edit fileserver.port inserting the port number on which the Alfresco FTP service is running (see next paragraph)
- edit the entry "**repository.fspath**" with the path of the space of CMDBuild on Alfresco, similar to the "**repository.app**"
- specify the CMDBuild category used in Alfresco to "**category.lookup**"

2) Alfresco side configuration

Refer to the chapter "Installing and using the system DMS Alfresco 3.4"

Backup of CMDBuild data

To backup data in CMDBuild you just have to backup the PostgreSQL database.

If you do not want to go on through graphical tools, like PgAdmin III, you can work on command lines, specifying, for example:

```
pg_dump --host localhost --port 5432 --username "postgres" --format custom --verbose --file /backup/cmdbuild/cmdbuild.backup cmdbuild_db
```

Such command will backup the db called “cmdbuild_db”, on the local listen server in the 5432 port using the postgres username. The format will be pressed (--format custom) and written into the directory /backup/cmdbuild/cmdbuild.backup. The command will also print the details of what it is doing per video (--verbose).

To get more detailed explanations about the syntax of pg_dump, please read the records relating to postgres (with Linux you can digit “man pg_dump” in order to have an excellent description of this command).

An example file to backup databases:

```
#!/bin/sh
# Executes a backup of the specified postgresql databases
#
# Steps:
# - dumps (compressed) the database into the target directory
# - archive (tgz) the backup
# - removes the backup

PG_DUMP="pg_dump"
PG_DUMP_OPTS="-h localhost -p 5432 -U postgres -F c --column-inserts"

TIMESTAMP=`date +%Y%m%d%H%M%S`

BACKUP_ROOT="$1"

while [ $# -gt 1 ]
do
        shift
        DB_NAME="$1"

        cd "${BACKUP_ROOT}"

        DB_BACKUP="${DB_NAME}-${TIMESTAMP}.backup"
        DB_ARCHIVE="${DB_BACKUP}.tar.gz"
```

```
        ${PG_DUMP} ${PG_DUMP_OPTS} -f "${DB_BACKUP}" "${DB_NAME}"

        if [ ! -r "${DB_BACKUP}" ];
        then
                logger "File '${DB_BACKUP}' not found"
        else
                tar -czf "${DB_ARCHIVE}" "${DB_BACKUP}"
                logger "Database '${DB_NAME}' has been successfully backed up"
                rm "${DB_BACKUP}"
        fi
done
```

Backup of Alfresco data

To backup data in Alfresco you just have to backup the PostgreSQL database (hereafter we will assume that it is in the database PostgreSQL).

As for CMDBuild, the command to backup the database on command line is as follows:

```
pg_dump --host localhost --port 5432 --username "postgres" --format custom --verbose --file /backup/alfresco/alfresco.backup alfresco_db
```

It is also necessary backup - as zip or tar.gz file - also the whole directory containing the repository where files are saved (e.g.: C:/alfresco/repository for Windows systems or /var/alfresco/repository for *nix systems).

```
#!/bin/sh
ALFRESCO_SERVICE="/etc/init.d/alfresco"

TT_PG_BACKUP="/usr/local/bin/tt_pg_backup.sh"

BACKUP_HOME="/backup"
ALFRESCO_BACKUP_HOME="${BACKUP_HOME}/alfresco"
BACKUP_LOG="/var/log/cmdbuild/crontab-backup.log 2>&1"

ALFRESCO_DATABASE="alfresco"

ALFRESCO_DATA="/var/lib/alfresco/data"
TIMESTAMP=`date +%Y%m%d%H%M%S`
ALFRESCO_DATA_BACKUP="alfresco-data-${TIMESTAMP}.tar.gz"

logger "Stopping Alfresco service"
${ALFRESCO_SERVICE} stop
sleep 5

logger "Backupping Alfresco database"
${TT_PG_BACKUP} "${ALFRESCO_BACKUP_HOME}" "${ALFRESCO_DATABASE}" > ${BACKUP_LOG}

logger "Backupping Alfresco data"
tar czf "${ALFRESCO_BACKUP_HOME}/${ALFRESCO_DATA_BACKUP}" ${ALFRESCO_DATA}
logger "Alfresco database/data have been backed up."

logger "Starting Alfresco service"
```

```
${ALFRESCO_SERVICE} start
```

Backup schedule

To schedule a periodical data backup on Linux systems, we suggest you to go on as follows:

1. create a /etc/cron.d/backup file
2. insert the commands into the backup file, with the proper cron syntax.

For a more efficient cleaning we recommend you to create various single scripts, one for each system, and call them in this file rather than write the commands directly into the cron file itself.

