

Joomla/Mambo Community Builder

Version 1.4

Application Programming Interface Guide



document version 1.4 [build 006] – 8.March.2011

Copyright 2004-2011 Joomla!polis.com

No portions of this manual may be reproduced or redistributed

Table of Contents

- 1 Introduction and Background Information6
 - 1.1 A few words about CB and CB Plugins6
 - 1.2 Useful URLs to bookmark.....7
 - 1.3 Document Outline.....8
 - 1.4 How to best use this Material8
 - 1.5 Acknowledgements, Credits and Copyrights8
- 2 CB Architecture Overview.....9
 - 2.1 Key Items9
 - 2.2 File System After Installation 10
 - 2.3 Database After Installation34
- 3 CB Plugins.....42
 - 3.1 Overview42
 - 3.2 Plug-in types.....42
 - 3.3 Plug-in naming.....42
 - 3.4 Installation Process43
 - 3.5 Structure of plug-ins43
 - 3.6 XML file43
 - 3.6.1 Header44
 - 3.6.2 Files.....45
 - 3.6.3 Plug-in Parameters.....47
 - 3.6.4 Tabs48
 - 3.6.5 Database tag description.....50
 - 3.6.6 SQL queries (install and un-install).....51
 - 3.6.7 Install code (also un-install code)51
 - 3.7 Language plug-ins52
 - 3.8 User plug-ins53
 - 3.8.1 Parameter passing53
 - 3.8.2 Error Management54
 - 3.8.3 Objects55

Community Builder 1.4 API Guide - Table of Contents

| | | |
|--------|---|----|
| 3.8.4 | Tabs | 55 |
| 3.8.5 | CB Events | 57 |
| 3.8.6 | Generating HTML output..... | 60 |
| 3.8.7 | User profile | 61 |
| 3.8.8 | User edit..... | 61 |
| 3.8.9 | Registration | 61 |
| 3.8.10 | Field Validation..... | 61 |
| 3.9 | Special user plug-ins | 63 |
| 3.9.1 | PMS | 63 |
| 3.9.2 | Menu | 64 |
| 3.10 | CB API..... | 64 |
| 3.10.1 | Menus | 64 |
| 3.10.2 | Status display | 65 |
| 3.10.3 | Forms | 66 |
| 3.10.4 | Generic list support | 68 |
| 3.10.5 | User lists support..... | 72 |
| 3.10.6 | User search support..... | 72 |
| 3.10.7 | Language support for plug-ins..... | 72 |
| 3.11 | Integrating with other components..... | 72 |
| 3.11.1 | Talk with the others | 72 |
| 3.11.2 | Preferred way: clean API..... | 72 |
| 3.11.3 | Other way: through SQL tables | 72 |
| 4 | Reference Snippets | 73 |
| 4.1 | Cheat sheet | 73 |
| 4.2 | Including CB Framework | 74 |
| 4.2.1 | Logging changes: | 76 |
| 4.2.2 | PMS to user link: | 76 |
| 4.2.3 | Generate HTML code to display avatar of a user: | 77 |
| 5 | Conclusions | 78 |

Table of Figures

Figure 1: CB Architectural Diagram 10

Figure 2: Contents of administrator → components → com_comprofiler 11

Figure 3: Contents of components → com_comprofiler 11

Figure 4: Contents of components → com_comprofiler → plugins 11

Figure 5: Contents of components → com_comprofiler → plugins → user 12

Figure 6: Addition of comprofiler folder in images root folder 12

Figure 7: Items in components → com_comprofiler folder 13

Figure 8: Items in components → com_comprofiler → plugin folder 14

Figure 9: Items in components → com_comprofiler → plugins → language folder 14

Figure 10: Items in components → plugin → languages → default_language 15

Figure 11: Items in components → com_comprofiler → plugin → templates folder 16

Figure 12: Items in components → com_comprofiler → plugin → templates → dark folder 17

Figure 13: Items in components → com_comprofiler → plugin → templates → default folder ... 18

Figure 14: Items in components → com_comprofiler → plugin → templates → luna folder 19

Figure 15: Items in components → com_comprofiler → plugin → templates → osx folder 20

Figure 16: Items in components → com_comprofiler → plugin → templates → webfx folder 21

Figure 17: Items in components → com_comprofiler → plugin → templates → winclassic folder
..... 22

Figure 18: Items in components → com_comprofiler → plugin → user folder 23

Figure 19: Items in the components → com_comprofiler → plugin → user →
plug_cbconnections folder 24

Figure 20: Items in the components → com_comprofiler → plugin → user → plug_cbcore
folder 24

Figure 21: Items in components → com_comprofiler → plugin → user → plug_cbmamblogtab
folder 24

Figure 22: Items in
components → com_comprofiler → plugin → user → plug_cbmamboauthortab folder 25

Figure 23: Items in components → com_comprofiler → plugin → user → plug_cbmenu folder 25

Figure 24: Items in components → com_comprofiler → plugin → user →
plug_cbsimpleboardtab folder 26

Figure 25: Items in components → com_comprofiler → plugin → user → plug_pms_mypmspro
folder 26

| | |
|--|----|
| Figure 26: Items in components→com_comprofiler→plugin→user→ plug_yancintegration folder..... | 27 |
| Figure 27: Items in administrator→components→com_comprofiler folder..... | 28 |
| Figure 28: Items in administrator→components→com_comprofiler→database folder | 29 |
| Figure 29: Items in administrator→components→com_comprofiler→language folder | 29 |
| Figure 30: Items in administrator→components→com_comprofiler→library folder | 30 |
| Figure 31: Items in administrator→components→com_comprofiler→library→cb folder | 31 |
| Figure 32: Items in administrator→components→com_comprofiler→library→cb→sql folder | 32 |
| Figure 33: Items in administrator→components→com_comprofiler→library→cb→xml folder..... | 32 |
| Figure 34: Items in administrator→components→com_comprofiler→library→pcl folder ... | 32 |
| Figure 35: Items in administrator→components→com_comprofiler→library→phpinputfiler folder..... | 32 |
| Figure 36: Items in administrator→components→com_comprofiler→library→phpmailer folder..... | 33 |
| Figure 37: Items in administrator→components→com_comprofiler→xmlcb folder..... | 33 |
| Figure 38: The comprofiler table | 35 |
| Figure 39: The comprofiler_fields table..... | 36 |
| Figure 40: The comprofiler_field_values table | 37 |
| Figure 41: The comprofiler_lists table | 38 |
| Figure 42: The comprofiler_members table | 39 |
| Figure 43: The comprofiler_plugin table..... | 40 |
| Figure 44: The comprofiler_userreports table | 41 |
| Figure 45: The comprofiler_views table | 41 |
| Figure 46 – Header for XML Plug-in Installation File | 44 |
| Figure 47 – PMS plug-in File section for XML Plug-in Installation File..... | 45 |
| Figure 48 - Language plug-in File section for XML Plug-in Installation File | 46 |
| Figure 49 - Params section for XML Plug-in Installation File | 47 |
| Figure 50 - Tabs section for XML Plug-in Installation File..... | 48 |
| Figure 51 – Tabs with Fields section for XML Plug-in Installation File | 49 |
| Figure 52 - Install section for XML Plug-in Installation File | 51 |
| Figure 53 – Language File Example for XML Plug-in Installation File..... | 52 |

1 Introduction and Background Information

This document presents the **A**pplication **P**rogramming **I**nterface (API) associated with the Joomla/Mambo **C**ommunity **B**uilder (CB) suite. The CB API can be used to extend or modify existing functionality of CB, to add new CB features, to create integrations between CB and other third-part extensions and finally to build totally new applications for Joomla/Mambo. The CB API provides the developer with a set of tools that can be used to create a variety of applications.

1.1 A few words about CB and CB Plugins

Community Builder is an open-source GPL v2 licensed script for the Joomla and Mambo **C**ontent **M**anagement **S**ystems (CMS). The CB suite adds a social networking framework to the underlying CMS layer. This framework has many built-in core functions that are key and instrumental in building and managing an online community. Advanced registration workflows, user profiles, moderation mechanisms, user lists and user connections are some of these core functions. The framework also provides the means to extend and add new social networking functionality to the core CB suite. This extensibility is accomplished using CB Plugins that can be created by any developer using the CB API. CB Plugins are distributed as zip-compressed packages and can be installed using the CB Plugin Management installer.

The CB initiative released its first beta for the Mambo CMS back in 2003 and has been identified as its most popular extension ever since. After the Joomla! fork from Mambo, the CB script has followed both code bases and today CB is compatible and works natively on all versions of Joomla (1.0.x and 1.5.x series and Joomla 1.6.0) and Mambo.

The CB project is managed by the CB Team at Joomlapolis and has always been a FREE GPL licensed script. The Joomlapolis community recently (February 2010) celebrated its 375'000 member and is one of the most active communities in the open-source ecosystem. CB is currently powering over 10 million websites!

Joomlapolis hosts a directory of commercial and non-commercial CB related plugins. The CB initiative is still recognized as the most popular extension on the Joomla Extension Directory and has its own CB Extension Specific directory category on the JED.

The CB Project is funded primarily through yearly subscription “Advanced/Professional” memberships. These subscriptions give download privileges (and upgrades) of detailed CB documentation and a rich set of CB Team released CB plugin / add-ons.

CB Team releases are widely recognized as high quality, robust and security-tight deliverables. CB’s quality, popularity, longevity, expandability, its third party developers, and its large community and its overall “fun-to-be-here” community attitude have made CB a truly “great experience”.

1.2 Useful URLs to bookmark

The following locations are of interest to the authors and will be referenced in various ways throughout the remaining sections of this document:

- Joomlapolis website:
<http://www.joomlapolis.com>
- CB Add-ons / Advanced members:
<http://www.joomlapolis.com/cb-solutions/add-ons>
- Incubator project description / Professional members:
<http://www.joomlapolis.com/cb-solutions/incubator>
- Joomlapolis forge:
<http://forge.joomlapolis.com>
- CB forge area:
<http://forge.joomlapolis.com/projects/cb>
- CB Language projects area:
<http://forge.joomlapolis.com/projects/lan-cb>
- Joomlapolis CB Extension Directory
<http://www.joomlapolis.com/cb-solutions/directory>

- CB listing on JED:
<http://extensions.joomla.org/extensions/clients-a-communities/communities/210>
- Joomla CB Specific Extension Directory
<http://extensions.joomla.org/extensions/extension-specific/community-builder-extensions>
- Joomlapolis Advanced/Professional membership support forums:
<http://www.joomlapolis.com/forum/index/152-priority-support-for-advancedprofessional-members>

1.3 Document Outline

The topics discussed cover the CB Architecture, the CB plugin types and reference scripts. It assumes the reader has PHP programming and Javascript / HTML knowledge.

1.4 How to best use this Material

This material should best be used in parallel to reviewing existing CB Team released plugins to see things demonstrated by example.

1.5 Acknowledgements, Credits and Copyrights

Any documentation that does not acknowledge the efforts of the development team and the community involved isn't worth the paper it's printed on (or the KB it occupies). The CB Team would also like to acknowledge its testing team, its languages team and the many third party developers that continuously release CB plugins. Special mention must be given to the members that chose to financially support the project through the membership subscriptions.

The component and modules of the CB suite are released under GPL with the following clause:

| |
|---|
| All copyright statements must be kept. Derivate work must prominently duly acknowledge original work and include visible online links. |
|---|

This document is not released under GPL and no reproduction or distribution may take place without the author's permission.

2 CB Architecture Overview

This section will just be presenting a brief overview of key architectural points associated with CB. It is by no means a definitive guide. The file system and database table presentation can be changed in future CB releases. Nevertheless these concepts and a current snapshot are presented in the following sections in order to give the reader a better understanding about how basic elements of CB are organized and structured.

2.1 Key Items

Key architectural points:

- PHP
- MYSQL
- XML
- jQuery
- triggers

A high level overview of the Community Builder architecture is illustrated in **Figure 1**.

