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

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

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

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

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

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

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

Introduction ................................................................................................................................................. 4
  CMDBuild modules ................................................................................................................................. 4
  Available documentation ....................................................................................................................... 5

Description of the workflow system ........................................................................................................ 6
  General Information ............................................................................................................................... 6
  Purposes .................................................................................................................................................. 6
  Used tools ................................................................................................................................................ 6
  Terminology ............................................................................................................................................. 7
  Refactoring 2.0 ....................................................................................................................................... 8

Implementation method ............................................................................................................................. 10
  Workflows as special classes .................................................................................................................. 10
  Building the workflow ........................................................................................................................... 10
  Defining a new process ............................................................................................................................ 11
  Initiation and progress of a process ......................................................................................................... 12

Interaction of the workflow with external tools ....................................................................................... 14
  General Information ............................................................................................................................... 14
  Start of a process from an intranet portal via CMDBuild GUI Framework .............................................. 14

Example of configuration of a new process ............................................................................................... 16
  General Information ............................................................................................................................... 16
  Description of the RfC process used as example .................................................................................... 16
  Phase 1 – Items creation in CMDBuild .................................................................................................... 17
  Phase 2 – Configuration of the flow with TWE ....................................................................................... 23
  Phase 3 – Importation of the XPDL file in CMDBuild .......................................................................... 26
  Phase 4 – Implementation of the process from CMDBuild .................................................................... 27

Widgets prompted to use in the user activities of the workflow ............................................................. 36
  Widget list .................................................................................................................................................. 36

API prompted to use in the automatic activities of the workflow ......................................................... 44
  General Information ............................................................................................................................... 44
  Key words ................................................................................................................................................ 44
  Management of CMDBuild items ............................................................................................................ 44
  Access methods to CMDBuild ................................................................................................................ 47
  Methods for types conversion ................................................................................................................ 58

Appendix: Documentation to use TWS 2.3 ............................................................................................ 60
  Foreword ................................................................................................................................................ 60
  Automatic methods used in the workflow ............................................................................................... 60
  Template automatic methods usable in the workflow ............................................................................ 67

APPENDIX: Glossary .................................................................................................................................. 69
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 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.

**Available documentation**

This manual describes the workflow process included in the CMDBuild application, through which you can configure (Administration module) and run (Management Module) processes for the management of collaborative activities.

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

• system overview ("Overview Manual")
• system administration ("Administrator Manual")
• system usage ("User Manual")
• installation and system management ("Technical Manual")
• webservice details and configuration ("Webservice Manual")
• connectors to sync data through external systems ("ConnectorsManual")
Description of the workflow system

General Information

In order to support ITIL methodological indications, CMDBuild is able not only to manage the update of the asset inventory and their functional relations, but also to enable the definition and control of the processes for IT service management.

A process includes an activity sequence, carried out by operators and/or computer applications, every application represents an operation that has to be carried out within the process, related, in this case, to the IT asset management with quality criteria.

Given the amount of processes options, the organizational procedures and the flexibility pursued by the CMDBuild project, we chose not to implement a series of rigid and predefined processes, but a generic workflow engine to model processes case-by-case.

In the first part of this document you will find general concepts and basic mechanisms implemented in the system with the CMDBuild 2.0 refactoring.

In the second part there is a sample of simplified workflow, described in its configuration steps.

In the third part, you will find the technical tools available for the configuration of a workflow: widgets definition, description of API functions which can be used in the scripts for the definition of automatisms, performed in the workflow.

In appendix there is the specified technical documentation of the workflow system used until CMDBuild 1.5, whose compatibility is maintained also in CMDBuild 2.0; it will be discarded as soon as possible.

Purposes

The workflow management system is an important feature of CMDBuild and provides:

- a standard interface for users
- a secure update of the CMDB
- a tool to monitor provided services
- a repository for activities data, useful to check SLA

ITIL processes, which can be configured in CMDBuild, include: Incident Management, Problem Management, Change Management, Configuration Management, Service Catalogue Management, etc.

Other workflow types concern asset movement, entry of new staff, activation of new work projects, etc.

Used tools

The application chosen for the workflow management uses the following tools:

- XPDL 2.0 (http://www.wfmc.org/xpdl.html) as definition language (standardized from WfMC, WorkFlow Management Coalition according to the model as follows)
- open source TWS Together Workflow Server 4.4 engine (http://www.together.at/prod/workflow/tws), an extensible framework for a complete and standard implementation of the specific WfMC (http://www.wfmc.org/) and OMG, using XPDL as a native language
- the graphical editor TWE Together Workflow Editor 4.4 (http://www.together.at/prod/workflow/twe) for the workflow design and for the definition of
integration mechanisms with CMDBuild

The following schema shows the workflow management according to the model standardized with the WfMC.

Terminology

The following "vocabulary" includes the following terms:

- **process**: sequence of steps that realize an action
- **activity**: workflow step
- **process instance**: active process created executing the first step
- **activity instance**: creation of an activity, accomplished automatically or by an operator

The above terms are arranged into CMDBuild as follows:

- each process is related to a special class defined by the Administration Module under the heading "Processes"; the class includes all attributes of the scheduled activities
- each "process instance" corresponds to a card of the "process" class (current activity), combined with the list of its versions (ended activities)
- each activity instance corresponds to a card of the "process" class (current activity) or to a historicized version (ended activity)

Each process has a name, one or more participants, some variables and a sequence of activities and transitions which carry out it.

The process status can be:

- "active", i.e. it is still in an intermediate activity
- "complete", i.e. it ends its activity
- "aborted", i.e. unnaturally terminated
- "suspended", i.e. maintained only for retrocompatibility with workflow system until CMDBuild
1.5

Each activity can be distinguished by:

- a name
- a performer, which necessarily corresponds to a "user group" and optionally to an operator
- a type: process start, process ending, activity performed by an operator, activity automatically carried out by the system
- any attributes coming from CMDBuild or inside the workflow, which will be set during its implementation
- any widgets (visual controls of some predefined typologies) that will be activated during its implementation
- a script (in the Beanshell, Groovy or Javascript languages), provided in the automatic activities, through which the operations between an user activity and the following can be carried out

Refactoring 2.0

With the 2.0 release we revised the workflow system, with upgrade to Together Workflow Server 4.4, 2.0 XPDL standard adoption, full support in CMDBuild to the native parallelism in the flow and important performance improvements.

In order to simplify the writing we decided to provide a different definition modality of the automatic activities, supporting the scriptwriting and excluding the use of "tools", available in the previous version of CMDBuild.

The scripts can be written in the BeanShell, Groovy or Javascript language and can use API functions provided for the definition of automatisms (manipulation of process variables, card creation and relations in CMDB, e-mail sending, report creation, etc).

The adoption of the new workflow system implies the lost of the retrocompatibility with workflows developed up to the present day.

In order to grant longer period for the migration of the old workflows to the new adopted solutions we decided to maintain in CMDBuild 2.0 the possibility to work - alternatively - both with Together Workflow Server 2.3 (the version used until CMDBuild 1.5, based on XPDL 1.0) and with the new version Together Workflow Server 4.4 (based on XPDL 2.0).

Then the adopted solution allows:

- new CMDBuild users to work with the new Together Workflow Server 4.4 and with the new functionalities developed in the version 2.0 (native parallelism, automatisms configured through scripts)
- old users to split the migration into two steps:
  1. to activate the 2.0 version immediately to make use of the new dashboards and other implemented functionalities, maintaining Together Workflow Server 2.3 active (with improved performances)
  2. to commute to Together Workflow Server 4.4 just after the test of the new environment on the test instance, when the conversion tool is available.

The support tool to the workflows migration developed with the previous CMDBuild versions will be then released.

It is advisable to migrate in a short time, since the double CMDBuild compatibility with Together
Workflow Server 2.3 (XPDL 1.0) and Together Workflow Server 4.4 (XPDL 2.0) will be maintained for a limited period.
Implementation method

Workflows as special classes

The mechanisms for the workflow management are implemented in CMDBuild through concepts and procedures entirely consistent with the mechanisms already in the system for the management of the cards.

The workflow management includes:

- "special" Process classes; each corresponds to a type of workflow
- attributes, corresponding to the information presented (for read or write) in the forms which manage the realization of each single step of the process
- relations with other process instances or standard cards involved in the process
- users' groups, that will be able to perform every activity, coinciding with CMDBuild users' groups
- special tools for customizing the behavior of the workflow (widgets and scripts written with proper APIs)

Within the same homogeneity criteria between "normal" and "process" classes, we adopted the following technical tactics:

- the new "confidential" superclass called "Activity" contains some attributes shared with specific workflows, whose workflows are underclasses
- the "history" mechanism was used to draw the progress reports of a process
- the "relations" mechanism has been kept to create automatic or manual links between a card and a process instance, or between two process instances

Building the workflow

The tools usable through the workflow visual editor are of utmost importance in enabling the design of complex processes, and include:

- the choice of those attributes which can be placed on each form corresponding to a user activity
- the choice of those widgets (visual controls) which can be placed on each form corresponding to a user activity (viewing or creating or editing cards, viewing or creating relations, single or multiple selection of cards, upload of the attached files, implementation of reports)
- flow-control mechanisms, among them parallel activities and subprocesses
- scripting language (BeanShell, Groovy or Javascript) for the definition of those automatisms which must be carried out between a user activity and the following
- API functions which can be called in the scripts

If you are interested in the documentation of further mechanisms used in the workflows, developed for CMDBuild versions until 1.5 (and supported in CMDBuild 2.0 if you use Together Workflow Server 2.3), see the documentation in Appendix (dedicated to the presentation of basic tools and to the mechanism which defines the custom tool through proper templates).
Defining a new process

To create a new "Process" class, you should follow the next logic sequence of passages:

• analysis of the new process which has to be implemented, in order to single out:
  ▫ list of the users' groups involved in the process
  ▫ workflow: user activities, automatic activities, transition conditions, etc
  ▫ descriptive attributes of the process in its user activities and the related typologies
    (strings, integer, etc) and the presentation mode (read-only, reading and writing,
    possible compulsory)
  ▫ predefined lists of values required for the creation of "Lookup" attributes
  ▫ domains required to deal correlations between the new process and other classes or
    other pre-existing processes (which might also be used to create the "Reference"
    attributes)
  ▫ widgets to configure in every user activity
  ▫ widgets to configure in every automatic process activity

• creation of the new process class, which will be defined in the "Processes" section of the
  CMDBuild Administration Module, complete of:
  ▫ specific attributes identified in the previous step
  ▫ domains identified in the previous step

• creation of missing users' groups, that should be added through the Administration Module

• through the Administration Module (from the "XPDL" TAB available for each "Process" class)
  export of the new process template, which includes:
  ▫ process name
  ▫ list of process attributes, that will be placed in the various user activities
  ▫ list of "actors" (users) that interact with the process (the "System" role is automatically
    created to position system activities)

• design of detail flow of the workflow using the TWE external editor, which will help the
  completion of the template exported by CMDBuild

• save, using the special functions of TWE external editor of the XML file (to be exact XPDL
  2.0) corresponding to the designed process

• import of the process schema in CMDBuild, using the special "XPDL" TAB, available under
  the heading "Processes" in the Administration Module

Once concluded the operations described above, the new process can be used in the
Management Module, (Menu “Processes” or headings like “process” in the Navigation Menu), thus
the process can be executed using the workflow engine Together Workflow Server 4.4.