An example:

```
00 19 * * *     root   /usr/local/bin/tt_pg_backup.sh /backup/cmdbuild cmdbuild
&>> /var/log/cmdbuild/backup/backup.log

10   19   *   *   7            root      /usr/local/bin/alfresco-backup.sh   &>>
/var/log/cmdbuild/backup/backup.log
```

Authentication modes

Introduction

Thanks to proper configurations of CMDBuild, you can delegate the authentication control to access external services.

This possibility concerns the control of the account (username and password). Profiles and permissions are still managed within the CMDBuild group to which the user belongs.

The parameters to configure the behavior of CMDBuild when authenticating are indicated in the file **auth.conf** in the directory WEB-INF of the CMDBuild webapps within Tomcat.

From this file is possible.

The file is divided into 4 sections:

- configuring the type of authentication
- configuring header authentication
- configuring LDAP authentication
- Single sign on configuration through CAS

Configuring authentication type

Below are the parameters to be set during configuration and their meaning is specified.

1) auth.methods

With this parameter you can define the authentication "chain" of CMDBuild.

It is possible, i.e., to define in cascade which types of authentication you can use to allow access to the user and set the priority.

Example

```
auth.methods=LdapAuthenticator,DBAuthenticator
```

The configuration in the previous example indicates that whenever a user logs into the system, CMDBuild must first verify the credentials via LDAP, and if they fail, via the data base in CMDBuild.

The accepted parameters are:

- HeaderAuthenticator (authentication via header control)
- LdapAuthenticator (authentication via credential verification on LDAP)
- CasAuthenticator (single sign through CAS)
- DBAuthenticator (standard authentication)

2) serviceusers

With this parameter you can define the “service” users of CMDBuild. This kind of privileged users is planned for the exclusive use of external systems such as, for example, Liferay portlet. So, the interface login for that kind of users will be disabled.

3) force.ws.password.digest

If this parameter is set to "true", it forces the use of specific Username Token with password digest for authenticating using webservice.

Setting that parameter to "false" you can also use plain text passwords for authentication via Username Token. This can be useful combined with the use of LDAP for access control within CMDBuild.

Configuring Header Authentication

From this section you can configure the authentication using the header verification.

To do this simply edit the file **header.attribute.name** and specify the name of the attribute present in the HTTP header to be used to authenticate the user on CMDBuild.

This type of authentication requires the presence of an upstream authentication system, which takes care of generating the specific headers that can then be used to authorize access to CMDBuild.

Configuring LDAP authentication

This section documents how to configure authentication within CMDBuild via LDAP.

CMDBuild currently supports only authentication "simple bind". However, you can use the '"anonymous bind" for the user search in the LDAP tree.

In order to manage the user permissions within CMDBuild is necessary that users that have to access to CMDBuild they are also present within the webapp.

For example, if a user with LDAP UID j.doe needs accessing CMDBuild as a user of the "technicians" group, you have to perform these steps:

- user creation in j.doe CMDBuild with a default password (not necessarily that of LDAP)
- creation of the Technical Group and definition of the relevant permits
- adding user to group j.doe Technicians

At this point, when you authenticate j.doe, his credentials will be verified (using the authentication chain defined in auth.methods) against the LDAP tree.

Below there is a description of the configuration parameters.

1) ldap.server.address

This attribute is used to specify the address to which you can reach the LDAP server.

Example:

```
ldap.server.address=localhost
```

2) ldap.server.port

This attribute is used to specify the port the LDAP server. The default is 389.

Example:

```
ldap.server.port=389
```

3) ldap.use.ssl

It specifies whether to use an encrypted connection to the LDAP server. The default is disabled.

Example:

```
ldap.use.ssl=true
```

4) ldap.basedn

This attribute indicates the Base DN that will be used to query the LDAP tree.

Example:

```
ldap.basedn=dc=example,dc=com
```

5) ldap.bind.attribute

This attribute indicates the attribute will be run on the bind user.

For example, as an attribute for specifying bind dn uid and considering the basis indicated above, the LDAP query that will be generated uid = username, dc = example, dc = com.

Example:

```
ldap.bind.attribute=uid
```

6) ldap.search.filter

You can specify with this attribute, a search filter to be used for research.

Single sign on configuration through CAS

This section documents how to configure the single sign on (SSO) within CMDBuild via CAS.