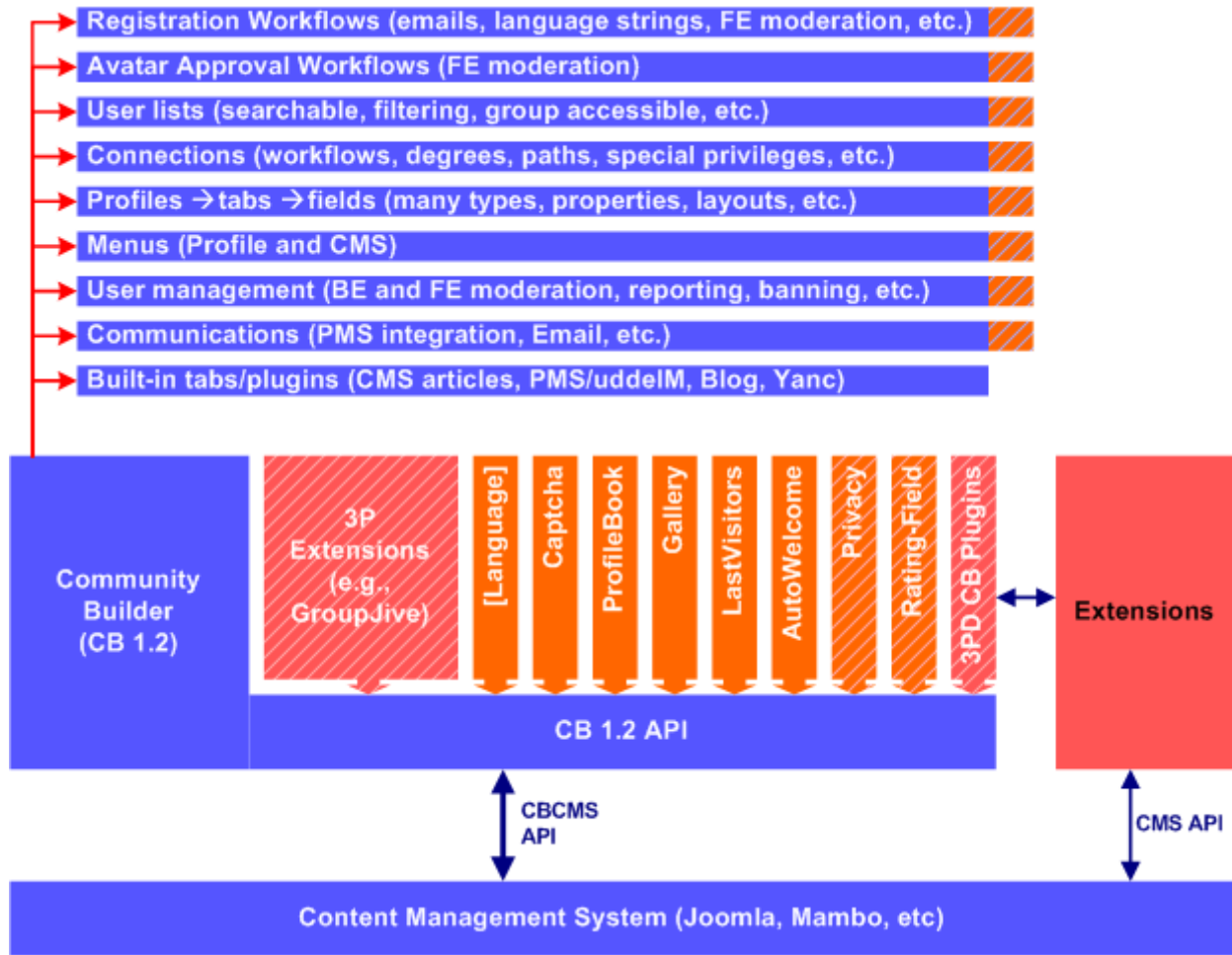


Figure 1: CB Architectural Diagram

The CB API supports development of CB plugins that extends existing CB functionality or adds new functionality to CB. The API has a series of event triggers that are fired when specific events take place in CB.

2.2 File System After Installation

Once you have successfully installed all elements of the CB suite (CB 1.4) on a Joomla 1.5.x or 1.6.0 website environment, you will see the file structure modifications/additions illustrated in **Figure 2**, **Figure 3**, **Figure 4**, **Figure 5** and **Figure 6**.

This file system structure is of course subject to change and is only included for completeness reasons. You should not base your development efforts on this structure.

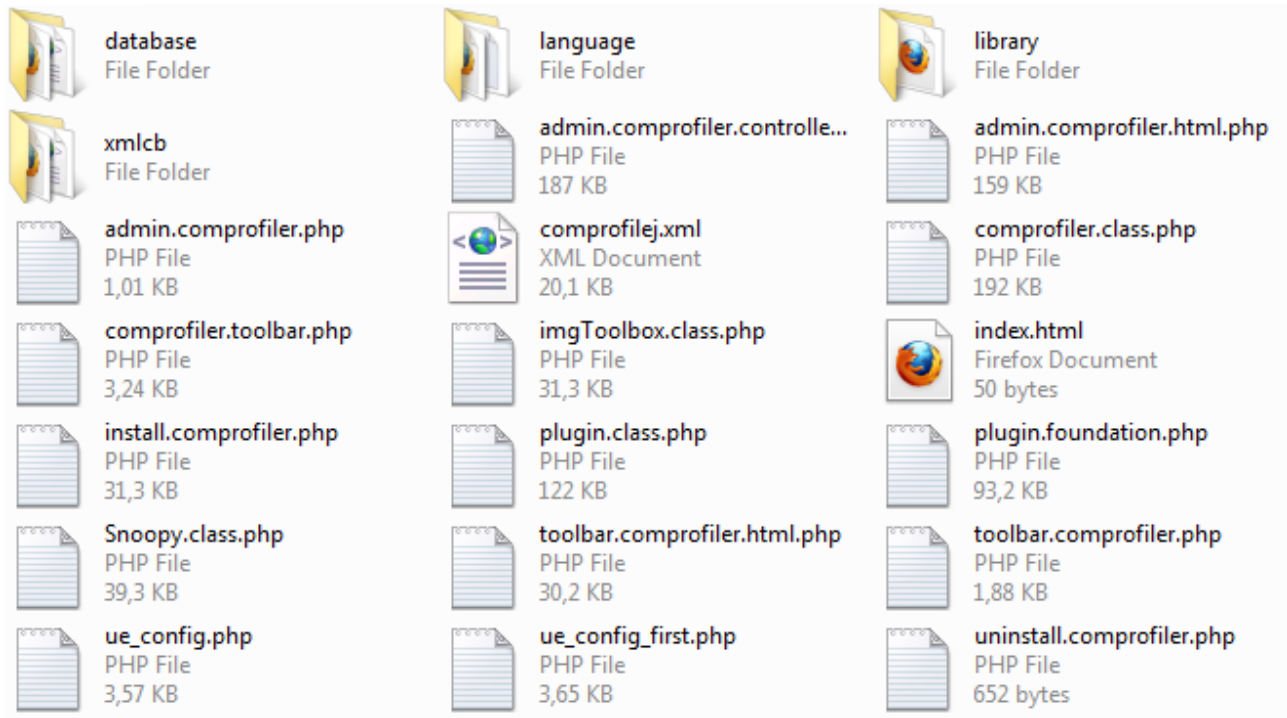


Figure 2: Contents of administrator → components → com_comprofiler

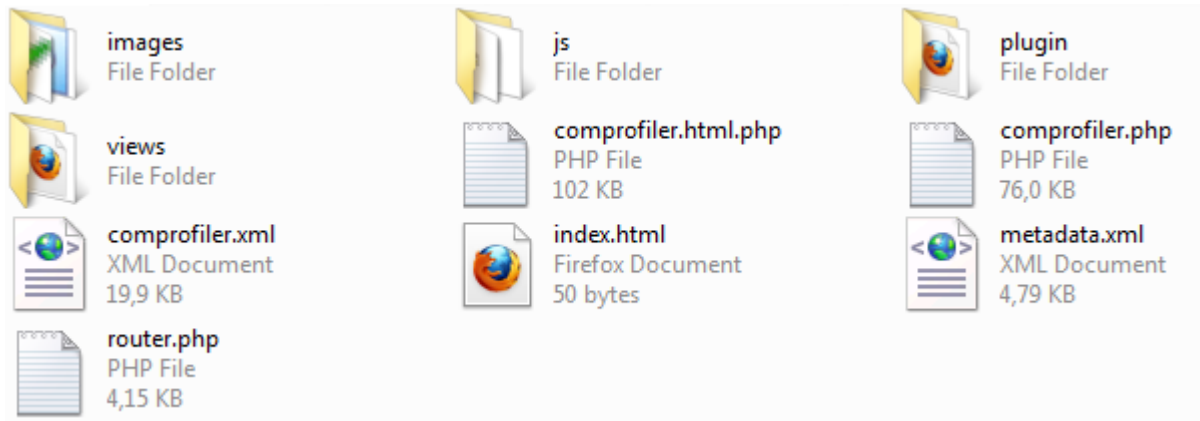


Figure 3: Contents of components → com_comprofiler

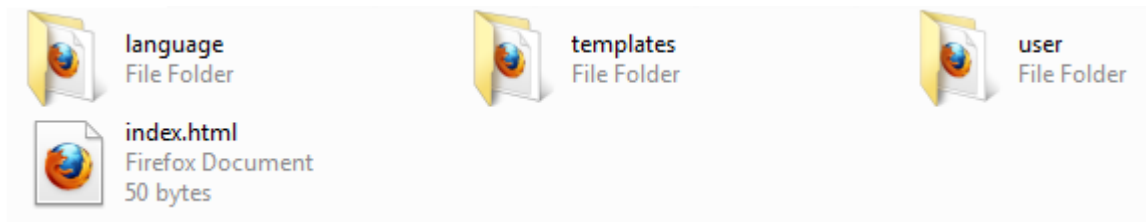


Figure 4: Contents of components → com_comprofiler → plugins

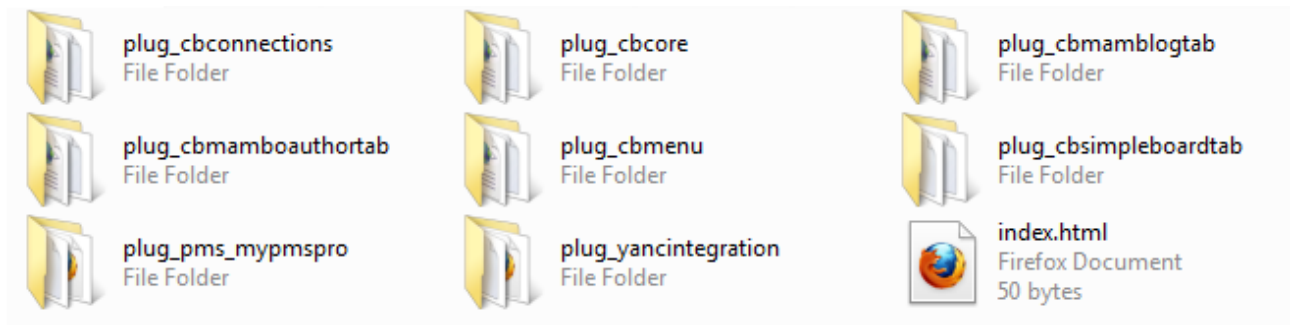


Figure 5: Contents of components→com_comprofiler→plugins→user

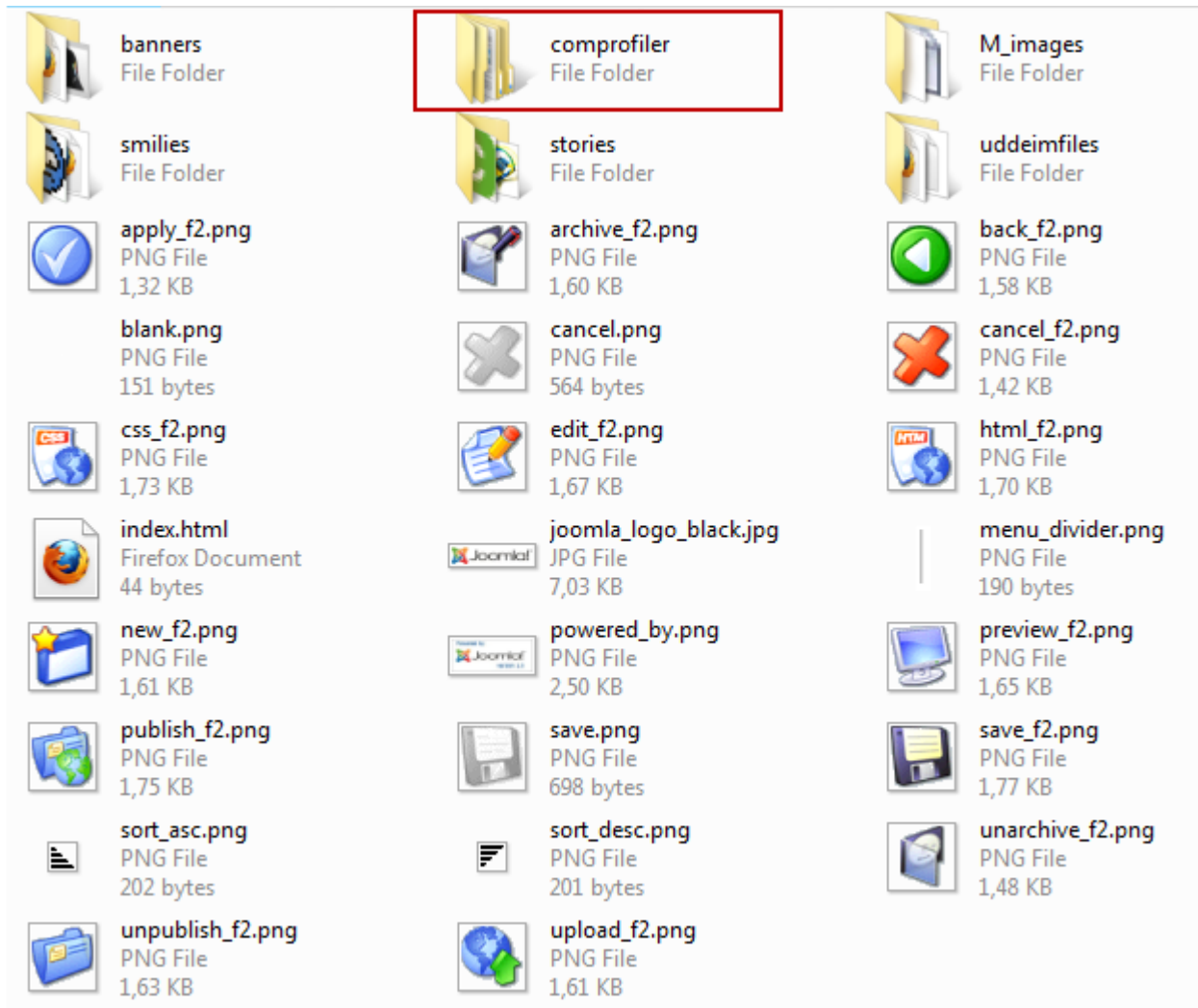


Figure 6: Addition of comprofiler folder in images root folder

A more detailed description of the file contents and basic functionality coded in the files located in the CB file system structure is given in the tables illustrated in the following figures. Once again all of this is subject to change.