The above mentioned operations can be carried out when you need to edit an imported process,
but the changes must be received only through the new process instances which will be started.
Initiation and progress of a process

In the Management Module, CMDBuild can perform, through the support of the TWS Together Workflow Server engine, the processes designed with TWE Together Workflow Editor and then imported through the Administration Module.

In order to keep the greatest coherence with the CMDBuild functionalities, dedicated to the management of the items cards in the system, the user interface of the Management Module was designed so that it is consistent with the management of the normal data "classes":

- there is a special menu item "Processes" consistent with the "data sheets" (otherwise "process" elements can be inserted in the Navigation menu with "data sheets" elements or reports and dashboards)
- the process management draws on the standard managements which already exist for the normal cards: “List”, “Card”, “Details” “Notes”, “Relations”, “History”, “Attachments”
- in the “List” TAB of a specific process, the user can see the activities instances, in which he/she is involved (since he/she attends that activity or previous activities of that process) with:
  - filters by status (started, completed, suspended)
  - data area with tabular display of the information (process name, activity name, request description, process status and further attributes defined as "display base" in the Administration Module), which you can click on in order to access to the management card of that activity
  - possible evidences of parallel activities for that process instance
  - buttons to create a new activity or to make that choice
• in the “Card” TAB you can visualize or fill in the attributes provided for that process activity instance (write or read-only access can be set up through the TWE editor) otherwise you can carry out further operations through the proper widgets (visual controls) configured with the TWE editor
• in the “Notes” TAB you can visualize or insert notes about the activity instance
• in the “Relations” TAB you can visualize or insert relations between the activity instance and the instances of other classes (“cards”)
• in the “History” TAB you can visualize the previous versions of that activity instance (instances already carried out)

The list of activities is displayed high up in the following exemplifying form, while you can carry out an activity filling in the card at the bottom.

Since workflows are peculiar classes, you can find the control buttons also in the upper right of the workflow management form to display full screen the upper or lower side of the form.
Interaction of the workflow with external tools

General Information

In some cases it may be required that a process (like a new request of HelpDesk) is started by a user who is not expert enough at using CMDBuild (such as the user of the item or of the IT service).

This can be solved using the CMDBuild GUI Framework, described as follows.

Start of a process from an intranet portal via CMDBuild GUI Framework

The GUI Framework is a javascript / Jquery development environment, used to implement a simplified user interface. Thanks to it, non-expert users can interact with the CMDBuild application.

The GUI Framework includes the following main features:

- it can be activated in portals based on different technologies
- 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 adapts to workflow advancement forms designed through the visual editor TWE
- 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

An sample of implementation based on the CMDBuild GUI Framework is the Self-Service portal, which is part of the preconfigured version CMDBuild READY2USE.

The CMDBuild Ready2Use Self-Service portal allows non-technician users to interact with the IT employees in order to point out their needs and to keep up to date on the resolution activities.

Every user can access to the portal upon local authentication or authentication connected to the Active Directory repository of the company.

The home page of the portal includes:

- a complete menu, on the left
- a quick access to the main features, up on the right
- the most recent IT news
- the advancement situation of the last forwarded requests

CMDBuild suggests an implementation of the functioning portal as portlet in the open source Liferay portal. The used CMDBuild GUI Framework can be activated on portals based on different technologies, since it is developed in javascript / JQuery environment. So, you can ask for custom implementations of the self-service portal, which can work on various portals.

The CMDBuild Ready2Use Self-Service portal implements the following features:

- publication of IT news
- request for technical information
• opening of a damage notice
• request for an IT service, selected from the services catalogue
• consultation of the advancement of your own requests
• approval of authorization requests
• FAQ
• summary of received notification emails
• connected user profile
• list of assets and services assigned to the connected user
• useful links

The CMDBuild GUI Framework is not the only one possible option. You can even implement external web interfaces from the beginning, using your favourite development language and interacting with CMDBuild through its REST and SOAP webservice. But this solution is less efficient if compared to the re-use of the available GUI Framework.
Example of configuration of a new process

General Information

The chosen process that describes the various passages necessary for its configuration is a simplified Request for Change (or RfC) process.

It is an extremely simplified process, modelled only for educational purposes; it was suggested for configuration modalities, not for a real use in the production.

The sample process, complete with the definition in CMDBuild and the XPDL flow designed with TWE, is available on the demo database supplied with CMDBuild.

Description of the RfC process used as example

The actors of the process are the users’ groups:

- Helpdesk, which carries out the initial registration of the request received by a user
- Change Manager, responsible for the changes to the IT assets of the company
- IT expert, involved for the production of analysis documents and for the change execution

Here’s a logic schema of the process:

The process includes the following operations:
Workflow Manual

Example of configuration of a new process

- RfC recording
- evaluation of the request's formal aspects:
  - direct closing if the RfC is not acceptable
  - shift to the decisional step, if analysis activities are not requested
  - execution request of one or more analysis typologies, among impact, cost, risk analysis
- execution of the requested analysis typologies (impact, cost, risk analysis)
- decision by the Change Manager, which might be closed if the RfC is not approved
- RfC execution by an IT expert, if the RfC is approved
- final closing

Phase 1 – Items creation in CMDBuild

Through the Administration Module, under the heading Menu Processes, the process RequestForChange is created in order to manage the workflow:

Some attributes provided in the process are Lookup, so they require the preventive definition of the related lists, as you can see in the following screenshots.
Lookup RFC category (linked to the “Category” attribute of the process)

Lookup RFC decision (linked to the “Decision” attribute of the process)
Lookup RFC final result (linked to the “FinalResult” attribute of the process)
Lookup RFC formal evaluation (linked to the “FormalEvaluation” attribute of the process)

Lookup RFC priority (linked to the “RFCPriority” attribute of the process)

Lookup RFC status (linked to the “RFCStatus” attribute of the process)
The following "domains" are created in order to define in the process the “Requestor” attributes as foreign keys on the class “Employee” and the relations with the Change Manager and IT experts, who respectively assesses and execute the RFC:
At this point the attributes of the process can be created:

Last thing you can create the users' groups involved in the workflow:

At this point you can export the XPDL schema produced by CMDBuild, used with the visual editor.
TWE to design the detail flow of the process itself:

The XPDL file will include only the general data available at the moment:

- process name
- list of unreserved process attributes present in the process management class
- list of the roles defined in the system

Those data will be the starting point of the activities carried out through the TWE editor, in which all aspects related to the specific process flow will be enriched.

Phase 2 – Configuration of the flow with TWE

Through the TWE editor it's possible to perform the following operations:

- flow design by placing the activities of the various provided typologies (process starting and ending, user activities, automatic activities, routing activities for the parallelism management) and their connection according to the provided transition typologies
- completion of user activities, specifying what process attributes will be shown in the form related to that activity (by indicating if read-only or read/write) and what widgets will be made available in the same form (by indicating the parameters provided for each one)
- completion of automatic activities, writing the script which implements the automatisms required in that activity (using the API available for that aim)
- completion of transitions among activities, specifying the criteria for the flow to cover a transition or another, when the choice is binding

Here are some descriptive screenshots of the above mentioned activities.
General flow design:

User activities - “Variables” TAB, used to choose the attributes which must appear in the form (with possible indication of read-only modality):

User activities - “Extended Attributes” TAB, used to indicate the compulsory attributes ("UPDATEREQUIRED") and to request the input in the form of one or more widgets (in the sample
openAttachments for the attachments and createModifyCard to consult the requester card)

**Automatic activity - “Type” TAB**, used to write the script that implements the provided automatisms (in the sample, the activity SYS010 carries out the automatic structuring of the system date, the automatic attribution of a univocal progressive number, the building of a significant description, the structuring of a new state reached by the process).
Workflow Manual

Example of configuration of a new process

Transition, used to link two activities, conditionally or not (in the example it is provided a condition related to the formal acceptance of the RfC)

Phase 3 – Importation of the XPDL file in CMDBuild

When the configuration of the process in TWE is complete, you will load in CMDBuild the related XPDL file.

The process flow can be then modified several times, only exporting the last version from CMDBuild, editing it with TWE and importing it again in CMDBuild. You have to consider that the new version will be used when new processes are started, while each current process will go on with the XPDL version valid when they first started.
Phase 4 – Implementation of the process from CMDBuild

The workflow imported in CMDBuild is available to be used by the provided operators groups.

In the example, the management workflow of the RfC will be started by an operator of the Helpdesk group, valued by an operator of the Change Manager group, analysed and carried out by an operator of the IT expert group. You have to consider that the operators of the SuperUser group can "personify" any other group defined in CMDBuild.

From the RfC process management, the RfC are presented as open (or in the state selected on the upper list: open, suspended, complete, aborted, all).

Through the button “Start Request for Change” the Helpdesk can register a new request.
Previous to filling in the form, the operator can refer to the operative instructions associable with every user activity (which can be formulated with TWE, filling in the field “Description” in the “General” TAB of the activity).
By validating the progress at the following step, the activity is taken by the Change Manager, that - in our simplified example - will fill in the following information:

![Image of Change Manager interface](image1)

In the example we provide at this step the possible use of the enclosed loading widgets and those for the reference of the complete requester card:

![Image of Requester card interface](image2)
The Change Manager demanded in our example two typologies of analysis, so the workflow moves to IT experts, that, in parallel (using one of the new functionalities implemented in CMDBuild 2.0), can carry out their analysis (respectively risk and cost analysis) and transfer the results.
The Change Manager currently provides the results of the requested discussions and can take his/her decision.