The authentication runs as follows:

- the user asks for the CMDBuild url
- the CAS authenticator sends the request to the CAS server (${cas.server.url} + ${cas.login.page}) specifying the CMDBuild access url (in the ${cas.service.param} paramter)
- the CAS server answers with a ticket (${cas.ticket.param} paramter) which you can extract the username from
- if the username is properly validated/extracted, then CMDBuild gets on with the login

In order to manage the user permissions within CMDBuild is necessary that users that have to access to CMDBuild they are also present within the webapp.

Below there is a description of the configuration parameters.

1) cas.server.url

This attribute is used to specify the address to which you can reach the CAS server.

Example:

```
cas.server.url=https://cas.mycompany.com/cas
```

2) cas.login.page

This attribute is used to specify the CAS login page

Example:

cas.login.page=/login

3) cas.service.param

Name of the parameter that CAS uses to specify the url of the authentication service.

Example:

cas.service.param=service

4) cas.ticket.param

Name of the parameter that CAS uses to specify the ticket reference that it generates until the authentication is valid.

Example:

cas.ticket.param=ticket

Access to CMDBuild resources via URL

Introduction

CMDBuild allows you to set oneself via URL on your resources.

At the moment "classes" resources are being managed, but in the future the mechanism will be extended to manage filters, reports, etc.

In case of previous authentication of the user, the criteria-compatible URL query will cause a new positioning on that asset. Alternatively you will be redirected to the login page and only later you will be moved to the aforementioned asset.

The feature can be useful to insert into CMDBuild notifications the link to the ticket page, in order to allow the management software of a telephone exchange to open a client card, to view an asset card reading a code, etc.

Examples of valid URL

Samples of URL, valid according to the foreseen implementation criteria.

We remember you that placement URLs can be used on a class and on a card according to the value of its key-field (the first three examples).

Further extensions will be gradually activated with next releases.

All URLs are related to the CMDBuild base address. In case of CMDBuild on-line demo instance, the URL #classes/Employee/cards/ should be:

http://demo.cmdbuild.org/cmdbuild/index.jsp#classes/Employee/cards/

Some examples:

#classes/Employee/cards/

display of a class card

#classes/Building/cards/76/

card selection with specific “Id”

#classes/Employee/cards/TelephoneNumber~123456/

selection of a card from a telephone number using the simple filter

#classes/Employee/cards?clientFilter={"attribute":{"simple":{"attribute":"Phone","operator":"equal","value":["76543"]}}}

selection of a card from a telephone number

#classes/Building/cards?clientFilter={"attribute":{"simple":{"attribute":"Description","operator":"contain","value":["Building"]}}}

selection of a card from a general filter

#classes/Building/cards/76/print?format=pdf

card print in PDF format

#classes/Building/print?format=csv

grid print of the class in CSV format

Mobile interface activation

Introduction

CMDBuild "mobile" is an interface that can be used on smartphones and tablets in order to run useful application features during the activities in the field.

It is created with Sencha Touch (Javascript framework developed by Sencha, the same producer of the Ext JS framework used by CMDBuild desktop) and it is able to interact with CMDBuild through the REST webservice.

CMDBuild mobile implements the main features of the desktop interface: multilingual, multigroup login, navigation menu, class management with relations and attachments, researches and filters, workflow management with the most used widgets, report management, usefulness ("app" settings and log management).

Components and Architecture

This APP is created using the following components:

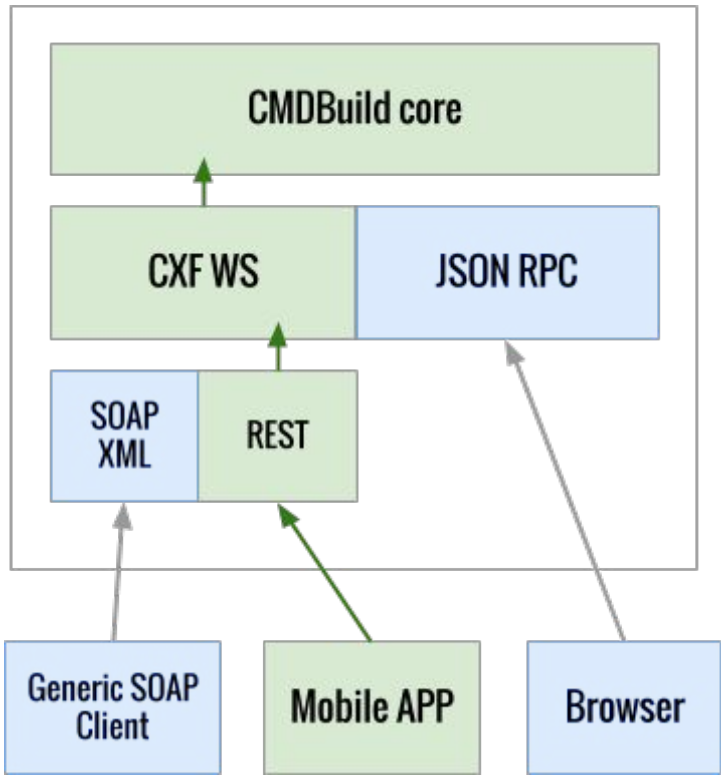
- Sencha Touch
  framework Javascript created by Sencha
- Cordova
  framework mobile cross-platform
- Deft JS
  extension for mobile enterprise applications developed with Sencha Touch
- log4javascript
  javascript logging framework
- Crosswalk (Android)
  tool to deploy the application on the custom webview independent from the Android version
- Siesta
  library for unit tests / integration tests