| Filename | Comment about content |
|----------------------|--|
| comprofiler.html.php | This file handles the display element of CB frontend. |
| comprofiler.php | This file prepares CB frontend and handles “behind doors” elements such as Login, Registration, and Lost Password. |
| router.php | This file parses links to CB profiles and userlists. |
| comprofiler.xml | This file stores CB component install structure. |
| metadata.xml | This file is used to provide cross-platform support in alternative CMS systems (Joomla 1.0.x vs Joomla 1.5.x/1.6.0). |
| plugin/ | This folder stores all user and core plugins. |
| js/ | This folder stores all of CB JS structure (Custom and jQuery library). |
| images/ | This folder stores all CB default core images used throughout CB frontend (in some cases Backend as well). |
| | |

Figure 7: Items in components → com_comprofiler folder

| Filename | Comment about content |
|-----------|---|
| language/ | This folder contains language folders of all CB Language plugins installed. Initially it contains only the default_language fall-back folder as shown in Figure 9 and Figure 10 . |
| template/ | This folder contains template folders of all CB Template plugins installed. Core template plugin folder are shown in Figure 11 . |
| user/ | This folder contains user plugin folders of all CB (user-type) plugins installed. Core user-type plugins shown in Figure 18 . |
| | |

Figure 8: Items in components→com_comprofiler→plugin folder

| Filename | Comment about content |
|-------------------|---|
| default_language/ | Default language folder contains all language related translations to be used as a fall-back scenario if relevant language plugin is not found. Contents of default_language folder are in fact English language strings. |
| | Additional folder will be added for every CB Language plugin installed. |

Figure 9: Items in components→com_comprofiler→plugins→language folder

| Filename | Comment about content |
|---|--|
| images/ | This folder contains images that could be language dependent. |
| admin_language.php | This file contains all CB administration (backend) area language strings. |
| admin_reference_language.php | |
| calendar-locals.js | This file contains the javascript used to handle the calendar popup and has language strings embedded in it. |
| cbteampugins_language.php | This file contains language strings associated with CB Team released add-ons. |
| default_language.php | This file contains all frontend language strings for CB |
| default_language.xml | This is the XML file associated with the plugin |
| <p>The contents of the entire folder serve as the default language plugin used as a fall-back in case website (Joomla) has a default language setting that does not have a corresponding CB Language plugin. You should study the “How to prepare a CB Language plugin” guide available on JoomlaPolis.</p> | |

Figure 10: Items in components→plugin→languages→default_language

Community Builder 1.4 Stable – API Guide

| Filename | Comment about content |
|---|---|
| dark/ | This folder contains all files associated with the “dark” CB Template plugin. |
| default/ | This folder contains all files associated with the “default” CB Template plugin. |
| luna/ | This folder contains all files associated with the “luna” CB Template plugin. |
| osx/ | This folder contains all files associated with the “osx” CB Template plugin. |
| webfx/ | This folder contains all files associated with the “webfx” CB Template plugin. |
| winclassic/ | This folder contains all files associated with the “winclassic” CB Template plugin. |
| Additional folder will be added for every CB Template plugin installed (e.g., “coolness”, “mycommunity”, etc) | |

Figure 11: Items in components→com_comprofiler→plugin→templates folder

Community Builder 1.4 Stable – API Guide

| Filename | Comment about content |
|-------------------|--|
| images/ | |
| jqueryui/ | Contains css stylings for jquery ui usage. |
| calendar.css | |
| calendar_icon.jpg | |
| calendar_next.gif | |
| calendar_prev.gif | |
| dark.xml | |
| lightbox.css | |
| menuarrow.gif | |
| noprofiles.gif | |
| profiles.gif | |
| required.gif | |
| template.css | File containing css styling for CB profile and registration pages. |
| tooltip.png | |
| | |

Figure 12: Items in components→com_comprofiler→plugn→templates→dark folder

Community Builder 1.4 Stable – API Guide

| Filename | Comment about content |
|-------------------|--|
| images/ | |
| jqueryui/ | Contains css stylings for jquery ui usage. |
| calendar.css | |
| calendar_icon.jpg | |
| calendar_next.gif | |
| calendar_prev.gif | |
| dark.xml | |
| default.php | Contains php code for profile and user list rendering. |
| lightbox.css | |
| menuarrow.gif | |
| noprofiles.gif | |
| profiles.gif | |
| required.gif | |
| rtl.css | |
| template.css | File containing css styling for CB profile and registration pages. |
| tooltip.png | |
| | |

Figure 13: Items in components→com_comprofiler→plugn→templates→default folder

Community Builder 1.4 Stable – API Guide

| Filename | Comment about content |
|-------------------|--|
| images/ | |
| calendar.css | |
| calendar_icon.jpg | |
| calendar_next.gif | |
| calendar_prev.gif | |
| lightbox.css | |
| luna.xml | |
| menuarrow.gif | |
| noprofiles.gif | |
| profiles.gif | |
| required.gif | |
| tab.active.png | |
| tab.hover.png | |
| tab.png | |
| template.css | File containing css styling for CB profile and registration pages. |
| tooltip.png | |

Figure 14: Items in components→com_comprofiler→plugn→templates→luna folder

| Filename | Comment about content |
|-------------------|--|
| images/ | |
| calendar.css | |
| calendar_icon.jpg | |
| calendar_next.gif | |
| calendar_prev.gif | |
| lightbox.css | |
| menu_pro1.jpg | |
| menu_pro2.jpg | |
| menuarrow.gif | |
| noprofiles.gif | |
| osx.xml | |
| profiles.gif | |
| required.gif | |
| tab.png | |
| tab_active.png | |
| tab_hover.png | |
| template.css | File containing css styling for CB profile and registration pages. |
| tooltip.png | |

Figure 15: Items in components→com_comprofiler→plugn→templates→osx folder

Community Builder 1.4 Stable – API Guide

| Filename | Comment about content |
|-------------------|--|
| images/ | |
| calendar.css | |
| calendar_icon.jpg | |
| calendar_next.gif | |
| calendar_prev.gif | |
| lightbox.css | |
| menuarrow.gif | |
| noprofiles.gif | |
| profiles.gif | |
| required.gif | |
| template.css | File containing css styling for CB profile and registration pages. |
| tooltip.png | |
| webfx.xml | |

Figure 16: Items in components→com_comprofiler→plugn→templates→webfx folder

Community Builder 1.4 Stable – API Guide

| Filename | Comment about content |
|-------------------|--|
| images/ | |
| calendar.css | |
| calendar_icon.jpg | |
| calendar_next.gif | |
| calendar_prev.gif | |
| lightbox.css | |
| lookxpback.gif | |
| menuarrow.gif | |
| noprofiles.gif | |
| profiles.gif | |
| required.gif | |
| template.css | File containing css styling for CB profile and registration pages. |
| tooltip.png | |
| winclassic.xml | |

Figure 17: Items in components→com_comprofiler→plugn→templates→winclassic folder

Community Builder 1.4 Stable – API Guide

| Filename | Comment about content |
|--|--|
| plug_cbconnections/ | This folder contains all files associated with the connections plugin. |
| plug_cbcore/ | This folder contains all files associated with the CB core plugin that contains all core CB field-types code . |
| plug_cbmamblogtab/ | This folder contains all files associated with the CB Mamblog integration plugin (really left for legacy reasons). |
| plug_cbmamboauthortab/ | This folder contains all files associated with the CB content article integration plugin that is used to show articles posted by author in author's profile. |
| plug_cbmenu/ | This folder contains all files associated with the CB profile menu plugin (CB Menu). |
| plug_cbsimpleboardtab/ | This folder contains all files associated with the CB forum integration plugin that is used to show forum posts (Kunena, etc,) in poster's profile. |
| plug_pms_mypmspro/ | This folder contains all files associated with the CB PMS integration plugin used to embed private messaging functionality in user profile. Note: PMS component (e.g., uddelM must be installed separately). |
| plug_yancintegration/ | This folder contains all files associated with CB YANC newsletter integration plugin (really left for legacy reasons). |
| Additional folder will be added for every CB user-type plugin installed (e.g., plug_cbautowelcome, plug_helloworld, etc) | |

Figure 18: Items in components→com_comprofiler→plugin→user folder

| Filename | Comment about content |
|--------------------|---|
| cb.connections.php | This file handles the entire creation and display process of a users connections. |
| cb.connections.xml | This file is the install XML file associated with the connection plugin. |
| | |

Figure 19: Items in the components→ com_comprofiler→ plugin→ user→ plug_cbconnections folder

| Filename | Comment about content |
|-------------|---|
| cb.core.php | This file handles the entire formatting, validation, and display of CBs core field-types such as Text, Integer, etc.. |
| cb.core.xml | This file is the install XML file associated with the core plugin. |
| | |

Figure 20: Items in the components→ com_comprofiler→ plugin→ user→ plug_cbcore folder

| Filename | Comment about content |
|-------------------|---|
| cb.mamblogtab.php | This file handles the format and display of a users Mamblog entries. |
| cb.mamblogtab.xml | This file is the install XML file associated with the mamblog plugin. |
| | |

Figure 21: Items in components→com_comprofiler→plugin→user→plug_cbmamblogtab folder

| Filename | Comment about content |
|------------------|---|
| cb.authortab.php | This file handles the format and display of a users published CMS content articles. |
| cb.authortab.xml | This file is the install XML file associated with the article plugin. |
| | |

Figure 22: Items in components→com_comprofiler→plugin→user→plug_cbmamboauthortab folder

| Filename | Comment about content |
|-------------|---|
| cb.menu.php | This file handles the format and display of a users profile menu (View Profile, Connections, etc...). |
| cb.menu.xml | This file is the install XML file associated with the menu plugin. |
| | |

Figure 23: Items in components→com_comprofiler→plugin→user→plug_cbmenu folder

| Filename | Comment about content |
|-----------------------------|--|
| images/ | This folder stores all of the forum integrations images. |
| view/ | This folder stores the template-able views for the forum integration plugin. |
| cb.simpleboardtab.model.php | This file handles the forum integrations dynamic usage of multiple forum extensions allowing integration to multiple extensions. |
| cb.simpleboardtab.php | This file handles the rendering of the forum display (brings model and template-able views together and displays). |
| cb.simpleboardtab.xml | This file is the install XML file associated with the forum plugin. |
| | |

Figure 24: Items in components→com_comprofiler→plugin→user→ plug_cbsimpleboardtab folder

| Filename | Comment about content |
|------------------|---|
| pms.mypmspro.php | This file handles the usage of PMS within a users profile such as Quick Message tab and Send Private Message menu link. |
| pms.mypmspro.xml | This file is the install XML file associated with the PMS plugin. |
| | |

Figure 25: Items in components→com_comprofiler→plugin→user→ plug_pms_mypmspro folder

| Filename | Comment about content |
|----------|---|
| yanc.php | This file handles the integration of YANC to users profiles allowing easy subscription to various newsletters from within CB. |
| yanc.xml | This file is the install XML file associated with the YANC plugin. |
| | |

Figure 26: Items in components→com_comprofiler→plugin→user→ plug_yancintegration folder

| Filename | Comment about content |
|----------------------------------|--|
| database/ | This folder contains CB database structure files. |
| language/ | This folder contains backend CB language API files allowing inclusion of backend language plug-ins. |
| library/ | This folder contains all CB core API files. |
| xmlcb/ | This folder contains CB xml API usage (such as user-lists). |
| admin.comprofiler.controller.php | This file handles CB backend functionality. |
| admin.comprofiler.html.php | This file handles all of CBs backend display. |
| admin.comprofiler.php | This file is the CB backend “router” for bringing controller and html together. |
| comprofilej.xml | This file is the Joomla 1.5.x installation file for CB. |
| comprofiler.class.php | This file handles various aspects of CB front and backend API usage (generally included via cbimport function). |
| comprofiler.toolbar.php | This file handles CB backend toolbar preparation. |
| imgToolbox.class.php | This file handles CB image usage such as imagemagick or GD (image fields, etc...). |
| install.comprofiler.php | This file handles CB installation processes and checks. |
| plugin.class.php | This file handles CB field, plugin, tab, etc... API. |
| plugin.foundation.php | This file is the CB primary API file used to import CB plugin class or comprofiler class files for further extension of API usage by way of cbimport() function. |
| Snoopy.class.php | This file handles CB HTTP request usage via snoopy interface allowing fetch and submit requests. |
| toolbar.comprofiler.html.php | This file handles CB backend toolbar display. |
| toolbar.comprofiler.php | This file is simply an include for comprofiler.toolbar.php and handles preparation of plugin menus. |
| ue_config.php | This file stores CB configuration on a file based level. |
| ue_config_first.php | This file is a backup of CB configuration on a file based level. |
| uninstall.comprofiler.php | This file handle the CB uninstall process. |

Figure 27: Items in administrator→components→com_comprofiler folder

| Filename | Comment about content |
|---------------------|--|
| database.cbcore.xml | This file contains the CB database structure for all core CB tables (“#__comprofiler “, “#__comprofiler_fields”, etc.) |
| | |

Figure 28: Items in administrator→components→com_comprofiler→database folder

| Filename | Comment about content |
|-------------|---|
| english.php | This file attempts to include CB admin language files allowing translation in CB backend. If language plugin not found default language file is loaded. |
| | |

Figure 29: Items in administrator→components→com_comprofiler→language folder

Community Builder 1.4 Stable – API Guide

| Filename | Comment about content |
|-----------------|---|
| cb/ | This folder contains CBs more direct APIs such as database or authentication APIs. |
| pcl/ | This folder contains CBs archive APIs for handling of ZIP, etc.. when installing plugins. |
| phpinputfilter/ | This folder contains CBs filter API for cleaning input for safe usage in database, fields, etc... |
| phpmailer/ | This folder contains CBs phpmailer library supporting pop3, smtp, etc.. usage throughout CB. |
| | |