If the decision is positive, according to the flow designed with TWE, the IT experts are asked to carry out the RFC activity. At the beginning the operation makes the request with indication of the activities which must be carried out; at the end it registers the activities already carried out.
As last operation, the Change Manager closes the RfC stating a positive result.
At this stage, the RfC we worked on (number 7) will not appear in the list of the open RfC.

But it can be referred with all its information in the list of the completed RfC (the list can be selected in the upper part of the form)
In addition to the basic information, you can refer to the relations configured with that RfC process instance (Relations TAB).
You can also refer to the sequence complete with progress activities of the process (History TAB).

Uploading the attachments during the process (using the proper widget), you can refer to the possible available documents (Attachments TAB).
Widgets prompted to use in the user activities of the workflow

Widget list

CMDBuild makes some widgets available (visual controls), placed in the right part of the form, which manage the progress of the process through the provided activity.

Graphically, such controls are designed with buttons with the specified label during the definition step.

About the configuration, they are defined as “Extended attributes” (provided in the XPDL standard) using the TWE editor.

In this document, the data types are both original (integer, string, date, float, boolean) and complex types added in the workflows of CMDBuild (lookup = id + type + description, lookups = lookup array, reference = id + idclass + description, references = reference array).

<table>
<thead>
<tr>
<th>Visual control</th>
<th>Description</th>
<th>Parameters</th>
<th>Notes</th>
</tr>
</thead>
</table>
| manageRelation | It shows the card list (which can be selected) in relation to the specified card according to the specified domain | Input:  
  DomainName string  
  ClassName string  
  ObjId integer  
  ButtonLabel string  
  EnabledFunctions character array  
  Required integer  
  IsDirect string  
or  
  Input:  
  DomainName string  
  ObjRef reference  
  ButtonLabel string  
  EnabledFunctions character array  
  Required integer  
  Output:  
  CheckArray references | EnabledFunctions is an array of boolean values which enables different functionalities according to the following positional method:  
- link element  
- add and link element  
- activate selection check  
- activate selection radio button  
- modify relation  
- disconnect element  
- modify element  
- delete element  
The parameter Required = 1 must be indicated only if the selection of at least one element is compulsory  
IsDirect can take the values “true” or “false” |
| linkCards | It shows the paginated list - which can be selected - of all cards belonging to a class, with possible display on a geographical map | Input:  
  ClassName string  
  ButtonLabel string  
  SingleSelect integer  
  NoSelect integer  
  Required integer  
  Filter string  
  DefaultSelection string | The parameter SingleSelect = 1 must be indicated only if the selection of one single row is allowed (radio-button rather than checkbox)  
The parameter NoSelect = 1 desables the selection of rows (neither radio button nor checkbox)  
The parameter Required = 1 forces the selection of one row at least |
### Workflow Manual

Widgets prompted to use in the user activities of the workflow

<table>
<thead>
<tr>
<th>AllowCardEditing</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>DisableGridFilterToggler</td>
<td>boolean</td>
</tr>
<tr>
<td>Map</td>
<td>string</td>
</tr>
<tr>
<td>StartMapWithLatitude</td>
<td>integer</td>
</tr>
<tr>
<td>StartMapWithLongitude</td>
<td>integer</td>
</tr>
<tr>
<td>StartMapWithZoom</td>
<td>integer</td>
</tr>
<tr>
<td>Metadata</td>
<td>string</td>
</tr>
<tr>
<td>MetadataOutput</td>
<td>string</td>
</tr>
</tbody>
</table>

**Output:**

CheckArray references [metadataOutput] text

The Filter parameter is a CQL expression (CMDBuild query language)

Example: Filter = “from Person where Id = {client:Customer.Id}"

The optional parameter DefaultSelection specifies the CQL query used for the automatic selection when opening the widget.

The optional parameter AllowCardEditing = 1 adds an icon to edit the card.

The optional parameter DisableGridFilterToggler = “true” hides the button “Disable filter”.

The optional Map parameter enables the map visualization (if it is set = ’enabled’).

The parameters related to the initial presentation of the map are optional.

The Metadata variable accepts as unique value (waiting for future extensions) the ‘point:POINT’ string.

The MetadataOutput variable accepts as unique value the '_metadataOutput' string that represents the name of the output variable.

They both are necessary to manage the selection of a single point on an already existing polygonal.

The point coordinates will be given back in the metadataOutput variable in the WGS84 format.

A possible example:

point:POINT(5847010.6684071 1438393.2786558)

---

### createModifyCard

It shows the specified card in the change (if ObjId is specified), otherwise it allows the creation of a new card in the specified class.

<table>
<thead>
<tr>
<th>Input:</th>
<th>ClassName string ButtonLabel string ReadOnly integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>or</td>
<td></td>
</tr>
<tr>
<td>Input:</td>
<td>Reference reference ButtonLabel string ReadOnly integer</td>
</tr>
<tr>
<td>or</td>
<td></td>
</tr>
<tr>
<td>Input:</td>
<td>ClassName string ObjId integer ButtonLabel string ReadOnly integer</td>
</tr>
</tbody>
</table>

Example:

ClassName=’User’

ObjId=client:Requester

ButtonLabel = ‘Create or modify User’

Requester

Note:

the prefix “client:” is required to access to a variable before the workflow is proceeded to the following step.

ReadOnly=1 shows the card read-only.
<table>
<thead>
<tr>
<th>Widget</th>
<th>Description</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>createReport</td>
<td>It allows to produce a report through template or write free e-mails which will be sent during the development of the process.</td>
<td>ReportType string, ReportCode string, ButtonLabel string, ForcePDF integer, ForceCSV integer, Parameter-1, Parameter-2, ..., Parameter-n</td>
<td>ReportURL string</td>
</tr>
<tr>
<td>manageEmail</td>
<td>It allows to produce e-mails or write free e-mails which will be sent during the development of the process.</td>
<td>ButtonLabel string, ToAddresses string, CCAddresses string, Subject string, Content string, Assignments string, ReadOnly boolean</td>
<td></td>
</tr>
<tr>
<td>openNote</td>
<td>It visualizes the page which includes the HTML editor to insert notes.</td>
<td>ButtonLabel string</td>
<td></td>
</tr>
<tr>
<td>openAttachment</td>
<td>It visualizes the page provided for the uploading of the file which has to be enclosed to the current process.</td>
<td>ButtonLabel string</td>
<td></td>
</tr>
<tr>
<td>calendar</td>
<td>It displays the calendar with the selected dates.</td>
<td>ButtonLabel string, ClassName string, Filter string, EventStartDate date, EventEndDate date, EventTitle string</td>
<td></td>
</tr>
<tr>
<td>presetFromCard</td>
<td>It populates the current activity with those data recovered by a selected card.</td>
<td>ButtonLabel string, ClassName string, Filter string, AttributeMapping string</td>
<td></td>
</tr>
</tbody>
</table>
| **webService** | It displays the result of a call to Web Service (at the moment SOAP only) as a grid. You can select some rows of this grid to obtain their XML serialization as widget output. | **Input:**
- ButtonLabel *string*
- EndPoint *string*
- Method *string*
- NameSpacePrefix *string*
- NameSpaceURI *string*
- NodesToUseAsRows *
- NodesToUseAsColumns *
- SingleSelect='true'
- Mandatory='true'
- ReadOnly='true'
- String parameters
- OutputSeparator *string*

**Output:**
- Output *string* variable

| **Input:**
- EndPoint=Service URL
- Method=Method name
- NameSpacePrefix=namespace prefix' (optional)
- NameSpaceURI='namespace URI' (optional)
- NodesToUseAsRows=
- NodesToUseAsColumns=Names of the elements (separated by commas without spaces) of the answer to display in the grid
- Call parameters (optional) = possible parameters provided for in the Web Service.
- Output variable(optional) that will be optimized through the XML serialization related to the selected nodes. If it is string type, then the separator has also to be specified.
- OutputSeparator (optional)= character to separate the results. If it is missing, they will be given back as string array. |

| **startWorkflow** | It allows to start a workflow according to two modalities: 1) configuration read by widgets 2) configuration read by a “support” table. | **1) Input:**
- ButtonLabel *string*
- WorkflowCode *string*

or

2) **Input:**
- ButtonLabel *string*
- FilterType *string*
- Filter *

**Output:**
- processRef
- ReferenceType

| **1) WorkflowCode name of the starting process**

2) FilterType supports at the moment just “cql” Filter the cql filter to select a series of card from a CMDBuild table. The result of the filter should be the same as the name list of the processes that should be started from the widget itself. |

| **grid** | It allows to manage a row grid (by adding, removing and/or modifying the rows ). | **Input:**
- ClassName *string*
- ButtonLabel *string*
- CardSeparator *string*
- AttributeSeparator *string*
- KeyValueSeparator *

| **ClassName** is the name of the class in which you want to work

CardSeparator separator among the various inserted cards (default ",")

AttributeSeparator separator among the attributes of the same card (default "&") |
### string
- **PresetsType**="function"
- **Presets** string
- **Output:**
  - **Output string variable**

Key-ValueSeparator separator between an attribute and its value (default ":=")
The output variable will be a single string containing the serialization of the inserted data, separated by the above-mentioned characters.

PresetsType and Preset are necessary to upload in advance some values into the grid; these values will be then edited directly by the user. If you want to use a variable as input or the output of another grid, you simply have to specify the key `Preset=InputString` where `InputString` is a formatted string like the grid output.

If you want to upload in advance the grid starting from a function (having as column names the same reference class fields) you have specify `PresetsType="function"` and `Presets="wf_function_name"` where "wf_function_name" is the name of a stored procedure in the database defined according to criteria used to create dashboards.

Any parameter has to be specified in succession in the form of:
- `Param1="value1"` (function input parameter)
- `Param2="value2"` (function input parameter)

There is also the possibility to upload in advance the values into the grid through the proper widget button called "Import from CSV". The file must follow the norms defined for the CSV file importation into CMDBuild. You will also have the possibility to specify the separator and the way to import the data (Replace or Add).

### customForm
- **Input:**
  - **ButtonLabel** string
  - **ModelType** "[form|class|function]"
  - **Layout** "[grid|form]"
  - **DataType** [raw_json|raw_text|function]
  - **ReadOnly** "[true|false]"
  - **Required** "[true|false]"
  - **AddDisabled** "[true|false]"
  - **DeleteDisabled** "[true|false]"
  - **ImportDisabled** "[true|false]"

It allows to manage a form or a row grid (by adding, removing and/or modifying the rows).