Server-side the REST web service layer is developed using the CXF framework Apache, already integrated in CDMBuild and used for the SOAP web service layer.

The system retraces the software architecture of the REST web service with the following features:

- features divided in web resources
- addressable resources (URI)
- HTTP standard methods (GET, PUT, POST, DELETE)
- JSON media type
- link for the resource navigation
- stateless

Architecture drawing:



Compatibility

- Android 4.0.3 or more recent
- iOS 6 or more recent

Limitations of use

The mobile interface is made available with non-open source license, which allows only those who have signed with Tecnoteca srl a maintenance service for the CMDBuild application to use the service, and only until that service is active, with a limited additional charge.

GUI Framework activation

Introduction

The GUI Framework makes available a simplified interface to non-technical staff.

The GUI Framework includes the following main features:

- it can be activated in portals based on different technologies as it is developed in javascript / JQuery environment
- it allows an (almost) unlimited freedom when projecting the graphic layout, defined through an XML descriptor and with the possibility of intervening on the CSS
- it grants a quick configuration thanks to predefined functions (communication, authentication logics, etc.) and to native graphic solutions (forms, grids, upload buttons and other widgets)
- it interacts with CMDBuild through the REST webservice
- it is able to gather data from the database of other applications, allowing the management of mix solutions

Configuration

The framework defines HTML pages starting from their definition in XML.

Such pages can be inserted into a HTML file, allowing the framework to insert into existing portals access points to CMDBuild data.

The system configurability can be achieved through the definition of custom CSS and Javascript.

The element that should be inserted into the portal html is a html container (DIV, IFRAME ...) in the following format:

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>
<script type="text/javascript" src="http://10.0.0.107:8080/cmdbuild-gui-framework/api/cmdbuildAP.js"></script>
<script type="text/javascript">
$( document ).ready(function(){
    $("#cmdbuilForm").cmdbuildAP({
        apiUrl : 'http://10.0.0.107:8080/cmdbuild/services/rest/',
        appRootUrl : 'http://10.0.0.107:8080/cmdbuild-gui-framework/api/',
        appConfigUrl : 'http://10.0.0.107:8080/cbap/cbap/config/bologna/',
        cqlUrl : 'http://10.0.0.107:8080/cbap/cbap/cql/',
        customjs : [
            'backend/GUIForm.js',
            'backend/Process.js'
        ],
        start: "home.xml",
        theme: [
            "jquery-ui.theme-crt-toscana.min.css",
```