Figure 30: Items in administrator→components→com_comprofiler→library folder

Community Builder 1.4 Stable – API Guide

| Filename | Comment about content |
|------------------------|--|
| sql/ | This folder stores CBs SQL handlers. |
| xml/ | This folder stores CBs XML handlers. |
| cb.acl.php | This file provides ACL API usage throughout CB allowing detection and or manipulation of user ACL usergroups, etc... |
| cb.adminfilesystem.php | This file provides CB with backend admin specific API usage throughout CB and its plugins. |
| cb.authentication.php | This file handles CBs frontend authentication API such as login and logout (registration TBA). |
| cb.database.php | This file handles CBs database API usage allowing cross platform database calls. |
| cb.dbchecker.php | This file handles CBs database checking (used primarily in CBs tools). |
| cb.installer.php | This file handles CBs installer for CB plugins, languages, templates, etc... |
| cb.pagination.php | This file handles CBs pagination API on frontend and backend allowing paging of large queries. |
| cb.params.php | This file handles CBs plugin parameter usage. |
| cb.session.php | This file handles CBs database and cookie session handling in API format. |
| cb.tables.php | This file handles CBs database tables API usage such as storing of new users or updating of existing users. |
| | |

Figure 31: Items in administrator→components→com_comprofiler→library→cb folder

| Filename | Comment about content |
|---------------------|---|
| cb.sql.upgrader.php | This file handles upgrading of CB plug-ins and CB itself. |
| | |

Figure 32: Items in administrator→components→com_comprofiler→library→cb→sql folder

| Filename | Comment about content |
|----------------------|---|
| cb.xml.domit.php | This file handles CBs domit XML API usage. |
| cb.xml.simplexml.php | This file handles CBs SimpleXML API usage. |
| cb.xml.xml.php | This file handles CBs adaptation of simplexml and domit API usage applying various fixes. |
| | |

Figure 33: Items in administrator→components→com_comprofiler→library→cb→xml folder

| Filename | Comment about content |
|-------------------|--|
| pcl.pclziplib.php | This file handles CBs ZIP archive API. |
| pcl.tar.php | This file handles CBs TAR archive API. |
| | |

Figure 34: Items in administrator→components→com_comprofiler→library→pcl folder

| Filename | Comment about content |
|--------------------------------|---|
| phpinputfilter.inputfilter.php | This file handles CBs input filtering for storage in database, fields, etc... |
| | |

Figure 35: Items in administrator→components→com_comprofiler→library→phpinputfiler folder

| Filename | Comment about content |
|----------|-----------------------|
| | |

| | |
|-------------------------|--|
| phpmailer.phpmailer.php | This file contains the CB PHPMailer handler for sending emails with the PHPMailer method configured via CMS. |
| phpmailer.pop3.php | This file contains the CB POP3 handler for sending emails with the POP3 method configured via CMS. |
| phpmailer.smtp.php | This file contains the CB SMTP handler for sending emails with the SMTP method configured via CMS. |

Figure 36: Items in administrator→components→com_comprofiler→library→phpmailer folder

| Filename | Comment about content |
|--------------|---|
| cb.lists.xml | This file contains code to handle the CB userlist backend parameters structure. |
| | |

Figure 37: Items in administrator→components→com_comprofiler→xmlcb folder

2.3 Database After Installation

Once you have successfully installed the CB suite you should see a set of 10 new tables with the string 'comprofiler' in their name. These tables are (assuming that the Joomla prefix is set to 'jos_'):

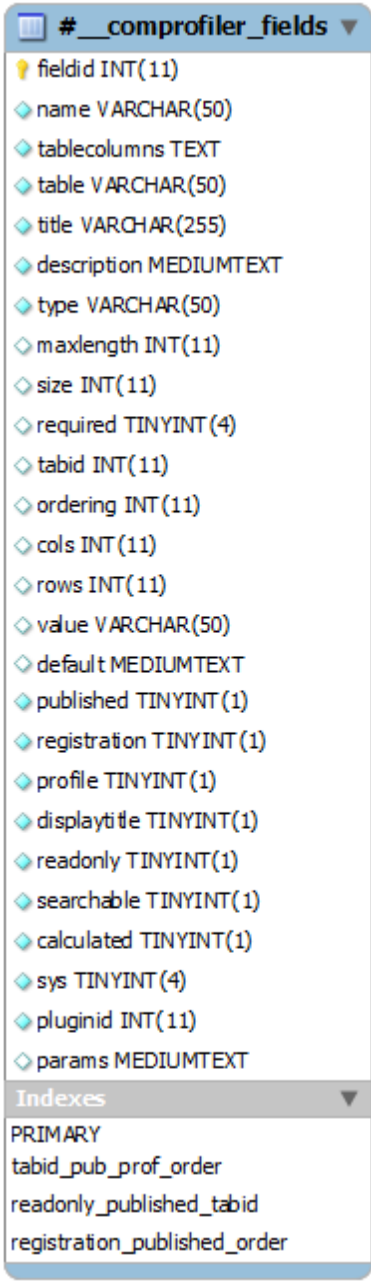
- jos_comprofiler
- jos_comprofiler_fields
- jos_comprofiler_field_values
- jos_comprofiler_lists
- jos_comprofiler_members
- jos_comprofiler_plugin
- jos_comprofiler_sessions
- jos_comprofiler_tabs
- jos_comprofiler_userreports
- jos_comprofiler_views

The contents of these tables and some basic information about the table usage is provided in the following sections. Once again, the database structure is subject to change and all information provided in this section should not be used as the basis for plugin development.

The CB Database should not be directly accessed. Instead the php API library functions should be used to manipulate data. The Database reference that follows is simply provided for documentation purposes.

| | |
|--|--|
| | <ul style="list-style-type: none"> • The comprofiler table links (1 to 1 relationship) with the CMS user table using the user_id field. • There is one row for each registered user • If the number of rows in this table does not match the number of rows in the CMS user table (e.g., 'jos_users' in Joomla CMS case), this means that website has been bypassing the CB registration process (perhaps using the CMS login module, instead of the CB login module). This case will be flagged when CB Tool checks are executed. • No duplication of data already available in CMS user table • When you add a new field via CB Field Management a new column will be added to this table |
|--|--|

Figure 38: The comprofiler table



- This table contains one row for every CB field created in CB Field Management area.
- As you can see the structure allows you to have fields that are associated directly with plugins (see pluginid column). This is important as we will see that CB fields can be extended with CB plugins or even new CB fieldtypes can be created by coding a CB fieldtype plugin.

Figure 39: The comprofiler_fields table

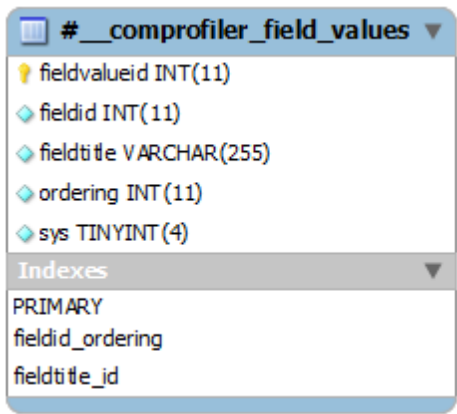
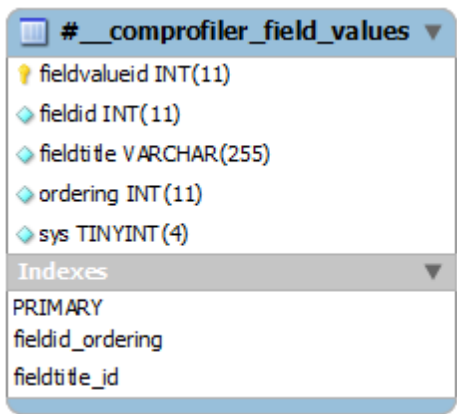
| | |
|---|--|
|  | <ul style="list-style-type: none"> • This table contains one row for every CB field value set (in case of fieldtypes that allow values from specific value lists) • The 'fieldid' column is used to associate the row with the actual CB field row of the 'comprofiler_fields' table • To see all allowed values of a CB field, just query for all rows that have the same fielded value as the CB field. |
|---|--|

Figure 40: The comprofiler_field_values table

| | |
|---|--|
|  | <ul style="list-style-type: none"> • This table contains one row for every CB field value set (in case of fieldtypes that allow values from specific value lists) • The 'fieldid' column is used to associate the row with the actual CB field row of the 'comprofiler_fields' table • To see all allowed values of a CB field, just query for all rows that have the same fielded value as the CB field. |
|---|--|

| | |
|---|---|
| <pre> #_comprofiler_lists listid INT(11) title VARCHAR(255) description MEDIUMTEXT published TINYINT(1) default TINYINT(1) usergroupids VARCHAR(255) useraccessgroupid INT(9) sortfields VARCHAR(255) filterfields MEDIUMTEXT ordering INT(11) col1title VARCHAR(255) col1enabled TINYINT(1) col1fields MEDIUMTEXT col2title VARCHAR(255) col2enabled TINYINT(1) col1captions TINYINT(1) col2fields MEDIUMTEXT col2captions TINYINT(1) col3title VARCHAR(255) col3enabled TINYINT(1) col3fields MEDIUMTEXT col3captions TINYINT(1) col4title VARCHAR(255) col4enabled TINYINT(1) col4fields MEDIUMTEXT col4captions TINYINT(1) params MEDIUMTEXT Indexes PRIMARY pub_ordering default_published </pre> | <ul style="list-style-type: none"> • This table contains one row for every CB user list created in the CB User List Management area. • The 'fieldid' column is used to associate the row with the actual CB field row of the 'comprofiler_fields' table • To see all allowed values of a CB field, just query for all rows that have the same fielded value as the CB field. |
|---|---|

Figure 41: The comprofiler_lists table

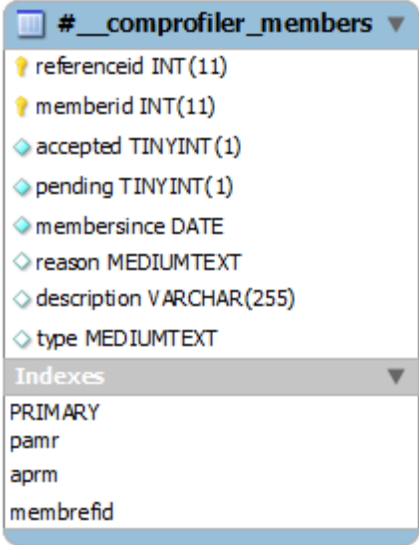
| | |
|---|--|
|  <p>#_comprofiler_members</p> <ul style="list-style-type: none">referenceid INT(11)memberid INT(11)accepted TINYINT(1)pending TINYINT(1)membersince DATEreason MEDIUMTEXTdescription VARCHAR(255)type MEDIUMTEXT <p>Indexes</p> <ul style="list-style-type: none">PRIMARYpamraprmmembrefid | <ul style="list-style-type: none">This table contains row for every CB connection established between CB user profiles |
|---|--|

Figure 42: The comprofiler_members table

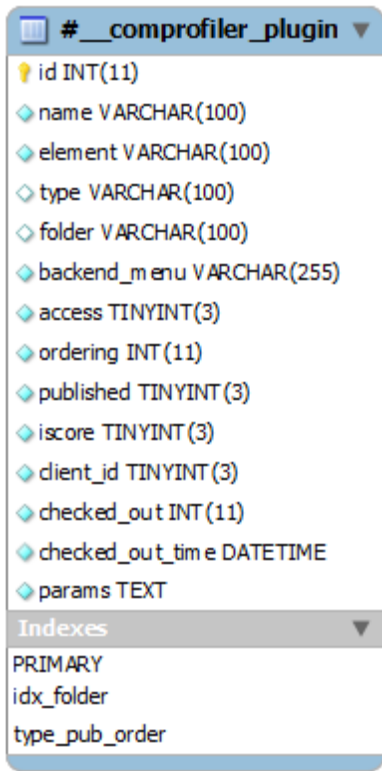
| | |
|---|--|
|  <p>#__comprofiler_plugin</p> <ul style="list-style-type: none">id INT(11)name VARCHAR(100)element VARCHAR(100)type VARCHAR(100)folder VARCHAR(100)backend_m enu VARCHAR(255)access TINYINT(3)ordering INT(11)published TINYINT(3)iscore TINYINT(3)client_id TINYINT(3)checked_out INT(11)checked_out_time DATETIMEparams TEXT <p>Indexes</p> <ul style="list-style-type: none">PRIMARYidx_foldertype_pub_order | <ul style="list-style-type: none">• This table contains row for every installed CB Plugin (core, template, language, user, etc.) |
|---|--|

Figure 43: The comprofiler_plugin table

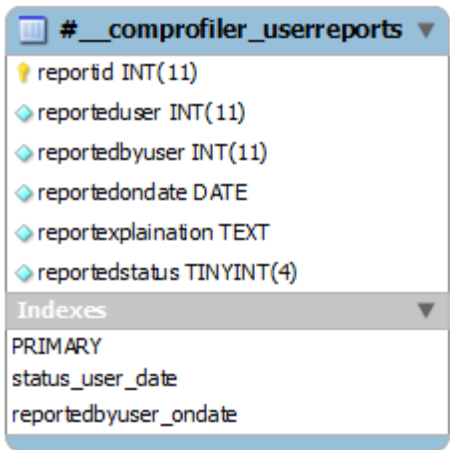
| | |
|--|--|
|  <pre>#__comprofiler_userreports reportid INT(11) reporteduser INT(11) reportedbyuser INT(11) reportedondate DATE reportexplanation TEXT reportedstatus TINYINT(4) Indexes PRIMARY status_user_date reportedbyuser_ondate</pre> | <ul style="list-style-type: none">• This table contains row for every user report created.• The reporteduser colum contains the userid of the offending user• The reportedbyuser column contains the userid of the user reporting the offence. |
|--|--|

Figure 44: The comprofiler_userreports table

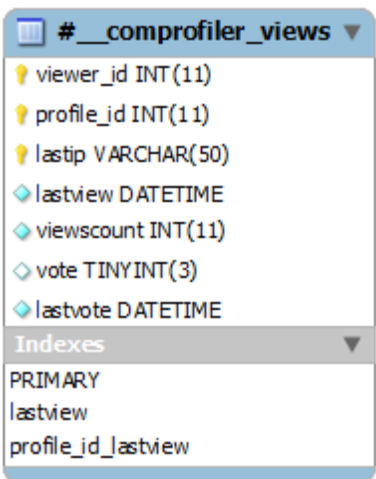
| | |
|---|--|
|  <pre>#__comprofiler_views viewer_id INT(11) profile_id INT(11) lastip VARCHAR(50) lastview DATETIME viewscout INT(11) vote TINYINT(3) lastvote DATETIME Indexes PRIMARY lastview profile_id_lastview</pre> | <ul style="list-style-type: none">• This table contains row for every profile view |
|---|--|

Figure 45: The comprofiler_views table

3 CB Plugins

3.1 Overview

A CB plugin is distributed as a zip package. A CB plugin package can be installed using the installer located in the CB Plugin Manager. We will see that CB plugins can be of different types and can contain many files.