The structure of the custom form can be defined starting from:
- form - JSON item array
- class - attributes of a class
- function - function input parameters

The layout can be a form (as if it is a CMDBuild card) or a row series.

The data of the widget can be initialized starting from:
- raw_json - JSON item array
- raw_text – well-structured strings of text
- function – output values of a function

Data can be serialized as type of text (see...
### Workflow Manual

**Widgets prompted to use in the user activities of the workflow**

<table>
<thead>
<tr>
<th>Widget</th>
<th>Description</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModifyDisabled</td>
<td>The <code>ModifyDisabled</code> widget allows you to choose whether a specific action is enabled or disabled. Options are true or false.</td>
<td>`ModifyDisabled = [true</td>
<td>false]`</td>
</tr>
<tr>
<td>SerializationType</td>
<td>This widget provides options for serialization type. Possible values are json and text.</td>
<td>`SerializationType = [json</td>
<td>text]`</td>
</tr>
<tr>
<td>KeyValueSeparator</td>
<td>Use this widget to specify the key-value separator for data.</td>
<td><code>KeyValueSeparator = string</code></td>
<td></td>
</tr>
<tr>
<td>AttributesSeparator</td>
<td>Define the attributes separator used for organizing data.</td>
<td><code>AttributesSeparator = string</code></td>
<td></td>
</tr>
<tr>
<td>RowsSeparator</td>
<td>Set the rows separator for data presentation.</td>
<td><code>RowsSeparator = string</code></td>
<td></td>
</tr>
<tr>
<td>NavigationTreeName</td>
<td>This widget allows you to select one or more data cards through an interface that is based on a preconfigured navigation tree (subset of domain graph).</td>
<td><code>NavigationTreeName = string</code></td>
<td></td>
</tr>
<tr>
<td>ButtonLabel</td>
<td>Use this parameter to label the button for navigation tree.</td>
<td><code>ButtonLabel = string</code></td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>Define the output variable for the grid.</td>
<td><code>Output = string variable</code></td>
<td></td>
</tr>
<tr>
<td>navigationTree</td>
<td>It allows to select one or more data cards through an interface that is based on a preconfigured navigation tree (subset of domain graph).</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**adminStart**

- By a process with more start activities distinct for each group, it singles out the activity for the administer user.
- **Input parameters**
  - `NavigationTreeName = string`  
  - `ButtonLabel = string`  
  - **Output**
    - `CheckArray = references`

**Output parameters**

- No input nor output parameters
- It is an “extended attribute”, not a widget (it doesn't have a user interface), but it is described in this section, since it is configured like widgets.

### Further information for the use of “string template” in the tool manageEmail

The tool `manageEmail` allows to write e-mails which will be sent during the development of the process. Visualizing e-mails, the electronic mailbox will be checked for possible new e-mails to visualize the grid.

**Input parameters**

- `string ButtonLabel`
- one or more blocks for the e-mails definition
  - `string template ToAddresses: recipient’s addresses`
  - `string template CcAddresses: carbon copy addresses`
  - `string template Subject: e-mail subject`
  - `string template Content: e-mail body (HTML)`
  - `string template Condition: javascript expression whose evaluation defines if the e-mail is generated or not`
- other optional parameters which include queries or javascript expressions
- `flag ReadOnly: read-only email`

**Output parameters**

- none

The only-read `flag` is seen as a boolean value; a boolean value (of the process), a positive integer value or a non empty string are considered `true`

In the **template strings** the variables, written in the form `{namespace:localname}`, are interpreted in a different way depending on the namespace (if omitted, it defaults to "server").
Widgets prompted to use in the user activities of the workflow

| client:name | Form’s name variable; for attributes such as LookUp or Reference you have to specify, with the bullet list, whether you want the Id or the Description |
| client:name.Id | |
| client:name.Description | |
| server:name | Process name variable in the previous step |
| xa:name | Variable name of the extended attribute definition, extended as template excluding the variables with namespace js and cql |
| user:id | ID and name of the connected user |
| user:name | |
| group:id | ID and name of the connected group |
| group:name | |
| js:name | Variable name of the extended attribute definition interpreted as a template and evaluated as a javascript code |
| cql:name.field | Variable name of the extended attribute definition interpreted as a template and evaluated carrying out a CQL query, whose field is identified by field |

The definition blocks of the e-mails can be written in two ways:

```
ToAddresses="..."
CcAddresses="..."
Subject="..."
Content="...
```

or (if you want to specify more than one e-mail):

```
ToAddresses1="..."
CcAddresses1="..."
Subject1="..."
Content1="...
ToAddresses2="..."
CcAddresses2="..."
Subject2="..."
Content2="...
```

**Example 1**

```
ToAddresses="foo@example.com"
Subject="{cql:QueryRequester.Description} - {client:Request}" QueryRequester="select Description,Email,Office from Employee where Id = {cql:SillyQuery.Id}" SillyQuery="select Id from Employee where Id={client:Requester}"
```

Address: The recipient's address is statically completed with the string foo@example.com

Body: Message Body Empty

Subject:

- The variable QueryRequester selects an Employee card which includes the fields Description, Email and Office; the extracted values are available using for example the syntax `\{cql:QueryRequester.Description\}`, which will be replaced with the field Description extracted from the variable QueryRequester
- Inside QueryRequester, `\{cql:SillyQuery.Id\}` will be replaced with the Id field of the card returned from the SillyQuery (indeed nested queries are supported), replaced before with `\{client:Requester\}` with the value taken in the form
- `\{client:Request\}` of will be completed with the form value
Example 2

...Content="The requester, {js:JoinJS}, belonging to the office {cql:QueryRequester.Office_value} requests:<br />
   /\{server:Request\}" JoinJS="\{js:FirstJS\}+#+\{js:SecondJS\}"
FirstJS="\{cql:QueryRequester.Description\}.slice(0,\{xa:SplitLength\})"
SecondJS="\{cql:QueryRequester.Description\}.slice(\{xa:SplitLength\})"
SplitLength=2 QueryRequester="select Description,Email,Office from Employee where Id = {Requester}" This is an example of more complexity.

In the body there are three variables which must by replaced:

- {js:JoinJS} values the extended attribute variable like a javascript expression, splitting with # the variables FirstJS and SecondJS, always valued through javascript
- {js:FirstJS} and {js:SecondJS} include both a variable taken from a field of CQL query QueryRequester and a static variable taken from the ones of the extended attribute
- {cql:QueryRequester...} includes a reference to a server side variable called Requester
- {cql:QueryRequester.Office_value} uses the Office reference description instead of its ID (that would be just Office)
- {server:Request} takes a server side variable (as Requester), but it also states the namespace
API prompted to use in the automatic activities of the workflow

In CMDBuild there are some APIs (Application Programming Interface) which can be used in the automatic activities of the workflow for the script writing; so it is possible to implement custom behaviors (manipulation of process variables, card creation and relations in CMDB, e-mail sending, report creation, etc).

- The condition to send e-mails is always verified since \{xa:SplitLength\} is constant and the javascript expression is always true.

General Information

Key words

<table>
<thead>
<tr>
<th>Process</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ProcessId: int</td>
<td>Id of the current process</td>
</tr>
<tr>
<td>ProcessClass: String</td>
<td>Class name of the current process</td>
</tr>
<tr>
<td>ProcessCode: String</td>
<td>univocal ProcessInstanceId of the current process</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performer</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>_CurrentUser: ReferenceType</td>
<td>reference to the User that performed the last activity of the current process</td>
</tr>
<tr>
<td>_CurrentGroup: ReferenceType</td>
<td>reference to the Role that performed the last activity of the current process</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>API</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cmdb</td>
<td>it identifies the native functions in CMDBuild</td>
</tr>
</tbody>
</table>

Management of CMDBuild items

They concern the CMDBuild specific data; for other data (integer, string, date, float) you can use all manipulation methods offered by the Java language.

<table>
<thead>
<tr>
<th>ReferenceType</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Methods</td>
<td></td>
</tr>
<tr>
<td>getld(): int</td>
<td></td>
</tr>
</tbody>
</table>
It returns the reference id

`getDescription(): String`
It returns the reference description

### LookupType

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getId(): int</code></td>
<td>It returns the lookup id</td>
</tr>
<tr>
<td><code>getType(): String</code></td>
<td>It returns the type of lookup</td>
</tr>
<tr>
<td><code>getDescription(): String</code></td>
<td>It returns the lookup description</td>
</tr>
<tr>
<td><code>getCode(): String</code></td>
<td>It returns the lookup code</td>
</tr>
</tbody>
</table>

### CardDescriptor

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getClassName(): String</code></td>
<td>It returns the class name for a CardDescriptor variable</td>
</tr>
<tr>
<td><code>getId(): int</code></td>
<td>It returns the id name for a CardDescriptor variable</td>
</tr>
<tr>
<td><code>equals(CardDescriptor cardDescriptor): boolean</code></td>
<td>It compares the CardDescriptor variable with the specified one</td>
</tr>
</tbody>
</table>

### Card

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getCode(): String</code></td>
<td>It returns the code for a Card variable</td>
</tr>
<tr>
<td><code>getDescription(): String</code></td>
<td>It returns the description for a Card variable</td>
</tr>
<tr>
<td><code>has(String name): boolean</code></td>
<td>It controls the presence of the specified attribute in the Card variable</td>
</tr>
<tr>
<td><code>hasAttribute(String name): boolean</code></td>
<td>It controls the presence of the specified attribute in the Card variable</td>
</tr>
<tr>
<td><code>get(String name): Object</code></td>
<td>It returns the specified attribute value of the Card variable</td>
</tr>
<tr>
<td><code>getAttributeNames(): Set&lt;String&gt;</code></td>
<td>It returns the attributes list of the Card variable</td>
</tr>
<tr>
<td><code>getAttributes(): Map&lt;String, Object&gt;</code></td>
<td>It returns the attributes list of the Card variable</td>
</tr>
</tbody>
</table>
it returns the attributes list and their values of the Card variable. The returned values respect the CMDBuild types (ReferenceType, LookupType, Date, Integer, ...)