```
                "custom.css"
            ],
        });
    });
</script>
<div id="cmdbuilForm"></div>
```

The style configuration is designated to CSS, which can directly uploaded from the Html tag. This makes the GUI flexible, allowing the programmer to include it into the portal so that is does not look like an unrelated element.

You can also define the servers that contain the CMDBuild system, the form configuration files and the Cql engine, the native data query language in CMDBuild.

In the tag customjs some GUI customization files are inserted.

The tag uploads the GUI engine configuring it with the above-mentioned files.

The forms are defined in XML through a language that is similar to HTML, with the possibility of defining data management forms directly linked to metadata defined in CMDBuild.

All GUI tags, commands and behaviours can be configured again by the programmer according to the system criteria.

Since the library refers to JQuery, where the GUI Framework was developed, the programmer can natively use every plugin compatible with JQuery, developing the available features.

Another element that marks the GUI operation is the "backend" mechanism, i.e. the Javascript classes that link the GUI to the servers. In this way the application provides further freedom provided by the possibility to define the data format the server is waiting for and the name of the server you want to link to.

Liferay portlet activation

Introduction

Even though it is a solution overreached by the GUI Framework, CMDBuild makes still available JSR186 portlets through which you can export some of its features in compatible intranet portals (the CMDBuild portlet is currently certified with the open source Liferay portal).

JSR168 Portlets are Java web components that can be used as a plug-in within "containers" such as web portals compatible with this standard.

With this mechanism you can configure a custom portal by placing portlets in the pages of interest and sharing some services of the host system including the authentication.

As in the previous paragraph CMDBuild portlets communicate through their webservice.

The CMDBuild portlet makes available some features to non-technical users, because would find difficult to use the application standard interface.

The portlet includes the following functions:

- management of a data sheet (insert, modify, delete)
- initiation and progress of a process
- running a report

The configuration includes two aspects:

- configuration of CMDBuild portlets in Liferay
- configuration of CMDBuild

The portlet CMDBuild JSR 168 includes three different management options:

- access only to users registered as system users of CMDBuild
- access also to users of "guest" type (i.e. not registered as system users of CMDBuild), but included in a class "application" CMDBuild configured (e.g. employees, suppliers, etc.)
- access also to users of "guest" type (ie not registered CMDBuild System), which are not even included in an "application" class of CMDBuild (e.g. citizens, students, etc.)

Installing Liferay portlet

The configuration options are stored in two files:

- portlet.properties (this file is a "template" and should not be changed)
- portlet-ext.properties (overwrites the "template" file, all customizations are to be included in this file)

The two files are stored in the directory:

WEB-INF/classes

within the portlet webapp.

The main parameters are:

- **cmdbuild.url**: URL of the CMDBuild webservice(default http://localhost:8080/cmdbuild/services/soap/Private)
- **cmdbuild.user**: user-id for accessing the webservice (default "portlet")
- **cmdbuild.password**: password for accessing the webservice (default "portlet")
- **cmdbuild.group**: the above user's group membership (default "Guest", you have to create it in CMDBuild)
- **user.class**: the "application" class of CMDBuild containing the list of users who will have access to portlets
- **user.attribute.username**: the attribute containing the username, in the class indicated in the previous step (default "Username")
- **user.attribute.email**: the attribute containing the email address, within the class specified in the preceding paragraph (default Email)
- **auth.method**: the authentication policy, which may take the value "username" or "email" (default "email")

Example:

```
cmdbuild.url=http://localhost:8080/cmdbuild-test/services/soap/Private

user.class=Employee
```

These values override the default properties with a new URL and a new class name.

Warning

For the changes of the configuration parameters to take effect, you have to restart Liferay.

CMDBuild configuration

In the following we will refer with ${CMDBUILD} the directory containing the CMDBuild "webapp" inside Tomcat.

Webservice user

You have to modify the file **${CMDBUILD}/WEB-INF/conf/auth.conf** and change the property **serviceusers** in accordance with the value defined for the property **cmdbuild.user** defined in the **portlet-ext.properties**. For the change to take effect you have to restart CMDBuild.

Then, from within the CMDBuild Administration module, you have to:

- create a new user (as defined in the **cmdbuild.user** property of the **portlet-ext.properties** file)
- create a new group (as defined in the **cmdbuild.group** property of the **portlet-ext.properties** file)
- add the new user to the new group
- set the new group as a group "**default login**" for the new user (required for authentication type "guest")
- create a "custom" menu for the new group (optional)

GeoServer

Introduction

CMDBuild includes the ability to manage the geo-reference of the assets or of other information entities (customers, suppliers, locations, etc.) through visualization on maps and/or floor plans.

The geo-reference on the territory is made by integrating external services such as OpenStreetMap, Google maps, etc., while the management of plans has been implemented by using the GeoServer open source system.

Through the use of GeoServer you can add custom layers (e.g. plans) that you can use in the GIS module.

Supported formats are:

- Shape
- GeoTiff
- WorldImage.

Installing Geoserver

In order to install Geoserver it is necessary to:

- download the application from the official website (http://geoserver.org/)
- deploy the war file in Tomcat
- log-in using the username **admin** and password **geoserver**
- delete all preinstalled workspaces
- create one workspace called, for example, "cmdbuild"

Configuring CMDBuild

Once you select the Administration Module of CMDBuild, you have to access the GIS page of the Setup menu and to enable the GIS module.

In order to do so you access the GIS menu and set the following items:

- **External Services**
  - enable GeoServer
  - specify the parameters related to the installation
  - specify the name of the workspace you created earlier
  - specify the credentials of the administrator
- **Geoserver layers**
  - add the necessary layers, which are then stored in Geoserver
- **Layers order**
  - specify the order of layers as will be presented in the Management Module

BIMServer

General Information

CMDBuild includes the ability to manage the geo-reference of the assets through visualization with BIM features (Building Information Modeling).

With the georeference onto BIM models (exported to the IFC standard) the external open source BIMServer service -which should be set up separately - is integrated.

BIM IFC connector

CMDBuild integrates an IFC file viewer (ISO 16739:2013) and a connector to import automatically data from IFC files.

The BIM module of CMDBuild is based on a bidirectional data exchange between CMDBuild and the "BimServer" application (Open source Building Information Modelserver).

To configure the BIM features of CMDBuild follow the following steps:

- activate a functioning installation of BimServer
- configure the connection between CMDBuild and BimServer (Administration → Configuration → BIM)
- create a project and upload an IFC file from the page Projects (Administration → BIM → Projects)
- define the class to link the IFC file (e.g. Building) from the page Levels (Administration → BIM → Levels) setting up Root = true
- set up the connection between the project and the class card configured like Root at the previous point

At this point you are able to open the viewer from the data management module.

In order to import automatically some entities defined in the IFC file, please follow these steps:

- set up Active = true the classes where you want to import data from the Level page to
- configure mapping criteria between IFC entities and CMDBuild classes inserting an xml bloc into the column ImportMapping of the table _BimProject in the CMDBuild database; an example of the format supported by this field is as follows:

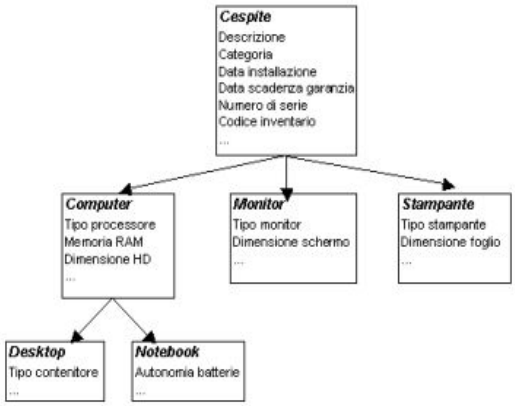
```
<bim-conf>
        <entity name="IfcBuilding" label="Building">
                <attributes>
                        <attribute type="simple" name="Name" label="Code" />
                        <attribute type="simple" name="Description" label="Name" />
                </attributes>
        </entity>
</bim-conf>
```

where

- “IfcBuilding” is the ifc source type, “Building” is the CMDBuild target class
- “Name” is the ifc source attribute, “Code” is the CMDBuild target attribute
    - “simple” is the IFC attribute type expected by the scheme that defines the IFC language
- click on the button “Import IFC” in the Project page in order to start the import

Database design details

Design Criteria



The database design had to respond to a set of basic requirements:

- managing the multi-level hierarchical structure of classes (superclass / subclass), in order to specialize a class while maintaining general attributes in the superclasses
- managing relationships "many to many" between classes
- tracing the complete history of data changes over time and relationships

For all three requirements have been identified as particularly useful the mechanism of "derivation" between classes made available by the open source object-relational database PostgreSQL.

It was therefore decided to use PostgreSQL as database support for CMDBuild, thereby designing in an extremely natural way the structures described above.

Inheritance

1) Multi-level hierarchical structure

Using the keyword "**inherits**" it is possible to create a table in PostgreSQL that "specializes" another, adding some specific attributes and finding all the attributes defined in the superclass.

Example:

```
CREATE TABLE "Asset"
(
    "Id" integer NOT NULL DEFAULT nextval('Asset_SEQ'::text),
    "Code" varchar(100),
    "Description" varchar(250),
    "SerialNo" varchar(40),
    "VersionNo" varchar(32),
    "InstallationDate" timestamp,
    "WarrantyExpireDate" timestamp,
    "State" varchar(16),
    "StateDate" timestamp,
    CONSTRAINT asset_pkey PRIMARY KEY ("Id")
)
```

```
CREATE TABLE "Monitor"
(
        "MonitorType" varchar,
        "ScreenSize" varchar(16),
        "MaxScreenRes" varchar(16)
) inherits ("Asset")
```

2) "Many to many" relations

The different types of relationships between classes are implemented, each with a specific relationship table "many to many", created by derivation from a superclass defined by default in order to simplify the creation of subclasses and to ensure the structural homogeneity.

Example:

```
CREATE TABLE "Map"
(
        "Id" integer NOT NULL DEFAULT nextval('Map_SEQ'::text),
        "IdDomain" regclass,
        "IdClass1" regclass,
        "IdObj1" integer NOT NULL,
        "IdClass2" regclass,
        "IdObj2" integer NOT NULL,
        "Status" character(1),
        "User" character varying(40),
        "BeginDate" timestamp without time zone NOT NULL DEFAULT now(),
        "EndDate" timestamp without time zone,
        CONSTRAINT map_pkey PRIMARY KEY ("Id")
)
CREATE TABLE " Map_aggregation"
(
) inherits ("Map")
```

3) Change history

The derivation mechanism of the classes in PostgreSQL is used also for the management of the history of changes. In order to achieve this, we create a derived class for each type of object, in which through special triggers the current record is stored in the database before changing its attributes, associating the "Id" of the record, the modification date and the login of the operator who made the change.

Example:

```
CREATE TABLE "Monitor"
(
    "MonitorType" varchar,
    "ScreenSize" varchar(16),
    "MaxScreenRes" varchar(16)
) inherits ("Asset")

CREATE TABLE "Monitor_history"
(
    "CurrentId" integer NOT NULL,
    "EndDate" timestamp NOT NULL DEFAULT now()
) inherits ("Monitor")
```

Through the same mechanism we handle the story of changes in relationships, that is, by creating a derived class for each type of relationship, in which through special database triggers records are stored before deleting the current relationship. This is done by attaching the "Id" of the record, the modification date and the login of the operator who made the change.

Primitive superclasses: "Class" and "Map"

On the basis of the philosophy described above we defined two system superclasses called respectively:

- "Class," which is the generic data class from which to derive the specific cards of the different data models
- "Map", which is the generic table of relationships between pairs of classes

The meaning of the attributes of both superclasses is described in the following table:

"Class" Superclass		
Activity name	**SQL Type**	**Description**
Id	integer	table primary key
IdClass	regclass	Table OID
BeginDate	timestamp	insertion date
User	varchar(40)	the user who inserted the record
Status	character(1)	logical state of the record (A = active, deleted = N, U = updated)
Code	varchar(100)	data field: it can be used to store the identifier of a record (e.g. the serial number of a computer) the "Code" field can be renamed (e.g. "VATCode" in a class containing customers or suppliers) or it can be disabled.

Description	varchar(250)	data field: is the description of the record in CMDBuild it is very important since the value is shown in the selection lists to set the "reference" attributes (foreign keys of other classes)
Notes	text	free notes

"Map" Superclass		
Activity name	**SQL Type**	**Description**
IdDomain	regclass	Table OID
IdClass1	regclass	OID of the first class
idObj1	integer	index of the object in relation for the first class
IdClass2	regclass	OID of the second class
idObj2	integer	index of the object in relation for the second class
User	varchar(40)	the user who inserted the record
Status	character(1)	logical state of the record (A = active, deleted = N, U = updated)
BeginDate	timestamp	insertion date
EndDate	timestamp	insertion date

All the tables defined in the CMDBuild application inherit from the "Class" table. This table add specific descriptive fields to the information the tables represent.

Derived tables can be "Super Class" or normal "Class". Only the latter contain data, the first are used as a logical grouping of classes useful for defining attributes common to multiple classes and domains in a more general way.

Each normal class in the database is accompanied by a further table, dedicated to the changed data history, whose name is the same but completed with the suffix "_history". The "historical" tables inherit from the tables and share all the fields with the tables from which they derive, completed only with the link to the active card and with the link to the expiry date.

Each time a record of the main table undergoes a change or cancellation the original record is moved to the history table. During this operation the values of the column "Status" (according to the table above) and of the column "EndDate" (with the timestamp of the transaction) are updated.

Similarly, all domains between Classes or Super Classes inherit from the "Map" Table and each inherited domain holds the corresponding table for versioning (with the same suffix "_history").

By convention all domains have the prefix "Map_".

Metadata

To define the data model CMDBuild uses a set of metadata, associated to the tables and their attributes which extend the basic metadata managed by PostgreSQL.

For storing such metadata the comments associated with tables and columns are used. They are stored in the system table "pg_description".

The metadata structure follows a strict and forced syntax, structured according to a key / value that follows the syntax defined by the following regular expression:

((\[A-Z0-9]+): ([#A-Za-z0&-9éèìùòà_\-\:\s]*)|+)*

For example, a comment can be:

MODE: read|DESCR: some text

The valid comments related to a class as a whole are:

Metadata for class definition			
Key	**Meaning**	**Possible values**	**Notes**
MODE	access modes	reserved|read|write	mandatory
TYPE	table type	class|domain	mandatory
SUPERCLASS	indicates whether the table is a superclass or not	true|false	default = false
DESCR	description shown to the user	any value	mandatory
STATUS	status of use of the table	active|not active	mandatory

All other keys are ignored by the system that interprets the metadata.

The valid comments related to an attribute are:

Attributes metadata			
Key	**Meaning**	**Possible values**	**Notes**
MODE	access modes	reserved|read|write	mandatory
DESCR	description shown to the user on the management form (label)	any value	mandatory
INDEX	display order of the attribute on the management form	a number	physical position in the DB, if absent
BASEDSP	indicates whether the attribute is shown in the "grid" display	true|false	default = false
GROUP	Display "page" of the attribute (for paging)	a string containing the label assigned to the page	
FIELMODE	Field display mode	write|hidden|read	