3.2 Plug-in types

CB currently supports:

- Language plug-ins
- User plug-ins
- Template plug-ins
- Field-type plug-ins
- Component-like plug-ins

There are special classes of plug-ins:

- Core plug-ins (starting with “**cb**.”)
- Private Messaging plug-ins (starting with “**pms**.”)

3.3 Plug-in naming

To avoid plugin name conflicts and duplicate developments, plugin-authors are encouraged to contact the CB Team before starting development and choosing a plugin name.

Names from the English dictionary or reflecting other Joomla or Mambo components, as well as composed of two such names, as well as any name starting with «**cb**» are reserved either for Community Builder itself, or for coordinated developments. In fact the CB Team will not be allowing any listings on the JoomlaPolis directory area that do not adhere to these naming conventions.

3.4 Installation Process

Plug-in installation is documented in the detailed CB user manual (offered to CB Documentation subscribers). The CB Plugin Manager installer must be used to install CB Plugins.

3.5 Structure of plug-ins

The plug-in package is a zip file containing:

- An **XML** file (with **.xml** extension) including all information for installation and uninstall.
- An empty index **.html** file for each folder created by the installer.
- At least one **php** (with **.php** extension) file containing the plug-in main code (or language definitions).
- Further files and folders can be contained in the plug-in, as required by the plug-in functions (images, php files, language files folder etc).

3.6 XML file

The XML file is the first and most important file for a CB plug-in. It contains all “instructions” for CB to handle correctly the plug-in installation, settings and un-installation. It must be XML-language compliant (otherwise the simplified XML handler used by Joomla! will not flag errors in the file, but just stop working or flag other error).

It is divided into following main parts (some of which are optional):

- Header
- Files
- Plug-in parameters
- Tabs section with:
 - Tab definition
 - Tab parameters
 - Fields
- Installation SQL queries
- Uninstall SQL queries
- Installation file for PHP function
- Uninstall file for PHP function.
- Database install/upgrade function

3.6.1 Header

The XML header looks as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<cbinstall version="1.0.3" type="plugin" group="user">
  <name>pms.MyPMSPro</name>
  <author>JoomlaJoe and Beat</author>
  <creationDate>September 2005</creationDate>
  <copyright>(C) 2005 MamboJoe.com</copyright>
  <license>http://www.gnu.org/copyleft/gpl.html GNU/GPL</license>
  <authorEmail>mambojoe@mambojoe.com</authorEmail>
  <authorUrl>www.mambojoe.com</authorUrl>
  <version>1.0 RC 2</version>
  <description>Provides the MyPMS and PMS Pro integration for Community
Builder.</description>
```

Figure 46 – Header for XML Plug-in Installation File

The file character-set should be set right from the begin to UTF-8 for future compatibility with Joomla! 1.1, but accented characters escaped properly in html coding:

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

The installation file is included in this flag:

```
<cbinstall version="1.0.3" type="plugin" group="user">
```

The version indicates the used installer version for compatibility check.

The Type must be “plugin” for CB.

The Group indicates the type of plug-in and may be:

- “user” for user-type plug-ins
- “templates” for templates-plug-ins
- “language” for language-plug-ins
- other words are reserved for future use

```
<name>pms.MyPMSPro</name>
```

The name is used for two purposes:

1. To differentiate special kinds of plug-ins (first part separated by a dot “.”):
 - “cb.” Indicates core plug-ins (reserved for CB’s own use only)
 - “pms.” Indicates PMS-capable plug-ins

The kinds of plug-ins also set the class from which the plug-in object is derived.

2. To name the folder to create for installation (prefixed by “plug_” in case of user-plug-ins). Dots “.” And spaces “ ” are converted to underscores “_” for the folder name.

```
<author>JoomlaJoe and Beat</author>
<creationDate>September 2005</creationDate>
<copyright>(C) 2005 MamboJoe.com</copyright>
<license>http://www.gnu.org/copyleft/gpl.html GNU/GPL</license>
<authorEmail>mambojoe@mambojoe.com</authorEmail>
<authorUrl>www.mambojoe.com</authorUrl>
```

are free fields. Email and URL must be valid for future use.

```
<version>1.0 RC 2</version>
```

This is important: The version must strictly match the CB version. It is expected that the API may evolve until 1.0 final release without full backwards compatibility, if needed.

```
<description>Provides the MyPMS and PMS Pro integration for Community
Builder.</description>
```

This description is shown after successful install and when displaying the plug-in. It may contain instructions for the admin.

3.6.2 Files

The next section is about files to install:

```
<files>
  <filename plugin="pms.mypmspro">pms.mypmspro.php</filename>
  <filename>index.html</filename>
</files>
```

Figure 47 – PMS plug-in File section for XML Plug-in Installation File

The XML file should not be included; it is copied by the CB installer at the end if there are no XML errors.

The main file containing the main classes should have the tag argument “**plugin=**” with the name of the plug-in as above.

Don’t forget the empty index.html file for web server settings allowing directory listings.

A language plug-in needs to have following files:

```
<files>
  <filename plugin="english">default_language.php</filename>
  <filename>index.html</filename>
```

```
<filename>calendar-locals.js</filename>
<filename>images/index.html</filename>
<filename>images/nophoto.jpg</filename>
<filename>images/pendphoto.jpg</filename>
<filename>images/tnnophoto.jpg</filename>
<filename>images/tnpendphoto.jpg</filename>
</files>
```

Figure 48 - Language plug-in File section for XML Plug-in Installation File

we see that directories can be created and copied from the zip file.

3.6.3 Plug-in Parameters

Next tag is the `<params>` tag for the plug-in. We will see later that tabs can also have their own parameters.

```
<params>
  <param name="pmsType" type="list" default="1" label="PMS Component type"
description="Choose type of component installed. &lt;strong&gt;IMPORTANT: Component
configuration must also be done!&lt;/strong&gt;">
    <option value="1">MyPMS Open Source</option>
    <option value="2">PMS Pro</option>
  </param>
  <param name="@spacer" type="spacer" default="" label="" description="" />
  <param name="pmsMenuText" type="text" size="25" default="_UE_PM_USER" label="PMS Send
Menu/Link text" description="Default is _UE_PM_USER, the local translation of &quot;Send
Private Message&quot;" />
  <param name="@spacer" type="spacer" default="only for PMS Pro" label="Following
parameters:" description="" />
  <param name="online" type="radio" default="1" label="_UE_ONLINESTATUS"
description="IMPORTANT: General Community Builder configuration must also allow to show
this!">
    <option value="0">Hide</option>
    <option value="1">Show</option>
  </param>
  <param name="examplefieldsel" type="field" size="" default="" label="Example Field"
description="For example" />
  <param name="newslettersRegList" type="custom" class="getNewslettersTab"
method="loadNewslettersList" default="" label="_UE_NEWSLETTERSREGLIST"
description="_UE_NEWSLETTERSREGLIST_DESC" />
</params>
```

Figure 49 - Params section for XML Plug-in Installation File

Most/all Joomla! modules xml parameter types can be used also for plug-ins. A few extensions have been made:

- The “spacer” type has been extended to be also used as separator with text or as section header.
- The “field” type has been added to be able to select a user or CB field from the list of published fields, sorted by tab ordering & field ordering. The parameter value stored and passed to the plug-in is the fieldid, not the field name, so that in case of future change of name, the field stays referenced.
- The “custom” type has been added. This allows a plug-in to display his own controls, as for instance the list of newsletters in the **YaNC** plug-in.

Parameters have default values that must be matched when fetching the parameters in the plug-in.

Parameters are optional, but following tags need to be left if there are NO parameters:

```
<params>
</params>
```

Otherwise a text area will be displayed for typing in freely parameters.

3.6.4 Tabs

Next section is about tabs included in user plug-ins:

```
<tabs>
  <tab name="_UE_PMSTAB" description="" class="getMyPMSProTab" fields="0"
position="cb_right" displaytype="html">
  <params>
    <param name="@spacer" type="spacer" default="Quick PMS Settings" label=""
description="" />
    <param name="showTitle" type="list" default="1" label="Show Tab title"
description="Show the title of the tab inside this tab. The description is also shown, if
present. &lt;strong&gt;IMPORTANT: The title is the tab title here.&lt;/strong&gt;">
      <option value="0">Hide</option>
      <option value="1">Show</option>
    </param>
    <param name="width" type="text" size="10" default="30" label="Width (chars)"
description="" />
  </params>
  <fields>
  </fields>
</tab>
</tabs>
```

Figure 50 - Tabs section for XML Plug-in Installation File

A user plug-in may include multiple tabs with a “<tab> tag within the “<tabs>” tag.

3.6.4.1 Tab definition

The tab parameters are as follows:

| | |
|-------------|---|
| Name | Name appearing as title on tab |
| Description | Html text appearing usually at begin of tab. |
| Class | Name of the class in the main php file for handling the tab |
| Fields | 0: tab does not accept normal cb fields on it. Any fields defined under <fields> tag will be created and treated as private plug-in fields and will not appear in front-end or backend. 1: tab does accept normal cb fields on it. |
| Position | Default position of tab on user profile. Top-down / left to right: cb_head, cb_left, cb_middle, cb_right, cb_tabmain, cb_underall . |
| Displaytype | Default type of display of this tab: tab, div, html, overlib, overlibfix, overlibsticky . |

3.6.4.2 Tab parameters

All parameters rules listed above for plug-ins apply also to tab parameters.

3.6.4.3 Field

Tabs can define their own fields.

The fields section within tabs allows the plug-in to add its own CB-standard fields to the tab.

If fields="1" in the <tab> tag, they will display after the plug-in-specific tab content, and appear as any other field in front-end on user profile, update profile, and in the backend under fields management.

If fields="0" in the <tab> tag, these fields will not display in the front-end and not appear in the backend. They will be stored as usual in the #__comprofiler table along with other CB fields. This is the preferred way to store user-specific data for this tab.

The XML would look like this:

```
<tabs>
  <tab name="Example" description="" class="getExampleTab" fields="1"
position="cb_tabmain" displaytype="tab">
  <params>
  </params>
  <fields>
    <field title="Your first company" name="cb_firstcompany" description="First company
you worked in" type="text" registration="0" profile="1" readonly="0" params="" />
  </fields>
</tab>
</tabs>
```

Figure 51 – Tabs with Fields section for XML Plug-in Installation File

The <field> tag attributes correspond to the #__comprofiler_fields table fields and can be easily figured out from existing CB fields.

On uninstall plug-in, tabs and field definitions are removed, but the corresponding fields data in the user's Community Builder #__comprofiler table are kept, so on re-install, they are conserved.

On re-install, if the **type of field changes** compared to previous version of the plugin, the field type in the database **will be automatically updated**, which can potentially **lead to data-loss, if the field-type are different**.

3.6.5 Database tag description

The database tag section can be used to create your plugin's database structure and even add indexes to it. See example from CB ProfileBook plugin below:

```
<database version="1">
  <table name="#__comprofiler_plug_profilebook" class="" strict="true" drop="never">
    <columns strict="true" drop="never">
      <column name="id" type="sql:int(11)" unsigned="true" auto_increment="1"
        strict="true" />
      <column name="mode" type="sql:char(1)" null="false" default="g" strict="true" />
      <column name="posterid" type="sql:int(11)" unsigned="true" null="true"
        strict="true" />
      <column name="posterip" type="sql:varchar(255)" strict="true" />
      <column name="postername" type="sql:varchar(255)" null="true" default=""
        strict="true" />
      <column name="posteremail" type="sql:varchar(255)" null="true" strict="true" />
      <column name="posterlocation" type="sql:varchar(255)" null="true" strict="true" />
      <column name="posterurl" type="sql:varchar(255)" null="true" strict="true" />
      <column name="postervote" type="sql:int(11)" unsigned="true" null="true"
        strict="true" />
      <column name="postertitle" type="sql:varchar(128)" strict="true" />
      <column name="postercomment" type="sql:text" strict="true" />
      <column name="date" type="sql:datetime" null="true" strict="true" />
      <column name="userid" type="sql:int(11)" unsigned="true" strict="true" />
      <column name="feedback" type="sql:text" null="true" strict="true" />
      <column name="editdate" type="sql:datetime" null="true" strict="true" />
      <column name="editedbyid" type="sql:int(11)" unsigned="true" strict="true" />
      <column name="editedbyname" type="sql:varchar(255)" null="true" strict="true" />
      <column name="published" type="sql:tinyint(3)" strict="true" />
      <column name="status" type="sql:tinyint(3)" strict="true" />
    </columns>
    <indexes strict="true" drop="never">
      <index name="PRIMARY" type="primary">
        <column name="id" />
      </index>
      <index name="user_mode_date">
        <column name="userid" />
        <column name="mode" />
        <column name="date" />
      </index>
      <index name="pub_user_mode_date">
        <column name="published" />
        <column name="userid" />
        <column name="mode" />
        <column name="date" />
      </index>
      <index name="mode_pub_date">
        <column name="mode" />
        <column name="published" />
        <column name="date" />
      </index>
      <index name="status_user_mode">
        <column name="status" />
        <column name="userid" />
        <column name="mode" />
      </index>
      <index name="poster_mode_pub_date">
        <column name="posterid" />
        <column name="mode" />
        <column name="published" />
        <column name="date" />
      </index>
    </indexes>
  </table>
</database>
```

3.6.6 SQL queries (install and un-install)

On plug-in install and un-install, the xml file can specify SQL queries to be performed as follows:

```
<install>
  <queries>
    <query>
      CREATE TABLE IF NOT EXISTS `#__comprofiler_plug_example_test` (
        `id` int(11) NOT NULL default '0',
        `user_id` int(11) NOT NULL default '0',
        `example_text` mediumtext,
        PRIMARY KEY (`id`)
      ) TYPE=MyISAM;
    </query>
  </queries>
</install>
<uninstall>
  <queries>
    <query>
      DROP TABLE IF EXISTS `#__comprofiler_plug_example_test`
    </query>
  </queries>
</uninstall>
```

Figure 52 - Install section for XML Plug-in Installation File

Multiple queries can be made on install and un-install using multiple “<query>” tags. Tables specific to a plug-in should be named “#__comprofiler_plug_NAMEofPLUG”.

Note: The <query> tag should be used as last resort,. Instead the CB database xml tag should be used for handling databases as CB automatically handles database upgrading automatically.

3.6.7 Install code (also un-install code)

On plug-in install and un-install, the xml file can specify PHP code to be called as follows:

```
<installfile>example.userplugin.php</installfile>
<uninstallfile>example.userplugin.php</uninstallfile>
```

In this case, the filename is same as the main php file, but the files may differ. If these files are omitted from the files list, they will be still copied into the plug-in directory.

On Install, the function:

```
string plug_XXXX install();
```

will be called. The “XXXX” above will be replaced by the plugin system name: that is the the main file plug attribute: in the XML file: <file plug=“XXXX”>, with spaces removed (there shouldn’t be any spaces there anyway!) and dots “.” Changed to underscores “_”.

On Un-install, the function:

```
string plug_XXXX_uninstall();
```

will be called. The “XXXX” being replaced same as for install.

Both functions must return a string (can be empty) with html-formatted results to be displayed on the admin interface upon completion of install or uninstall.

3.7 Language plug-ins

Language plug-ins are a type of CB plug-ins. The Joomla!polis language forum area has a detailed pdf guide for translators describing how to construct a CB Language plugin.

Please refer to that document as the most up to date content regarding CB Language plugins. The rest of this section just gives an overview for completeness.

The language plugin XML file (from CB 1.4) typically looks as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<cbinstall version="1.0" type="plugin" group="language">
  <name>en-GB</name>
  <author>CB Team</author>
  <creationDate>2011-02-10</creationDate>
  <copyright>(C) joomlapolis.com</copyright>
  <license>http://www.gnu.org/licenses/old-licenses/gpl-2.0.html GNU/GPL version
2</license>
  <authorEmail>cbteam@joomlapolis.com</authorEmail>
  <authorUrl>www.joomlapolis.com</authorUrl>
  <version>1.4</version>
  <description>Provides the English language for Community Builder 1.4 core
functions.</description>
  <files>
    <filename plugin="language">language.php</filename>
    <filename>index.html</filename>
    <filename>admin_language.php</filename>
    <filename>cbteampugins_language.php</filename>
    <filename>calendar-locals.js</filename>
    <filename>images/index.html</filename>
    <filename>images/nophoto.jpg</filename>
    <filename>images/pendphoto.jpg</filename>
    <filename>images/tnnophoto.jpg</filename>
    <filename>images/tnpendphoto.jpg</filename>
  </files>
  <params>
</params>
  <database>
</database>
</cbinstall>
```

Figure 53 – Language File Example for XML Plug-in Installation File

The **bold underlined** elements above must be changed in the xml file for each other language to correspond to Joomla!’s language name.

A “`default_language`” core plug-in is the default plug-in for the default American English. An “`English`” plug-in can be loaded, and will be used if existing instead of the `default_language` plug-in.

Important: the language file character set of the plugin should match the character-set of the language file of the site (and if multiple languages are used, all should match the same character set: this is a Mambo/Joomla! constraint, since Joomla! does not support character sets transcodings).

The only universal solution to this problem is the use of the UTF-8 character-set (variable length-byte encodings from 1-8 bytes per character). As Joomla is moving towards UTF-8 being mandatory in Joomla! 1.1, **translators are encouraged to use UTF-8 from the start, and site-admins to install corresponding UTF-8 core language files.**

For performance reasons, the published status is not taken into account for language plug-ins.

The translation/localization of the calendar JavaScript program used to choose dates is in the `calendar-locals.js` file. The author site of the calendar application gives incomplete translations/localizations, which can also be used as example.

[Detailed instructions](#) for translators are available on the Joomla!polis [CB Language projects forge area](#) and assistance can be given also on our Joomla!polis [languages forum](#) area.

3.8 User plug-ins

User-type plug-ins are the functions-extending plug-ins for adding functions to CB. These can include tabs and userbots.

3.8.1 Parameter passing

Parameters are defined in the xml file, as shown above.

They are stored CB-internally with the plug-ins SQL table.

They can be used in the plug-in methods as follows:

```
$params = $this->params;
```

```
$pmsType      = $params->get('pmsType', '1');  
$showTitle    = $params->get('showTitle', "1");
```

The first argument of `$params->get` is the name of the parameter, which must correspond to the “`name=`” attribute of the corresponding “`<param>`” tag in the XML file.

The second optional argument is the default value of that parameter, which should correspond to the “`default=`” attribute of the corresponding “`<param>`” tag in the XML file.

The third optional parameter of this method is the name of the search/paging/sorting sorter (see paragraph 3.10.4 on page 68).

```
/**  
 * gets the name input parameter for search and other functions  
 * @param string name of parameter of plugin  
 * @param string postfix for identifying multiple pagings/search/sorts (optional)  
 * @returns string value of the name input parameter  
 */  
function getPagingParamName($name="search", $postfix="")
```

In case of tabs, the parameters in `$this->params` correspond to the parameters of the tab **and** of the parameters of the plug-in. Later user-specific parameters/preference settings may also be available probably this way.

3.8.2 Error Management

Plug-ins user bots and tabs can give multiple error messages, calling each time this method:

```
/**  
 * PRIVATE method: sets the text of the last error  
 * @param string error message  
 * @return boolean true  
 */  
function setErrorMSG($msg)
```

They must also raise an error condition to stop the normal CB workflow, and make CB display an error message (for now using a JS alert):

```
/**  
 * PRIVATE method: sets the error condition and priority (for now 0)  
 * @param error priority  
 * @return boolean true  
 */  
function raiseError($priority)
```

3.8.3 Objects

All interactions between CB and plug-ins is done with objects. Some functions are non-object, but the rule is to have objects, and as CB is developed further, it will become more and more object-oriented.

Normally, the objects of plug-ins are instantiated only once, and object variables are kept throughout the whole execution of the Community Builder call (page generation).

So for instance, if a given page generation calls a userbot method, then later a tab display method, it's the same object which is created only once, and the variables of the object will remain.

Each object should call the constructor (in php4 compliant manner) of its parent class.

Please keep in mind that in the future, not all tabs will be loaded and generated at each display.

CB developers reserve the possibility to either draw tabs only one by one and do page reloads on each tab switch, or to download tab content dynamically as needed. Therefore, the tab `getDisplayTab()` method may not be called upfront, even if the tab title is drawn, but not visible.

3.8.4 Tabs

[Note: needs to be updated based on `comprofiler.class.php`]

All tabs are derived from this class:

```
/**
 * Tab Class for handling the CB tab api
 * @package Community Builder
 * @author JoomlaJoe and Beat
 */
class cbTabHandler extends cbPluginHandler {
    /**
     * Constructor
     */
    function cbTabHandler() {
        $this->cbPluginHandler();
    }
    /**
```

Community Builder 1.4 Stable – API Guide

```
* Generates the menu and user status to display on the user profile by calling
back $this->addMenu
* @param object tab reflecting the tab database entry
* @param object mosUser reflecting the user being displayed
* @param int 1 for front-end, 2 for back-end
* @returns boolean : either true, or false if ErrorMSG generated
*/
function getMenuAndStatus($tab,$user,$ui) {
}
/**
* Generates the HTML to display the user profile tab
* @param object tab reflecting the tab database entry
* @param object mosUser reflecting the user being displayed
* @param int 1 for front-end, 2 for back-end
* @returns mixed : either string HTML for tab content, or false if ErrorMSG
generated
*/
function getDisplayTab($tab, $user, $ui) {
}
/**
* Generates the HTML to display the user edit tab
* @param object tab reflecting the tab database entry
* @param object mosUser reflecting the user being displayed
* @param int 1 for front-end, 2 for back-end
* @returns mixed : either string HTML for tab content, or false if ErrorMSG
generated
*/
function getEditTab($tab, $user, $ui) {
}
/**
* Saves the user edit tab postdata into the tab's permanent storage
* @param object tab reflecting the tab database entry
* @param object mosUser reflecting the user being displayed
* @param int 1 for front-end, 2 for back-end
* @param array _POST data for saving edited tab content as generated with
getEditTab
* @returns mixed : either string HTML for tab content, or false if ErrorMSG
generated
*/
function saveEditTab($tab, $user, $ui, $postdata) {
}
/**
* Generates the HTML to display the registration tab/area
* @param object tab reflecting the tab database entry
* @param object mosUser reflecting the user being displayed
* @param int 1 for front-end, 2 for back-end
* @returns mixed : either string HTML for tab content, or false if ErrorMSG
generated
*/
function getDisplayRegistration($tab, $user, $ui) {
}
/**
* Saves the registration tab/area postdata into the tab's permanent storage
* @param object tab reflecting the tab database entry
* @param object mosUser reflecting the user being displayed
* @param int 1 for front-end, 2 for back-end
* @param array _POST data for saving edited tab content as generated with
getEditTab
* @returns mixed : either string HTML for tab content, or false if ErrorMSG
generated
*/
function saveRegistrationTab($tab, $user, $ui, $postdata) {
}
```

PMS and Menu tabs definitions are derived from this base class, and are documented with comments in their implementation files.

Each tab in the `getDisplayTab` should include the code to display the description set by the admin in backend:

```
if($tab->description != null) $return .= "\t\t<div
class=\"tab_Description\">".unHtmlspecialchars(getLangDefinition($tab-
>description))."</div>\n";
```

Please **note that all user parameters are unescaped in all cases** independently of the `magic_quotes_gpc` PHP setting, and that if you want to use those in SQL queries, they need to be properly escaped, e.g. with `$database->getEscaped($row->username)` as escapings may vary between database types (either `'` or `"` for ' e.g.).

3.8.5 CB Events

CB Events are methods or functions called upon particular CB related actions. They are allowed to handle those events, and to output messages, errors and to block operations of CB.

Great care must be placed when modifying data passed to them.

A userbot must first be registered.

The CB API for this is:

```
/**
 * Registers a function to a particular event group
 * @param string The event name
 * @param string The function name
 */
function registerFunction( $event, $method, $class=null )

called as:
global $_PLUGINS
$_PLUGIN->registerFunction
```

This is done in the main plug-in php file as follows for a function

pluginExampleBeforeSaveUser ():

```
$_PLUGINS->registerFunction( 'onBeforeUserUpdate', 'pluginExampleBeforeSaveUser' );
```

and as follows for a method **pluginExampleBeforeSaveUser ()** of a class

'getExampleTab':

```
$_PLUGINS->registerFunction('onBeforeUserUpdate',  
'pluginExampleBeforeSaveUser','getExampleTab' );
```

Please **note that all user parameters are unescaped in all cases** independently of the `magic_quotes_gpc` PHP setting, and that if you want to use those in SQL queries, they need to be properly escaped, e.g. with `$database->getEscaped($row->username)` as escapings may vary between database types (either `\` or `"` for `'` e.g.).

Here is a list of events that can be registered and the arguments received:

3.8.5.1 User management events

CB triggers many user management events that can use to sync the user data and events of CB with third party applications or to extend the functionality of CB.

Much of the plug-in framework is leverages off of the core mambo Mambot code.

At a very minimum you must register the functions you want to leverage with the available CB events.