### Attachments

#### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fetch(): Iterable&lt;AttachmentDescriptor&gt;</td>
<td>it returns the attachments list of the Card or of the instantiated process</td>
</tr>
<tr>
<td>upload(Attachment... attachments):void</td>
<td>it attaches the documents to the card or to the instantiated process</td>
</tr>
<tr>
<td>upload(String name, String description, String category, String url):void</td>
<td>it creates an attachment with name, description and category specified starting from the file with the specified URL and attaches it to the card or to the instantiated process</td>
</tr>
<tr>
<td>selectByName(String... names): SelectedAttachments</td>
<td>it returns the attachments of the card or of the instantiated process with the specified name</td>
</tr>
<tr>
<td>selectAll(): SelectedAttachments</td>
<td>it returns all attachments of the card or of the instantiated process</td>
</tr>
</tbody>
</table>

#### AttachmentDescriptor

#### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getName(): String</td>
<td>it returns the name of the attachment</td>
</tr>
<tr>
<td>getDescription(): String</td>
<td>it returns the attachment description</td>
</tr>
<tr>
<td>getCategory(): String</td>
<td>it returns the attachment category</td>
</tr>
</tbody>
</table>

#### Attachment

#### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getUrl(): String</td>
<td>it returns the URL of the file</td>
</tr>
</tbody>
</table>

#### DownloadedReport

#### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getUrl(): String</td>
<td>it returns the local URL where the report has been saved</td>
</tr>
<tr>
<td>equals(DownloadedReport downloadedReport): boolean</td>
<td>it compares the DownloadedReport variable with the specified one</td>
</tr>
</tbody>
</table>
## Access methods to CMDBuild

### NewCard

<table>
<thead>
<tr>
<th>Builders</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>newCard(String className): NewCard</td>
<td>it creates a new Card created in the specified Class of CMDBuild</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modifiers</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>withCode(String value): NewCard</td>
<td>it adds the Code to the new card created in CMDBuild</td>
</tr>
<tr>
<td>withDescription(String value): NewCard</td>
<td>it adds the Description to the new card created in CMDBuild</td>
</tr>
<tr>
<td>with(String name, Object value): NewCard</td>
<td>it adds the value specified for the specified attribute to the new card created in CMDBuild</td>
</tr>
<tr>
<td>withAttribute(String name, Object value): NewCard</td>
<td>it adds the value specified for the specified attribute to the new card created in CMDBuild</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>create(): CardDescriptor</td>
<td>it creates the new card in CMDBuild setting the attributes previously defined</td>
</tr>
</tbody>
</table>

### Example:

```java
/*
 * Creation of a new card in the “Employee” class having
 * the following attributes:
 * “Code” = “T1000”
 * “Name” = “James”
 * “Surname” = “Hetfield”
 */

```

### ExistingCard

<table>
<thead>
<tr>
<th>Builders</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>existingCard(String className, int id): ExistingCard</td>
<td>it creates a Card existing in the specified Class having the specified Id to query CMDBuild</td>
</tr>
<tr>
<td>existingCard(CardDescriptor cardDescriptor): ExistingCard</td>
<td></td>
</tr>
</tbody>
</table>

---

**CMDBuild** – Open Source Configuration and Management Database

Page 47
it creates an existing Card indicated by the specified CardDescriptor to query CMDBuild

<table>
<thead>
<tr>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>withCode(String value): ExistingCard</code> it sets the Code for the Card requested to CMDBuild</td>
</tr>
<tr>
<td><code>withDescription(String value): ExistingCard</code> it sets the Description for the Card requested to CMDBuild</td>
</tr>
<tr>
<td><code>with(String name, Object value): ExistingCard</code> it sets the specified attribute with the specified value for the Card requested to CMDBuild</td>
</tr>
<tr>
<td><code>withAttribute(String name, Object value): ExistingCard</code> it sets the specified attribute with the specified value for the Card requested to CMDBuild</td>
</tr>
<tr>
<td><code>withAttachment(String url, String name, String category, String description): ExistingCard</code> it attaches a file (pointed out through a server local url) to the selected card by setting the file name, its category and its description</td>
</tr>
<tr>
<td><code>attachments(): ExistingCard</code> it allows you to access the attachments of the selected card</td>
</tr>
<tr>
<td><code>selectAll(): ExistingCard</code> it allows you to select all documents of the selected card</td>
</tr>
<tr>
<td><code>selectByName(String name1, String name2, ...)</code>: ExistingCard it allows you to select all documents of the selected card</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>update()</code> it updates the Card in CMDBuild by setting the attributes previously indicated with the specified values</td>
</tr>
<tr>
<td><code>delete()</code> it deletes (logic delete) the Card from CMDBuild If the &quot;attachments&quot; modifier has been used, it will delete only the selected files</td>
</tr>
<tr>
<td><code>fetch(): Card</code> it requests the Card to CMDBuild with the attributes previously indicated. If no modifier has been used, it requests the whole Card (with all attributes)</td>
</tr>
<tr>
<td><code>fetch(): Iterable&lt;AttachmentDescriptor&gt;</code> If the &quot;attachments&quot; modifier has been used, the method returns the list of the card attachments</td>
</tr>
<tr>
<td><code>upload(Attachment attachment, Attachment attachment2,...)</code> to be used in the presence of the &quot;attachments&quot; modifier: it attaches one or more files to the card</td>
</tr>
<tr>
<td><code>upload(Attachment attachment, String description, String category, String url)</code></td>
</tr>
</tbody>
</table>
to be used in the presence of the "attachments" modifier: it attaches to the card a single file with specified description and category

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>download()</td>
<td>If the &quot;attachments&quot; modifier has been used, the method returns the selected attachments of the card</td>
</tr>
<tr>
<td>copyTo()</td>
<td>If the &quot;attachments&quot; modifier has been used, the method copies a selected attachment of the card into a specified destination</td>
</tr>
<tr>
<td>moveTo()</td>
<td>If the &quot;attachments&quot; modifier has been used, the method moves a selected card attachment into a specified destination</td>
</tr>
</tbody>
</table>

**Examples:**

```java
/*
 * It modifies the card previously created in the class "Employee"
 * by setting the following attributes:
 *  "Phone" = "754-3010"
 *  "Email" = "j.hetfield@somemail.com"
 */
cmdb.existingCard(cdNewEmployee)
  .with("Phone", "754-3010")
  .withAttribute("Email", "j.hetfield@somemail.com")
  .update();

/*
 * (Logic) delete of the card previously created in the class
 * "Employee"
 */
cmdb.existingCard(cdNewEmployee)
  .delete();

/*
 * Delete of the card attachment that was previously created in the "Employee" class
 */
Iterable<AttachmentDescriptor> attachments =
cmdb.existingCard(cdNewEmployee)
  .attachments()
  .fetch();
```
/*
* Delete of the card attachment that was previously
* created in the “Employee” class
*/
cmdb.existingCard(cdNewEmplyee)
.attachments()
.selectByString(String[]{"attachment-name"})
.delete();

NewProcessInstance

 Builders

newProcessInstance(String className): NewProcessInstance
it creates a new process instance created in CMDBuild for the specified process

 Modifiers

 withDescription(String value): NewProcessInstance
it adds the Description to the new card created in CMDBuild

 with(String name, Object value): NewProcessInstance
it adds the value specified for the specified attribute to the new process created in CMDBuild

 withAttribute(String name, Object value): NewProcessInstance
it adds the value specified for the specified attribute to the new process created in CMDBuild

 Actions

start(): ProcessInstanceDescriptor
it creates the new process in CMDBuild setting the attributes previously defined, and does
not advance

startAndAdvance(): ProcessInstanceDescriptor
it creates the new process in CMDBuild setting the attributes previously defined, and
advances at the following step

Example:
/*
  * Creation of a new card in the “RequestForChange” class
  * having the following attributes
  * "Requester" = “James Hetfield”
  * "RFCExtendedDescription" = “My printer is broken”
  */
pidNewRequestForChange =
  cmdb.newProcessInstance("RequestForChange")
.with(“Requester”, “James Hetfield”)
 .withAttribute(“RFCExtendedDescription”, “My printer is broken”)
 .startAndAdvance();

**ExistingProcessInstance**

| Builders |
|------------------|-----------------|
| existingProcessInstance(String processClassName, int processId): ExistingProcessInstance |
| it creates a process instance existing in the specified process class with the specified Id |

| Modifiers |
|------------------|-----------------|
| with(String name, Object value): ExistingProcessInstance |
| it sets the specified attribute with the specified value for the process instance |
| withAttribute (String name, Object value): ExistingProcessInstance |
| it sets the specified attribute with the specified value for the process instance |
| withDescription(String value): ExistingProcessInstance |
| it sets the specified attribute with the specified value for the process instance |
| attachments(): Attachments |
| it allows you to access the attachments of the process instance |

| Actions |
|------------------|-----------------|
| abort(): void |
| it aborts the process instance |
| advance(): void |
| it advances a process instance |
| resume(): void |
| it resumes the hanging process instance |
| suspend(): void |
| it suspends the open process instance |
| update(): void |
| it updates the process instance |

**Example:**

```java
/*
 * Update of the process instance in the class “Request for change” with Id = pid by editing the requester and advancing the process at the following step
 */
```
cmdb.existingProcessInstance("RequestForChange", pid)
.with("Requester", cdNewEmployee.getId())
.advance();

NewRelation

 Builders
 newRelation(String domainName): ExistingProcessInstance
 it creates a new relation added in the specified Domain of CMDBuild

 Modifiers
 withCard1(String className, int cardId): NewRelation
 it sets the card in the source side of the relation

 withCard2(String className, int cardId): NewRelation
 it sets the card in the target side of the relation

 Actions
 create()
 it creates the new relation in CMDBuild among the Cards indicated in the specified Domain

 Example:
 /*
 * Creation of a new relation in the “AssetAssignee” domain
 * between a card of the selected “Asset” class,
 * through the "Item" Reference attribute, and
 * the card previously created in the “Employee” class
 */
 cmdb.newRelation("AssetAssignee")
 .withCard1("Employee", cdNewEmployee.getId())
 .withCard2("Asset", Item.getId())
 .create();

ExistingRelation

 Builders
 existingRelation(String domainName): ExistingRelation
 it creates an existing relation in the specified Domain of CMDBuild

 Modifiers
 withCard1(String className, int cardId): ExistingRelation
it sets IdClass and l'Objld of the Card from the source side of the relation

```java
withCard2(String className, int cardId): ExistingRelation
```

it sets IdClass and l'Objld of the Card from the target side of the relation

### Actions

**delete()**

- it deletes (logic delete) the relation existing in CMDBuild among the Cards indicated in the specified Domain