REFERENCEDOM	domain used to set a reference field	a valid domain for the class	not mandatory
REFERENCEDIRECT	direction of the relationship with respect to the corresponding domain	true|false	Mandatory only if you valorized REFERENCEDOM
REFERENCETYPE	reference type	restrict	Mandatory only if you

	according to delete operations		valorized REFERENCEDOM
LOOKUP	lookup list bound to the attribute	a lookup list	not mandatory
STATUS	status of use of the table	active|not active	mandatory

All other keys are ignored by the system that interprets the metadata.

The users are advised not to intervene manually on such metadata, so as not to cause serious malfunctions in the CMDBuild mechanisms and consequently in the consistency of the data stored in the system.

APPENDIX: Glossary

ATTACHMENT

An attachment is a file associated to a card.

Attachments containing text (PDF, Open Office, Microsoft Word, etc.) are indexed in full text mode so that they can appear in search results.

WORKFLOW STEP

"Activity" means one of the steps of which the process consists.

An activity has a name, an executor, a type, possible attributes and methods with statements (CMDBuild API) to be executed.

A process instance is a single process that has been activated automatically by the application or manually by an operator.

See also: Process

ATTRIBUTE

The term refers to an attribute of a CMDBuild class.

CMDBuild allows you to create new attributes (in classes and domains) or edit existing ones.

For example, in "supplier" class the attributes are: name, address, phone number, etc..

Each attribute corresponds, in the Management Module, to a form field and to a column in the database.

See also: Class, Domain, Report, Superclass, Attribute Type

BIM

Method with the aim to support the whole life cycle of a building: from its construction, use and maintenance, to its demolition, if any.

The BIM method (Building Information Modeling) is supported by several IT programs that can interact through an open format for data exchange, called IFC (Industry Foundation Classes).

See also: GIS

CI

We define CI (Configuration Item) each item that provides IT service to the user and has a sufficient detail level for its technical management.

CI examples include: server, workstation, software, operating system, printer, etc.

See also: Configuration

CLASS

A Class is a complex data type having a set of attributes that describe that kind of data.

A Class models an object that has to be managed in the CMDB, such as a computer, a software, a service provider, etc.

CMDBuild allows the administrator - with the Administration Module - to define new classes or delete / edit existing ones.

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

See also: Card, Attribute

CONFIGURATION

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

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

See also: CI, ITIL

DASHBOARD

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

See also: Report

DATABASE

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

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

DOMAIN

A domain is a relation between two classes.

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

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

It is possible to define custom attributes for each domain.

See also: Class, Relation

DATA FILTER

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

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

See also: Class, View

GIS

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

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

See also: BIM

GUI FRAMEWORK

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

See also: Mobile, Webservice

ITIL

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

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

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

See also: Configuration

LOOKUP

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

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

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

MOBILE

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

See also: GUI Framework, Webservice

PROCESS

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

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

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

See also: Workflow step

RELATION

A relation is a link between two CMDBuild cards or, in other words, an instance of a given domain.

A relation is defined by a pair of unique card identifiers, a domain and attributes (if any).

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

See also: Class, Domain

REPORT

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

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

See also: Class, Domain, Database

CARD

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

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

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

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

See also: Class, Attribute

SUPERCLASS

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

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

See also: Class, Attribute

ATTRIBUTE TYPE

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

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

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

See also: Attribute

VIEW

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

logical way.

In particular, a view can be defined in CMDBuild by applying a filter to a class (so it will contain a reduced set of the same rows) or specifying an SQL function which extracts attributes from one or more related classes.

The first view type maintains all functionalities available for a class, the second one allows the sole display and search with fast filter.

See also: Class, Filter

WEBSERVICE

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

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

CMDBuild includes a SOAP and a REST webservice.

WIDGET

A widget is a component of a GUI that improves user interaction with the application.

CMDBuild uses widgets (presented as "buttons") that can be placed on cards or processes. The buttons open popup windows that allow you to insert additional information, and then display the output of the selected function.