Following table shows the event name identical with function name, and the parameters passed to the function/method:

```
//Add user via Admin events  
function onBeforeNewUser (&$row, &$rowExtras, false)  
function onAfterNewUser ($row, $rowExtras, false, true)  
  
//New user Registration events  
function onBeforeUserRegistration (&$row,&$rowExtras, false)  
function onAfterUserRegistration ($row, $rowExtras, true)  
  
//Confirm User events  
function onBeforeUserConfirm ($userObject)  
function onAfterUserConfirm ($userObject, true)  
  
//Approve/Reject User events  
function onBeforeUserApproval ($row, $approved)  
function onAfterUserApproval ($row, $approved, $success)  
function onUserActive ($row, $activated=true)  
  
//Update User events  
function onBeforeUserUpdate (&$row,&$rowExtras)  
function onAfterUserUpdate ($row, $rowExtras, true)  
  
//Delete User events  
function onBeforeDeleteUser ($row->id)  
function onAfterDeleteUser ($row->id,true)
```

3.8.5.2 User session events

CB triggers two user session events u can use to authenticate users with external services, like for example LDAP or 3PD forums.

```
//Login User event
function onBeforeLogin ($username, $passwd2)
function onAfterLogin ($row, true)

//Logout User event
function onBeforeLogout ($row)
function onAfterLogout ($row, true)
```

3.8.5.3 View profile events

When profiles are viewed, events are generated for plug-ins:

- Before to allow the profile view and to alter if necessary profile content.
- After to perform other actions if necessary.

```
// View profile events
function onBeforeUserProfileDisplay ($user, $ui, $cbUserIsModerator, $cbMyIsModerator) //
ui=1 front, ui=2 backend, 2* boolean

function onAfterUserProfileDisplay ($user, $succes=true)
```

3.8.5.4 Connection events

Connection events are generated to enable plug-ins to perform particular actions on those changes of connections states.

```
// Connection events
function onBeforeAddConnection ($userid,$connectionid, $ueConfig['useMutualConnections'],
$ueConfig['autoAddConnections'],&$userMessage)

function onAfterAddConnection ($userid,$connectionid,
$ueConfig['useMutualConnections'],$ueConfig['autoAddConnections'])

function onBeforeRemoveConnection ($userid,$connectionid,
$ueConfig['useMutualConnections'],$ueConfig['autoAddConnections'])

function onAfterRemoveConnection ($userid,$connectionid,
$ueConfig['useMutualConnections'],$ueConfig['autoAddConnections'])

function onBeforeDenyConnection ($userid,$connectionid,
$ueConfig['useMutualConnections'],$ueConfig['autoAddConnections']))

function onAfterDenyConnection ($userid,$connectionid,
$ueConfig['useMutualConnections'],$ueConfig['autoAddConnections']))

function onBeforeAcceptConnection ($userid,$connectionid,
$ueConfig['useMutualConnections'],$ueConfig['autoAddConnections']))

function onAfterAcceptConnection ($userid,$connectionid,
$ueConfig['useMutualConnections'],$ueConfig['autoAddConnections']))
```

3.8.6 Generating HTML output

HTML can be generated by plug-ins on following occasions:

- User Profile Display (method `getDisplayTab`)
- User Edit (method `getEditTab`, saving output with method `saveEditTab`)
- Registration Page (method `getDisplayRegistration`, saving output with method `saveRegistrationTab`)

These methods are described in the Tab Object paragraph above.

3.8.6.1 Accessibility

Plug-ins should output accessible HTML code (user profile of CB already uses a table-less design), allowing disabled people to use reading browsers.

3.8.6.2 W3C Compliance

Code generated by CB aims to be compliant with W3C xhtml 1.0 transitional standards and be displayed correctly on all popular modern browsers including:

- Firefox 1.5+
- Internet Explorer 6+
- Opera 10+
- Safari 4+

It should display as correctly as expected (degrading gracefully) in following circumstances:

- JavaScript on/off
- Images on/off
- CSS on/off

The RC2 release achieves this for user profiles, all front-end management views, and started doing it for the registration and user edit views.

3.8.6.3 Separating logic from output

One sound design practice is to separate the business logic from output, so each method should first compute all variables to be displayed, and then output html code including those variables.

3.8.6.4 patTemplates

patTemplates are the output generation chosen by Joomla! 1.1 and 1.2. CB will follow this after mambo 4.5.0 support is dropped.

Plug-ins may explore this before, but core plug-in can't for backwards compatibility.

3.8.7 User profile

See **getDisplayTab** method.

3.8.8 User edit

See **getEditTab** and **saveEditTab** methods.

3.8.9 Registration

See **getDisplayRegistration** and **saveRegistrationTab** methods.

3.8.10 Field Validation

Fields on registration and user edit can be validated by plug-ins. Minimum validation is with PHP in the plug-in. Optionally, JavaScript can also be used.

3.8.10.1 In PHP in the plug-in

See CB Events:

- **onAfterUserUpdate** example:

```
$_PLUGINS->registerFunction( 'onBeforeUserUpdate', 'pluginExampleBeforeSaveUser' );  
  
/**  
 * Example store user method  
 * Method is called before user data is stored in the database  
 * @param array holds the core mambo user data  
 * @param array holds the community builder user data  
 * @param boolean true if a new user is stored  
 */  
function pluginExampleBeforeSaveUser(&$user,&$cbUser) {  
    global $_POST, $_PLUGINS;  
  
    if ($_POST['username'] == $_POST['password']) {  
        $_PLUGINS->raiseError(0);  
    }  
}
```

```
    $_PLUGINS->_setErrorMsg("Password must be different from username!");
}
return true;
}
```

- **onAfterUserRegistrationSave**

```
$_PLUGINS->registerFunction( 'onBeforeUserRegistration',
'pluginExampleBeforeUserRegistration' );

/**
 * Example registration verify user method
 * Method is called before user data is stored in the database
 * @param array holds the core mambo user data
 * @param array holds the community builder user data
 * @param boolean false
 */
function pluginExampleBeforeUserRegistration(&$user,&$cbUser, $stored) {
    global $_POST, $_PLUGINS;

    if ($_POST['username'] == $_POST['password']) {
        $_PLUGINS->raiseError(0);
        $_PLUGINS->_setErrorMsg("Password has to be different from username!");
    }
    return true;
}
```

3.8.10.2 In JavaScript generated by the plug-in

[Note: needs to be updated to discuss new jquery validation method used in email field type in cb.core.php file]

CB API has a method for tabs to add user-side JavaScript validation (keep in mind that these do not operate when the user disabled his browser JavaScript, so PHP validation is also needed):

```
/**
 * adds a validation JS code for the Edit Profile and Registration pages
 * @param string Javascript code ready for HTML output, with a tab \t at the begin and
a newline \n at the end.
 */
function _addValidationJS($js)
```

This function can be called from the plug-in from the **getEditTab** method (example):

```
/**
 * Generates the HTML to display the user edit tab
 * @param object tab reflecting the tab database entry
 * @param object mosUser reflecting the user being displayed
 * @param int 1 for front-end, 2 for back-end
 * @returns mixed : either string HTML for tab content, or false if ErrorMSG generated
 */
function getEditTab($tab,$user,$ui) {
    $params = $this->params;
    $exampleText = $params->get('exampleText', 'Text Parameter not set!');
```

```

$ret = "<p>Hello ".$user->name." !</p>";
$ret .= "<p>ParameterText: ".$exampleText."</p>";
$ret .= "<p>Be carefull: don't set password same as username !</p>";
$this->_addValidationJS( "\tif (me['username'].value == me['password'].value) {\n"
    ."\t  errorMsg += \"".html_entity_decode("Password must differ from
username!") ."\n\n\n"
    ."\t  me['password'].style.background = \"red\";\n"
    ."\t  iserror=1;\n"
    ."\t}\n");
// also see event: 'onBeforeUserUpdate', implemented here in function:
pluginExampleBeforeSaveUser().
return $ret;
}

```

and/or from the `getDisplayRegistration` methods as follows (example):

```

/**
 * Generates the HTML to display the registration tab/area
 * @param object tab reflecting the tab database entry
 * @param object mosUser reflecting the user being displayed (here null)
 * @param int 1 for front-end, 2 for back-end
 * @return mixed : either string HTML for tab content, or false if errorMsg generated
 */
function getDisplayRegistration($tab, $user, $ui) {
    $params = $this->params;
    $exampleText = $params->get('exampleText', 'Text Parameter not set!');

    $ret = "\t<tr>\n";
    $ret .= "\t\t<td class='titleCell'>\".Example plugin warnings:\".</td>\n";
    $ret .= "\t\t<td class='fieldCell'>";

    $ret .= "ParameterText: ".$exampleText;

    $ret .= "<p>Be carefull: don't set password same as username !</p>";
    $ret .= "</td>";
    $ret .= "\t</tr>\n";

    $this->_addValidationJS( "\tif (me['username'].value == me['password'].value) {\n"
        ."\t  errorMsg += \"".html_entity_decode("Password must differ from
username!") ."\n\n\n"
        ."\t  me['password'].style.background = \"red\";\n"
        ."\t  iserror=1;\n"
        ."\t}\n");
// also see event: 'onBeforeUserRegistration', implemented here in function:
pluginExampleBeforeUserRegistration().
return $ret;
}

```

3.9 Special user plug-ins

There are two special classes of user plug-ins:

- Private Messaging Systems – capable plug-ins
- Menu plug-ins (core)

Special plug-ins have a dot “.” in their name. Do not use a dot in normal plug-ins.

3.9.1 PMS

CB supports multiple PMS plug-ins. In case more than one is published, they will be used each to send the PMS.

PMS plug-ins must start with “`pms.`”

3.9.2 Menu

The Menu plug-in is core CB stuff. Please don't touch this, except for the CSS-setable parts, which are part of the template plug-ins.

3.10 CB API

CB provides a User-Interface (UI) Application Programming Interface (API) for plug-ins. Various displays and user interactions are provided in a generic way by CB. These are detailed in the next paragraphs.

3.10.1 Menus

CB provides natively for a CB menu (this can be switched off by the admin). The admin can choose in the backend between a menu bar and a menu list format.

CB RC2 provides for one level of submenu only, but in the future more levels will be supported.

As displaying tabs method may in the future not systematically be called, there is a separate plug-in method in tab objects used by CB to ask plug-ins to register menus:

```
/**
 * Generates the menu and user status to display on the user profile by calling
back $this->addMenu
 * @param object tab reflecting the tab database entry
 * @param object mosUser reflecting the user being displayed
 * @param int 1 for front-end, 2 for back-end
 * @returns boolean : either true, or false if ErrorMSG generated
 */
function getMenuAndStatus($tab,$user,$ui) {
}
```

Plug-ins can add their own menu items using following CB API:

```
/**
 * Registers a menu or status item to a particular menu position
 * @param array a menu item like:
 // Test example:
 $mi = array(); $mi["_UE_MENU_CONNECTIONS"]["duplique"]=null;
 $this->addMenu( array(
 "position" => "menuBar" , // "menuBar", "menuList"
 "arrayPos" => $mi ,
 "caption" => _UE_MENU_MANAGEMYCONNECTIONS ,
 "url" => sefRelToAbs($ue_manageConnection_url) , // can also be
"<a ...>" or "javascript:void(0)" or ""
 "target"=> "" , // e.g. "_blank"
 "img" => null , // e.g. "<img
src='plugins/user/myplugin/images/icon.gif' width='16' height='16' alt='' />"
 "alt" => null , // e.g. "text"
 "tooltip" => _UE_MENU_MANAGEMYCONNECTIONS_DESC ,
 "keystroke" => null ) ); // e.g. "P"
 // Test example: Member Since:
```



```

$mi = array(); $mi["_UE_MENU_STATUS"]["_UE_MEMBERSINCE"]["dupl"]=null;
$dat = cbFormatDate($user->registerDate);
if (!$dat) $dat="?";
$this->addMenu( array(
    "position"    => "menuList" ,      // "menuBar", "menuList"
    "arrayPos"    => $mi ,
    "caption"     => $dat ,
    "url"         => "" ,              // can also be "<a ....>" or
                                      // "javascript:void(0)" or ""
    "target"     => "" ,              // e.g. "_blank"
    "img"        => null ,           // e.g. "<img
src='plugins/user/myplugin/images/icon.gif' width='16' height='16' alt='' />"
    "alt"        => null ,           // e.g. "text"
    "tooltip"    => _UE_MEMBERSINCE_DESC ,
    "keystroke"  => null ) ); // e.g. "P"
*/
function addMenu( $menuItem )

```

Menu additions sort themselves on a first come first served base, but get hierarchically sorted, depending on main menu name, for a submenu.

As example, this adds a menu for PMS if it's not the user's profile which is displayed:

```

global $my;
if ($my->id!=$user->id && $my->id > 0) {
    $pmsurl=...
    $mi = array();
    $mi["_UE_MENU_MESSAGES"]["_UE_PM_USER"]=null;
    $this->menuBar->addObjectItem($mi, _UE_PM_USER, sefRelToAbs($pmsurl), "",
    "", "", _UE_MENU_PM_USER_DESC, "");
}

```

It's important for SEF-compatibility to call **sefRelToAbs** for all internal URLs.

3.10.2 Status display

Status displays are similar to menus. They are setup using the same API:

Example:

```

if ($sbConfig['showranking'] && ($params->get('statRanking', '1') == 1) &&
($sbUserDetails != false)) {
    $mi = array(); $mi["_UE_MENU_STATUS"][$params->get('statRankingText',
"_UE_FORUM_FORUMRANKING")]["_UE_FORUM_FORUMRANKING"]=null;
    $this->addMenu( array(
        "position" => "menuList" ,      // "menuBar", "menuList"
        "arrayPos" => $mi ,
        "caption"  => $sbUserDetails->msg_userrank. ($params->get('statRankingImg',
'1')==1 ? $sbUserDetails->msg_userranking : "" ) ,
        "url"      => "" ,              // can also be "<a ....>" or
        "javascript:void(0)" or ""
        "target"   => "" ,              // e.g. "_blank"
        "img"     => null ,           // e.g. "<img
src='plugins/user/myplugin/images/icon.gif' width='16' height='16' alt='' />"
        "alt"     => null ,           // e.g. "text"
        "tooltip" => "" ) );
}

```

3.10.3 Forms

Forms in plug-ins must be coded using CB's forms support, so that a form in a plug-in returns to that plug-in and parameter names get coded for the plug-in.

For now, forms are supported only on user profiles.

Forms can be sent using either GET or POST HTTP methods.

The parameters are prefixed with the name of the plug-in so that multiple plug-ins do not collide on parameters with same name (e.g. "search" will become "**simpleboardtabsearch**").

All parameters, when passed through GET are correctly sefed.

A complete example for forms is the **JoomlaBoard/SimpleBoard** plug-in (user profile tab): it needs to be activated in the backend Joomlaboard/simpleboard tab parameters, to activate search and pagination and sorting functions.