**Example:**

```java
/*
 * Delete the relation on the “AssetAssignee” domain
 * among the cards previously indicated
 */

cmdb.existingRelation(“AssetAssignee”)
.withCard1(“Employee”, cdNewEmployee.getId())
.withCard2(“Asset”, Item.getId())
.delete();
```

### QueryClass

**Builders**

**queryClass(String className): QueryClass**

- it creates a query that queries the class specified in CMDBuild

**Modifiers**

**withCode(String value): QueryClass**

- it sets the Card Code for the filter used to query CMDBuild

**withDescription(String value): QueryClass**

- it sets the Card Description for the filter used to query CMDBuild

**with(String name, Object value): QueryClass**

- it sets the value for the specified attribute of the Card for the filter used to query CMDBuild

**withAttribute(String name, Object value): QueryClass**

- it sets the value for the specified attribute of the Card for the filter used to query CMDBuild

### Actions

**fetch(): List<Card>**

- it performs the search query on the specified Class of CMDBuild and returns the list of those Cards that respect the filter previously set
Example:

```java
/*
 * List of the cards of the "Employee" class having
 * the "State" attribute set to 'Active'
 */
Employees = cmdb.queryClass("Employee")
    .with("State", "Active")
    .fetch();
```

CallFunction

<table>
<thead>
<tr>
<th>Builders</th>
</tr>
</thead>
</table>
| ```java
 callFunction(String functionName): CallFunction
```
| it creates a call to a stored procedure previously defined in PostgreSQL |

<table>
<thead>
<tr>
<th>Modifiers</th>
</tr>
</thead>
</table>
| ```java
 with(String name, Object value): CallFunction
```
| it sets the value of the input parameter specified for the stored procedure |

<table>
<thead>
<tr>
<th>Actions</th>
</tr>
</thead>
</table>
| ```java
 execute(): Map<String, String>
```
| it performs the stored procedure and returns the list of the output parameters with the related values |

Example:

```java
/*
 * Call of the stored PostgreSQL procedure
 * "cmwf_getImpact"(IN "DeliveryDate" date, IN "Cost" integer,
 * OUT "Impact" character varying)
 * that computes the impact level (attribute of
 * "Impact" process) of an activity on a scale of "High",
 * "Medium" and "Low", given in input the expected delivery
 * date (process attribute "ExpectedDeliveryDate") and
 * the price (attribute "ManHoursCost") expressed in hour/employee
 */
spResultSet = cmdb.callFunction("cmwf_getImpact")
    .with("DeliveryDate", ExpectedDeliveryDate.getTime())
    .with("Cost", ManHoursCost)
    .execute();
Impact = spResultSet.get("Impact")
```
Note: SQL functions - which should be called - must be defined according to CMDBuild standards. For their definition see the Administrator Manual, section Cart TAB, paragraph “Definition of the data source (PostgreSQL function)

**QueryRelations**

**Builders**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>queryRelations(CardDescriptor cardDescriptor): ActiveQueryRelations</code></td>
<td>it creates a query to ask CMDBuild the Cards related to the specified one</td>
</tr>
<tr>
<td><code>queryRelations(String className, int id): ActiveQueryRelations</code></td>
<td>it creates a query to ask CMDBuild the Cards related to that specified by className and id</td>
</tr>
</tbody>
</table>

**Modifiers**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>withDomain(String domainName): ActiveQueryRelations</code></td>
<td>it sets the Domain to perform the query</td>
</tr>
</tbody>
</table>

**Actions**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fetch(): List&lt;CardDescriptor&gt;</code></td>
<td>it performs the query on CMDBuild using the parameters previously defined, it returns the list of the linked Cards</td>
</tr>
</tbody>
</table>

**Example:**

```java
/*
 * List of “Assets” linked to the “Employee” card indicated
 * by the CardDescriptor cdNewEmployee previously created,
 * through the relation on the domain “AssetAssignee”
 */
assets = cmdb.queryRelation(cdNewEmployee)
  .withDomain(“AssetAssignee”)
  .fetch();
```

**CreateReport**

**Builders**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>createReport(String title, String format): CreateReport</code></td>
<td>it creates the Report in the specified format (pdf, csv) with the specified Title</td>
</tr>
</tbody>
</table>

**Modifiers**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>with(String name, Object value): CreateReport</code></td>
<td>it sets the input parameter value specified for the Report</td>
</tr>
</tbody>
</table>
**Actions**

`download()`: DownloadedReport

It generates the indicated Report using the parameters previously defined

**Example:**

```java
/*
 * It generated the Report “DismissedAssets” which shows the list
 * of the abandoned Assets
 */
newReport = cmdb.createReport(“Assigned assets to”).download();
```

**NewMail Builders**

`newMail()`: NewMail

It creates a new e-mail to send

**Modifiers**

`withFrom(String from)`: NewMail

It sets the sender of the e-mail to send

`withTo(String to)`: NewMail

It sets the recipients of the e-mail to send

`withCc(String cc)`: NewMail

It sets the carbon copy recipients of the e-mail to send

`withBcc(String bcc)`: NewMail

It sets the blind carbon copy recipients of the e-mail to send

`withSubject(String subject)`: NewMail

It sets the subject of the e-mail to send

`withContent(String content)`: NewMail

It sets the text of the e-mail to send

`withContentType(String contentType)`: NewMail

It sets the content MimeType of the e-mail to send, the allowed values are “text/html” or “text/plain”. If not otherwise specified, the default value is “text/plain”

`withAttachment(URL url)`: NewMail

It sets the url of a document to enclose to the e-mail

`withAsynchronousSend(bool boolean)`: NewMail

It sends the e-mail asynchronously in spite of the script; in this way any timeout problem will be avoided, but you will not be able to intervene in case of error by sending the e-mail
### Actions

<table>
<thead>
<tr>
<th>send()</th>
</tr>
</thead>
<tbody>
<tr>
<td>it performs the e-mail sending using the previously defined statements</td>
</tr>
</tbody>
</table>

### Example:

```java
/*
 * Send a new email
 */
cmdb.newMail()
    .withFrom("fromaddress@somemail.com")
    .withTo("toaddress@somemail.com")
    .withCc("ccaddress@somemail.com")
    .withSubject("Mail subject")
    .withContent("Mail content")
    .send();
```

### NewMailQueue

**Builders**

<table>
<thead>
<tr>
<th>newMailQueue(): NewMailQueue</th>
</tr>
</thead>
<tbody>
<tr>
<td>it creates a new email queue</td>
</tr>
</tbody>
</table>

**Methods**

<table>
<thead>
<tr>
<th>newMail(): QueueableNewMail</th>
</tr>
</thead>
<tbody>
<tr>
<td>it adds a new email to the queue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sendAll(): void</th>
</tr>
</thead>
<tbody>
<tr>
<td>it sends all emails from the queue</td>
</tr>
</tbody>
</table>

```java
/*
 * Send a new email
 */
cmdb.newMailQueue()
    .newMail()
    .withFrom("fromaddress@somemail.com")
    .withTo("toaddress@somemail.com")
    .withCc("ccaddress@somemail.com")
    .withSubject("Mail subject")
    .withContent("Mail content")
    .add()
```
Methods for types conversion

**ReferenceType**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>referenceTypeFrom(Card card): ReferenceType</td>
<td>it returns the ReferenceType item related to the specified Card</td>
</tr>
<tr>
<td>referenceTypeFrom(CardDescriptor cardDescriptor): ReferenceType</td>
<td>it returns the ReferenceType item related to the specified CardDescriptor</td>
</tr>
<tr>
<td>referenceTypeFrom(int id): ReferenceType</td>
<td>it returns the ReferenceType item related to the card with the specified Id</td>
</tr>
</tbody>
</table>

**Example:**

```java
/*
 * Set the “Requester” process attribute Reference type, given the “cdNewEmployee” CardDescriptor previously created
 */
Requester = cmdb.referenceTypeFrom(cdNewEmployee);
```

**LookupType**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>selectLookupById(int id): LookupType</td>
<td>it returns the LookupType item with the specified Id</td>
</tr>
<tr>
<td>selectLookupByCode(String type, String code): LookupType</td>
<td>it returns the LookupType item with specified Type and Code</td>
</tr>
<tr>
<td>selectLookupByDescription(String type, String description): LookupType</td>
<td>it returns the LookupType item with specified Type and Description</td>
</tr>
</tbody>
</table>

**Example:**

```java
/* Set the "State" process attribute Lookup type having:
 * "Type" = "Employee state"
 * "Code" = "ACTIVE"
 */
State = cmdb.selectLookupByCode("Employee state", "ACTIVE");
```

**CardDescriptor**

<table>
<thead>
<tr>
<th>Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cardDescriptorFrom(ReferenceType reference): CardDescriptor</td>
<td>it returns the CardDescriptor of the specified card through the specified ReferenceType item</td>
</tr>
</tbody>
</table>

.cmdb.sendAll();
Example:

```
/*
 * Get the CardDescriptor related to the "Requester"
 * process attribute Reference type
 */

cdSelectedEmployee = cmdb.cardDescriptorFrom(Requester);
```

Card

<table>
<thead>
<tr>
<th>Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>cardFrom(ReferenceType reference): Card</td>
</tr>
<tr>
<td>it returns the Card item of the specified card through the specified ReferenceType item</td>
</tr>
</tbody>
</table>

Example:

```
/*
 * Get the complete Card related to the "Requester"
 * process attribute Reference type
 */

selectedEmployee = cmdb.cardFrom(Requester);
```
Appendix: Documentation to use TWS 2.3

Foreword

In appendix you will find the specified technical documentation of the workflow system used until CMDBuild 1.5, whose compatibility is maintained also in CMDBuild 2.0; as soon as possible it will be discarded.

We must remember that in CMDBuild 2.0 there is the possibility to work - alternatively - both with Together Workflow Server 2.3 (the version used until CMDBuild 1.5, based on XPDL 1.0) and with the new version Together Workflow Server 4.4 (based on XPDL 2.0).

It is advisable to migrate in a short time, since that double compatibility will be maintained for a limited period.

Automatic methods used in the workflow

In order to use Together Workflow Server 2.3 (passed by the system based on Together Workflow Server 4.4), CMDBuild provides some methods ("tools"), which can be used inside the "tool activities" (automatic activities) in order to perform the various operation typologies:

- methods for the manipulation of the variables: conversion among data typologies, strings connection, etc.
- methods for the flow control: iterator, process suspension, process reboot
- access methods to CMDB: create a new card, read or change attribute, create relation, etc.
- external methods: sending e-mails, reading system time, etc.

Methods for the manipulation of the variables

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
<th>Input parameters</th>
<th>Output parameters</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>addDays</td>
<td>It adds to the specified date the indicated number of days</td>
<td>InputDate date days integer</td>
<td>OutputDate date</td>
<td></td>
</tr>
<tr>
<td>boolToString</td>
<td>It converts a boolean variable to a string</td>
<td>InputBool boolean</td>
<td>OutputString string</td>
<td></td>
</tr>
<tr>
<td>boolCopy</td>
<td>It copies the value of a boolean variable into another boolean one</td>
<td>From boolean</td>
<td>To boolean</td>
<td></td>
</tr>
<tr>
<td>clearIterator</td>
<td>It resets the iterator</td>
<td>RefArray references HasNext boolean CurrentIndex integer</td>
<td>RefArray is set to null, CurrentIndex is set to 0 and HasNext to false</td>
<td></td>
</tr>
<tr>
<td>clearLookup</td>
<td>It resets the value</td>
<td>Lookup lookup</td>
<td>The value of the &quot;Id&quot;</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>clearReference</td>
<td>It resets the value of a Reference variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ref reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The value of the &quot;Id&quot; attribute is set to -1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>concat / concat3 / ... / concat8</td>
<td>It concatenates two or more strings</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>InputString1 string InputString2 string ... InputStringn string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OutputString string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>createReferenceObj</td>
<td>It creates a reference variable and initializes it</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ClassName string ObjId integer Description string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OutRef reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The variable is initialized with the values read by the attributes &quot;ClassName&quot;, &quot;ObjId&quot; and &quot;Description&quot; of the specified card</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dateToString</td>
<td>It converts a date variable into a string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>InputDate date</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OutputString string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>floatToString</td>
<td>It converts a float variable into a string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>InputFloat float</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OutputString string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>floatCopy</td>
<td>It copies the value of a float variable into another float variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>From float</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>To float</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getReferenceId</td>
<td>It extracts the &quot;Id&quot; attribute from a reference variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ref reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CardId integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getReferenceClassId</td>
<td>It extracts the &quot;ClassId&quot; attribute from a reference variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ref reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ClassId integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getLookupDescription</td>
<td>It extracts the &quot;Description&quot; attribute from a lookup variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lookup lookup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Description string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getLookupId</td>
<td>It extracts the &quot;Id&quot; attribute from a lookup variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lookup lookup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Id Integer</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getLookupCode</td>
<td>It extracts the &quot;Code&quot; attribute from a lookup variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lookup lookup</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Code String</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getReferenceDescription</td>
<td>It extracts the &quot;Description&quot; attribute from a reference variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ref reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Description string</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getReferenceFromArray</td>
<td>It extracts the specified</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RefArray references</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>OutRef reference</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If the array is null or the index is higher than its</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Input parameters</td>
<td>Output parameters</td>
<td>Notes</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------</td>
<td>------------------</td>
<td>-------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>intToString</td>
<td>It converts an integer variable into a string</td>
<td>InputInt integer</td>
<td>OutputString string</td>
<td></td>
</tr>
<tr>
<td>intCopy</td>
<td>It copies the value of an integer variable into another integer variable</td>
<td>From integer</td>
<td>To integer</td>
<td></td>
</tr>
<tr>
<td>lookupToString</td>
<td>It converts the &quot;Id&quot; field of the lookup variable into a string</td>
<td>InputLookup lookup</td>
<td>OutputString string</td>
<td></td>
</tr>
<tr>
<td>nextInt</td>
<td>It increases the specified integer variable</td>
<td>InputInt integer</td>
<td>InputInt integer</td>
<td></td>
</tr>
<tr>
<td>referenceToString</td>
<td>It converts the “Id” field of the reference variable into a string</td>
<td>InputReference reference</td>
<td>OutputString string</td>
<td></td>
</tr>
<tr>
<td>stringToDate</td>
<td>It converts a string variable into a date</td>
<td>InputString string</td>
<td>OutputDate date</td>
<td>It accepts as input formats YY/MM/dd or YY/mm/dd HH:mm:ss</td>
</tr>
<tr>
<td>stringCopy</td>
<td>It copies the value of a string variable into another string variable</td>
<td>From string</td>
<td>To string</td>
<td></td>
</tr>
<tr>
<td>dateCopy</td>
<td>It copies the value of a date variable into another date variable</td>
<td>From date</td>
<td>To date</td>
<td></td>
</tr>
<tr>
<td>stringToBool</td>
<td>It converts a string variable into a boolean value</td>
<td>From string</td>
<td>To boolean</td>
<td>It accepts as input the true or false strings</td>
</tr>
<tr>
<td>stringToInt</td>
<td>It converts a string variable into an integer</td>
<td>From string</td>
<td>To integer</td>
<td>In input it accepts the representation of an integer number in the shape of string</td>
</tr>
<tr>
<td>stringToFloat</td>
<td>It converts a string variable into a float</td>
<td>From string</td>
<td>To float</td>
<td>In input it accepts the representation of a float in the shape of string</td>
</tr>
</tbody>
</table>

**Methods for the flow control**

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
<th>Input parameters</th>
<th>Output parameters</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextRef</td>
<td>It increases the RefArray</td>
<td>HasNext</td>
<td>RefArray is a reference</td>
<td></td>
</tr>
</tbody>
</table>
iterator on a reference array

references

CurrentIndex

integer

boolean

CurrentIndex

integer

CurrentValue

reference

array, CurrentValue is the reference corresponding to the current index

resetIterator

It resets the iterators

RefArray

references

HasNext

boolean

CurrentIndex

integer

CurrentIndex is set to 0, HasNext is true if the array is not empty

resumeProcess

It reboots the specified process

ProcessInstanceId

string

Complete

integer

The status of the specified process must be “Suspended”
If “Complete” takes on the value 1, the process steps forward

suspendProcess

It suspends the specified process

ProcessInstanceId

string

The constant “CURRENT” can be used to indicate the current process
The process is suspended immediately before the following manual activity

voidApp

Null tool

Access methods to CMDB

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
<th>Input parameters</th>
<th>Output parameters</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>createCard</td>
<td>It creates a new card and returns the &quot;Id&quot;</td>
<td>ClassName string</td>
<td>CardId integer</td>
<td>The method sets only the basic attributes &quot;Code&quot; and &quot;Description&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CardCode string</td>
<td></td>
<td>In order to set the other ones, you have to use the updateAttribute tool or define a createCard metatool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CardDescription string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>createCardRef</td>
<td>It creates a new card and returns the reference</td>
<td>ClassName string</td>
<td>CardReference reference</td>
<td>The method sets only the basic attributes &quot;Code&quot; and &quot;Description&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CardCode string</td>
<td></td>
<td>In order to set the other ones, you have to use the updateAttribute tool or define a createCard metatool</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CardDescription string</td>
<td></td>
<td></td>
</tr>
<tr>
<td>createRelation</td>
<td>It creates a relation between two cards</td>
<td>DomainName string</td>
<td>Done boolean</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ClassName1 string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ClassName2 string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ObjId1 integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ObjId2 integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Parameters</td>
<td>Return Type</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>createRelation1Ref</td>
<td>It creates a relation between two cards, the first of them is specified by reference</td>
<td>DomainName string, ObjReference1 reference, ClassName2 string, ObjId2 integer</td>
<td>Done boolean</td>
<td></td>
</tr>
<tr>
<td>createRelation2Ref</td>
<td>It creates a relation between two cards, the second of them is indicated by reference</td>
<td>DomainName string, ClassName1 string, ObjId1 integer, ObjReference2 reference</td>
<td>Done boolean</td>
<td></td>
</tr>
<tr>
<td>createRelationRefs</td>
<td>It creates a relation between two cards, both specified by reference</td>
<td>DomainName string, ObjReference1 reference, ObjReference2 reference</td>
<td>Done boolean</td>
<td></td>
</tr>
<tr>
<td>deleteRelation</td>
<td>It removes a relation between two cards</td>
<td>DomainName string, ClassName1 string, ClassName2 string, ObjId1 integer, ObjId2 integer</td>
<td>Done boolean</td>
<td></td>
</tr>
<tr>
<td>deleteRelationByReference</td>
<td>It deletes a relation between two cards, both specified by reference</td>
<td>DomainName string, ObjReference1 reference, ObjReference2 reference</td>
<td>Done boolean</td>
<td></td>
</tr>
<tr>
<td>selectAttribute</td>
<td>It reads an attribute of the specified card</td>
<td>ClassName string, AttributeName string, ObjId integer</td>
<td>AttributeValue string, The returned value is always represented by a string</td>
<td></td>
</tr>
<tr>
<td>selectAttributeFromReference</td>
<td>It reads an attribute of the specified card, specified by reference</td>
<td>ObjReference reference, AttributeName string</td>
<td>AttributeValue string, The returned value is always represented by a string</td>
<td></td>
</tr>
<tr>
<td>selectLookup</td>
<td>It reads the description of a Lookup entry, specified by type and &quot;Id&quot;</td>
<td>Type string, LookupId integer</td>
<td>LookupDescription string</td>
<td></td>
</tr>
<tr>
<td>selectLookupById</td>
<td>It returns a Lookup entry, specified by &quot;Id&quot;</td>
<td>LookupId integer</td>
<td>Lookup lookup</td>
<td></td>
</tr>
<tr>
<td>selectLookupByTypeDesc</td>
<td>It returns a Lookup entry, specified by type and description</td>
<td>Type string, Description string</td>
<td>Lookup lookup</td>
<td></td>
</tr>
<tr>
<td>selectLookupByType</td>
<td>It returns a Lookup entry</td>
<td>Type string</td>
<td>Lookup lookup</td>
<td></td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
<td>Parameters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| `selectReferenceByCode`                     | It returns a reference item corresponding to the card specified by code     | ClassName `string`  
Code `string`  
OutRef `reference` |
| `selectReferenceByCustomAttribute`          | It returns a reference item corresponding to the indicated card through a generic attribute | ClassName `string`  
AttributeName `string`  
AttributeValue `string`  
OutRef `reference` |
| `selectReferenceByReference`                | It returns a reference item corresponding to a reference attribute existing in the card specified by reference | ObjReference `reference`  
AttributeName `string`  
OutRef `reference` |
| `selectRelations`                           | It returns on a specific domain an array of references corresponding to the cards related to the one given by the specified id and class | ClassName `string`  
CardId `integer`  
DomainName `string`  
RefArray `relations` |
| `selectRelationsByReference`                | It returns on a specific domain an array of references corresponding to the cards related to the one given by the specified reference | ClassName `string`  
CardId `integer`  
DomainName `string`  
RefArray `relations` |
| `updateAttribute`                           | It modifies a card                                                          | ClassName `string`  
AttributeName `string`  
ObjId `integer`  
AttributeValue `string`  
Done `boolean`  
The method edits only the specified attribute. In order to edit more attributes, you have to define an `updateCard` metatool |
| `updateAttributeRef`                        | It edits a card specified by reference                                       | ObjRef `reference`  
AttributeName `string`  
AttributeValue `string`  
Done `boolean` |
## External methods

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
<th>Input parameters</th>
<th>Output parameters</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCurrentTimestamp</td>
<td>It returns the system date and time</td>
<td></td>
<td>TheDate date</td>
<td></td>
</tr>
<tr>
<td>getCurrentGroupReference</td>
<td>It returns a reference variable corresponding to the current user group</td>
<td></td>
<td>GroupRef reference</td>
<td>The returned item corresponds to the current group card in the CMDB &quot;Role&quot; table</td>
</tr>
<tr>
<td>getCurrentUserReference</td>
<td>It returns a reference variable corresponding to the current user</td>
<td></td>
<td>UserRef reference</td>
<td>The returned item corresponds to the current user card in the CMDB &quot;Role&quot; table</td>
</tr>
<tr>
<td>getReportFullUrl</td>
<td>It returns the link to the report created with the extended attribute createReport</td>
<td>ReportUrl string</td>
<td>ReportUrl string</td>
<td></td>
</tr>
<tr>
<td>sendMail</td>
<td>It sends an e-mail</td>
<td>FromAddresses string</td>
<td></td>
<td>The tool provides that Shark parameters related to the e-mail sending are correctly configured Parameters From, To and Attach can include more values concatenated with &quot;,&quot; Parameters CCAddresses, BCCAddresses and UrlAttachments can be set by an empty string MimeType can take the values &quot;text/html&quot; or &quot;text/plain&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ToAddresses string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CCAddresses string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>BCCAddresses string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Subject string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Content string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>UrlAttachments string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>MimeType string</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Template automatic methods usable in the workflow

In order to use Together Workflow Server 2.3 (passed by the system based on Together Workflow Server 4.4), CMDBuild provides some templates of automatic methods (meta-tools), used for the definition of tools.

For the creation of new “tools”, custom CMDBuild provides the following steps:

- creation of a new “Application” with TWE Together Workflow Editor 4.4 (you can access the list from the process features), with the proper button “Create new element”
- the completion of the “Application” definition by clicking on the new row added to the list and setting the following parameters:
  - Id = name for the new “tool”
  - Name = you can set the same value chosen in the previous field
  - Formal parameters = adding as many input and output parameters as provided in the tool (as you can see in the following table)
  - Extended attribute “ToolAgentClass”
  - further specific extended attributes in the meta-tool (as you can see in the following table)

<table>
<thead>
<tr>
<th>Template type</th>
<th>Description</th>
<th>Input parameters</th>
<th>Output parameters</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>createCard</td>
<td>Card creation in the CMDB</td>
<td>Attributes list set in the new card or ClassName string List of input parameters provided by the function</td>
<td>CardReference reference</td>
<td>It returns the id of the created card The second specification of the input parameters can be used if you exclude ClassName from the external attribute list (see the following table)</td>
</tr>
<tr>
<td>createReport</td>
<td>Running a report</td>
<td>List of input parameters provided by the report</td>
<td>ReportURL string</td>
<td>The returned URL can be used to enclose the report to an e-mail with the tool sendMail</td>
</tr>
<tr>
<td>executeFunction</td>
<td>Execute PostgreSQL functions</td>
<td>List of input parameters provided by the function</td>
<td>List of output parameters provided by the function</td>
<td>There must be at least one input parameter and one output parameter, even if they are fake</td>
</tr>
<tr>
<td>startProcess</td>
<td>Instance initiation of another process</td>
<td>Attributes list set during the process initiation</td>
<td>ProcessInstanceId string</td>
<td>It returns the process instance name (string type)</td>
</tr>
<tr>
<td>updateCard</td>
<td>Card update in the CMDB</td>
<td>ClassName string ObjId integer Attributes list updated in the card or</td>
<td>Done boolean</td>
<td></td>
</tr>
</tbody>
</table>
For a better readability, the following table shows separately the ToolAgent indication and other possible attributes that must be specified as TWE in the metatool definition.
All values of the attributes are string type.

<table>
<thead>
<tr>
<th>Template type</th>
<th>Metatool attribute</th>
<th>Metatool value</th>
</tr>
</thead>
<tbody>
<tr>
<td>createCard</td>
<td>ToolAgentClass</td>
<td>org.cmdbuild.shark.toolagent.CreateCardToolAgent</td>
</tr>
<tr>
<td></td>
<td>ClassName</td>
<td>[Class Name]</td>
</tr>
<tr>
<td>createReport</td>
<td>ToolAgentClass</td>
<td>org.cmdbuild.shark.toolagent.CreateReportToolAgent</td>
</tr>
<tr>
<td></td>
<td>Type</td>
<td>custom</td>
</tr>
<tr>
<td></td>
<td>Code</td>
<td>[Report code]</td>
</tr>
<tr>
<td></td>
<td>Format</td>
<td>pdf or csv</td>
</tr>
<tr>
<td>executeFunction</td>
<td>ToolAgentClass</td>
<td>org.cmdbuild.shark.toolagent.ExecuteStoredProcedureToolAgent</td>
</tr>
<tr>
<td></td>
<td>Procedure or CursorProcedure</td>
<td>[PostgreSQL function name with return single value]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[PostgreSQL function name with return multiple value]</td>
</tr>
<tr>
<td>startProcess</td>
<td>ToolAgentClass</td>
<td>org.cmdbuild.shark.toolagent.ProcessStartToolAgent</td>
</tr>
<tr>
<td></td>
<td>ProcessClass</td>
<td>[Class Name]</td>
</tr>
<tr>
<td></td>
<td>Complete</td>
<td>1 (to advance the process to the following activity)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or 0 (to stop the process on the first activity)</td>
</tr>
<tr>
<td>updateCard</td>
<td>ToolAgentClass</td>
<td>org.cmdbuild.shark.toolagent.UpdateAttributeToolAgent</td>
</tr>
<tr>
<td>updateProcess</td>
<td>ToolAgentClass</td>
<td>org.cmdbuild.shark.toolagent.ProcessUpdateToolAgent</td>
</tr>
<tr>
<td></td>
<td>ProcessClass</td>
<td>[Class Name]</td>
</tr>
<tr>
<td></td>
<td>Complete</td>
<td>1 (to advance the process to the following activity)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or 0 (to stop the process on the first activity)</td>
</tr>
<tr>
<td></td>
<td>If the process is &quot;Suspended&quot;, the resumeProcess method must be carried out in advance.</td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX: Glossary

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.

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

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

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

See also: Process

ATTRIBUTE
The term refers to an attribute of a CMDBuild class.
CMDBuild allows you to create new attributes (in classes and domains) or edit existing ones.

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

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

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

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

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

See also: GIS

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

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

See also: Configuration

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

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

See also: Card, Attribute

**CONFIGURATION**

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

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

See also: CI, ITIL

**DASHBOARD**

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

See also: Report

**DATABASE**

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

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

**DOMAIN**

A domain is a relation between two classes.

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

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

It is possible to define custom attributes for each domain.

See also: Class, Relation

**DATA FILTER**

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

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

See also: Class, View

**GIS**

A GIS is a system able to produce, manage and analyse spatial data by associating geographic elements to one or more alphanumeric descriptions.
GIS functionalities in CMDBuild allow you to create geometric attributes (in addition to standard attributes) that represent, on plans / maps, markers position (assets), polylines (cable lines) and polygons (floors, rooms, etc.).

See also: BIM

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

See also: Mobile, Webservice

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

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

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

See also: Configuration

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

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

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

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

See also: GUI Framework, Webservice

PROCESS
The term "process" (or workflow) refers to a sequence of steps that realize an action. Each process will take place on specific assets and will be performed by specific users. A process is activated by starting a new process (filling related form) and ends when the last workflow step is executed.

See also: Workflow step

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

CMDBuild allows users, through the Management Module, to define new relations among the
cards stored in the database.
See also: Class, Domain

REPORT
The term refers to a document (PDF or CSV) containing information extracted from one or more classes and related domains.
CMDBuild users run reports by using the Management Module; reports definitions are stored in the database.
See also: Class, Domain, Database

CARD
The term "card" refers to an element stored in a class.
A card is defined by a set of values, i.e. the attributes defined for its class.
CMDBuild users, through the Management Module, are able to store new cards and update / delete existing ones.
Card information is stored in the database and, more exactly, in the table/columns created for that class (Administration Module).
See also: Class, Attribute

SUPERCLASS
A superclass is an abstract class used to define attributes shared between classes. From the abstract class you can derive real classes that contain data and include both shared attributes (specified in the superclass) and specific subclass attributes.
For example, you can define the superclass "Computer" with some basic attributes (RAM, HD, etc.) and then define derived subclasses "Desktop", "Notebook", "Server", each one with some specific attributes.
See also: Class, Attribute

ATTRIBUTE TYPE
Each attribute has a data type that represents attribute information and management.
The attribute type is defined using the Administration Module and can be modified within some limitations, depending on the data already stored in the system.
CMDBuild manages the following attribute types: "Boolean", "Date", "Decimal", "Double", "Inet" (IP address), "Integer", "Lookup" (lists set in "Settings" / "LookUp"), "Reference" (foreign key), "String", "Text", "Timestamp".
See also: Attribute

VIEW
A view not only includes the whole content of a CMDB class, it is a group of cards defined in a logical way.
In particular, a view can be defined in CMDBuild by applying a filter to a class (so it will contain a reduced set of the same rows) or specifying an SQL function which extracts attributes from one or more related classes.
The first view type maintains all functionalities available for a class, the second one allows the sole display and search with fast filter.
See also: Class, Filter

**WEBSERVICE**

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

**WIDGET**

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