3.10.3.1 URL support

From plugin.foundation.php (lines 2476-2536) we find the following CB framework methods to render CB related URLs:

```

$ _CB_framework->userProfileUrl( $userId = null, $htmlSpecials = true, $tab = null,
$format = 'html' )

$ _CB_framework->userProfileEditUrl( $userId = null, $htmlSpecials = true, $tab = null,
$format = 'html' )

$ _CB_framework->userProfilesListUrl( $listId = null, $htmlSpecials = true, $searchMode =
null, $format = 'html' )

$ _CB_framework->viewUrl( $task, $htmlSpecials = true, $formId = null, $format = 'html' )

```

All URLs should be generated whenever possible in lower-case, but **should always be decoded also in lower-case**, as some SEF and URL rewriting tools allow the lowercasing of them.

URLs referring to the plug-in must always be generated using the CB API method:

```

/**
 * gives the URL of a link with plugin parameters.
 * @param array of string with key name of parameters
 * @param string cb task to link to (default: userProfile)
 * @param boolean TRUE to call sefRelToAbs (default), FALSE to leave URL unsefed
 * @param array of string with keys of parameters to not include
 * @return string value of the parameter
 */

```

```
function _getAbsURLwithParam($paramArray, $task="userProfile", $sefed=true,
$excludeParamList=array())
```

Parameters received can be taken back using following CB API:

```
/**
 * gets an ESCAPED and urldecoded request parameter for the plugin
 * @param string name of parameter in REQUEST URL
 * @param string default value of parameter in REQUEST URL if none found
 * @param string postfix for identifying multiple pagings/search/sorts (optional)
 * @return string value of the parameter (urldecode processed for international and
special chars) and ESCAPED! and ALLOW HTML!
 *          you need to call cbUnEscapeSQL to remove escapes, and htmlentities
before displaying.
 */
function _getReqParam($name, $def=null, $postfix="")
```

Please note that the parameters (e.g. \$paramArray array) are escaped using addslashes() . So stripslashes() should be used if unescaping is needed, and htmlentities(stripslashes()) for outputing into html.

The developer is **strongly encouraged** to test his plugin with both ON and OFF of gpc_magic_quotes PHP setting.

Characters to test on URLs, fields and SQL accesses include at least:

- single-quote ‘
- and double-quote “,
- escaped quotes \' \”
- as well as single and double backslash \ \\\,
- as well as few accented characters,
- 1-byte UTF-8 characters like à è à ü ö ä,
- and multi-byte ones as well (e.g. ñ which is %C3%B1 in UTF-8),
- and also newline escapes \n.

One of our test-string is `héllö' 'world""bàck\släsh\\ñ\n\' \'co\""` . This kind of test-string should make a safe and uncorrupted journey through your plugin workflow...

These tests are not only for beauty and usability, **but also for SQL injection-safety...**

The following gives any GET/POST parameter name using the CB prefixing:

```
/**
 * gets the name input parameter for search and other functions
 * @param string name of parameter of plugin
 * @param string postfix for identifying multiple pagings/search/sorts (optional)
 * @returns string value of the name input parameter
```

```
*/
function getPagingParamName($name="search", $postfix="")
```

An example from the PMS plug-in tab:

```
if (isset($_POST[$this->_getPagingSearchName("sndnewmsg")]) && $_POST[$this->_getPagingSearchName("sndnewmsg")] == _UE_PM_SENDMESSAGE) {
    $sender = $this->_getReqParam("sender", null);
    $recip = $this->_getReqParam("recip", null);
    if ($sender && $recip) {
        $newsb = htmlentities($this->_getReqParam("newsb", null)); //urldecode
done in _getReqParam
        $newmsg = htmlentities($this->_getReqParam("newmsg", null)); //don't allow
html input on user profile!
```

The developer is strongly advised to protect his forms from SQL injection attacks. Same also from bots by implementing a protection code, as done in the PMS plug-in.

The following API allows generating a proper URL for forms and links:

```
/**
 * gives the URL of a link with plugin parameters.
 * @param array of string with key name of parameters
 * @param string cb task to link to (default: userProfile)
 * @param boolean TRUE to call sefRelToAbs (default), FALSE to leave URL unsefed
 * @param array of string with keys of parameters to not include
 * @return string value of the parameter
 */
function _getAbsURLwithParam($paramArray, $task="userProfile", $sefed=true,
$excludeParamList=array())
```

In Quick PMS tab for instance this is used as follows:

```
$base_url = $this->_getAbsURLwithParam(array());
$ret .= '<form method="post" action="'. $base_url. '>';
...
$ret .= '<input type="text" name="'. $this->_getPagingSearchName("newsb") .'"
size="'. $width. '" value="'. $newsb. '" class="inputbox" />
...'
```

3.10.4 Generic list support

The CB API supports a list in the plug-in tab with paging, sorting and searching facilities.

A generic API method allows getting all relevant parameters for paging, sorting and searching, along with other plug-in specific parameters into an array of strings:

```
/**
 * gets the paging limitstart, search and sortby parameters, as well as
additional parameters
 * @param array of string : keyed additional parameters as "Param-name" =>
"default-value"
```

```
* @param mixed : array of string OR string : postfix for identifying multiple
pagings/search/sorts (optional)
* @return array("limitstart" => current list-start value (default: null), "search"
=> search-string (default: null), "sortby" => sorting by, +additional parameters as keyed
in $additionalParams)
*/
function getPaging($additionalParams=array(), $postfixArray=null)
```

This is called for instance as follows:

```
$pagingParams = $this->_getPaging();
```

These can then be used and set simply like this:

```
if ($pagingParams["limitstart"] === null) $pagingParams["limitstart"] = "0";
```

or when multiple lists are displayed on the plugin tab:

```
$pagingParams = $this->_getPaging(array(), array("posts_", "subscriptions_"));
```

These can then be used and set simply like this:

```
if ($pagingParams["posts_limitstart"] === null) $pagingParams["posts_limitstart"] = "0";
```

The whole array can be initialized using this function:

```
/**
 * sets the paging limitstart, search and sortby parameters, as well as additional
parameters
 * @param array of string : keyed additional parameters as "Param-name" => "value"
 * @param string search string
 * @param string sorting parameter added as &sortby=... if NOT NULL
 * @param array of string : keyed additional parameters as "Param-name" =>
"default-value"
 * @param string postfix for identifying multiple pagings/search/sorts (optional)
 * @return array("limitstart" => current list-start value (default: null), "search"
=> search-string (default: null), "sortby" => sorting by, +additional parameters as keyed
in $additionalParams)
*/
function
_setPaging($limitstart="0", $search=null, $sortBy=null, $additionalParams=array(),
$postfix="")
```

3.10.4.1 Pagination

Part of the list framework, CB API supports a powerful, yet simple, paging API using this function:

```
/**
 * Writes the html links for pages inside tabs, eg, previous 1 2 3 ... x next
 * @param array: paging parameters. ['limitstart'] is the record number to start
displaying from will be ignored
 * @param string postfix for identifying multiple pagings/search/sorts (optional)
```

```
* @param int Number of rows to display per page
* @param int Total number of rows
* @param string cb task to link to (default: userProfile)
* @return string html text displaying paging
*/
function _writePaging($pagingParams, $postfix, $limit, $total,
$task="userProfile")
```

A pagination line is simply written as follows for Joomlaboard/SimpleBoard tab:

```
if ($pagingEnabled && ($postsNumber < $total)) {
    $return .= "<div style='width:95%;text-align:center;'>"
    .$this->_writePaging($pagingParams,"posts_", $postsNumber,$total)
    ."</div>";
}
```

The total number of post is found with a separate SQL query taking in account the search criteria, and maximum amount of posts shown.

3.10.4.2 Search

Together with pagination functions (but also independently), the CB API provides a simple search box to be drawn by a plug-in:

```
/**
 * Writes the html search box as <form><div><input ....
 * @param array: paging parameters. ['limitstart'] is the record number to start
 displaying from will be ignored
 * @param string postfix for identifying multiple pagings/search/sorts (optional)
 * @param string the class/style for the div
 * @param string the class/style for the input
 * @param string cb task to link to (default: userProfile)
 * @return string html text displaying a nice search box
 */
function
_writeSearchBox($pagingParams,$postfix="", $divClass="style=\"float:right;\"", $inputClass=
"class=\"inputbox\"", $task="userProfile")
```

This is implemented for instance as follows (example is Joomlaboard/SimpleBoard):

```
if ($searchEnabled) {
    $searchForm = $this->_writeSearchBox( $pagingParams, "posts_",
$divClass="style='float:right;', $inputClass="class='inputbox'");
} else {
    $pagingParams["search"] = "0";
}
```

3.10.4.3 Sorting

Together with pagination and search functions (but also independently), the CB API provides a simple sorting fields header to be drawn by a plug-in, calling for each header this method:

```
/**
 * Writes the html links for sorting as headers
 * @param array: paging parameters. ['limitstart'] is the record number to start
 displaying from will be ignored
 * @param string postfix for identifying multiple pagings/search/sorts (optional)
 * @param string sorting parameter added as &sortby=... if NOT NULL
 * @param string Name to display as column heading/hyperlink
 * @param boolean true if it is the default sorting field to not output sorting
 * @param string class/style html for the unselected sorting header
 * @param string class/style html for the selected sorting header
 * @param string cb task to link to (default: userProfile)
 * @return string html text displaying paging
 */
function _writeSortByLink($pagingParams, $postfix, $sortBy, $sortName,
 $defaultSort=false, $unselectedClass='class="cbSortHead"',
 $selectedClass='class="cbSortHeadSelected"', $task="userProfile")
```

The table header is typically written as follows in Joomlaboard/SimpleBoard tab to be sortable:

```
$return .= "\n\t<tr class=\"sectiontableheader\">";
$return .= "<th>".$this->_writeSortByLink($pagingParams, "posts_", "date",
 _UE_FORUMDATE,true)."</th>";
$return .= "<th>".$this->_writeSortByLink($pagingParams, "posts_", "subject",
 _UE_FORUMSUBJECT)."</th>";
$return .= "<th>".$this->_writeSortByLink($pagingParams, "posts_", "category",
 _UE_FORUMCATEGORY)."</th>";
$return .= "<th>".$this->_writeSortByLink($pagingParams, "posts_", "hits",
 _UE_FORUMHITS)."</th>";
$return .= "</tr>";
```

The user clicking on such a link will generate a POST/GET HTTP request, giving a **PLUGINTABNAMEsortby** parameter, to be collected using **_getPaging** method.

Parameters for sorting are used as follows (example in Joomlaboard/SimpleBoard tab) to generate a “ORDER BY” SQL request:

```
switch ($pagingParams['posts_sortby']) {
    case "subject":
        $order = "a.subject ASC, a.time DESC";
        break;
    case "category":
        $order = "b.id ASC, a.time DESC";
        break;
    case "hits":
        $order = "c.hits DESC, a.time DESC";
        break;
    case "date":
        $order = "a.time DESC";
        break;
    default:
        $order = "a.time DESC";
        break;
}
```

3.10.4.4 Other form inputs

Other forms can be built using any combination of the above methods. A typical example is shown in the PMS plug-in Quick PMS tab.

3.10.5 User lists support

This is not supported yet. But will in the future.

3.10.6 User search support

This is not supported yet. But will in the future.

3.10.7 Language support for plug-ins

Plug-ins can use their own language files. The recommended way is to have a language directory with inside a `default_language.php` file, and language files for each supported language.

Uploading plug-in language files is not supported yet. We are waiting for Joomla! new language support before doing more work in this area.

3.11 Integrating with other components

To integrate well with other components, several ways are possible.

3.11.1 Talk with the others

A basic rule to integrate is to talk with the other dev team, and get their commitment if possible, so that future interoperability and backwards compatibility are maintained.

3.11.2 Preferred way: clean API

We recommend clean, abstract, encapsulated, object-oriented interfaces.

The **YaNC** interface is an example written by the CB authors in cooperation with the **YaNC** maintainer. Please note that YaNC integration only supports Joomla 1.0.x YaNC extensions (there is no 1.5.x version of YaNC).

3.11.3 Other way: through SQL tables

An alternate way is through SQL tables accesses, if this structure is known to be critical by both teams (plug-in and the other components' team).

4 Reference Snippets

4.1 Cheat sheet

All your CB plugin php files should start with the following define statement:

```
if ( ! ( defined( '_VALID_CB' ) || defined( '_JEXEC' ) || defined( '_VALID_MOS' ) ) ) { die( 'Direct Access to this location is not allowed.' ); }
```

| instead of: | use: |
|--|---|
| global \$mainframe | global \$ CB framework |
| global \$database | global \$ CB database |
| \$mosConfig XXX | \$ CB framework->getCfg('xxx') |
| \$mainframe->getCfg('live site') | \$ CB framework->getCfg('live site') |
| \$mosConfig live site | \$ CB framework->getCfg('live site') |
| \$my | see cases below |
| \$my->id | \$ CB framework->myId() |
| \$my->username | \$ CB framework->myUsername() |
| \$my->gid | \$ CB framework->myCmsGid() |
| sefRelToAbs(| cbSef(|
| mosRedirect(| cbRedirect(|
| mosgetParam(| cbGetParam(|
| mosArrayToInts | cbArrayToInts |
| mosUser | moscomprofilerUser |
| mosTabs | cbTabs |
| mosFormatDate(\$date) | cbFormatDate(\$date, \$serverTimeOffset = 1, \$showtime = true) |
| editorArea('editor'.\$oName, \$oValue, \$oName, 600, 350, \$oCols, \$oRows) | \$ CB framework->displayCmsEditor(\$oName, \$oValue, 600, 350, \$oCols, \$oRows) |
| mosHTML::makeOption('_UE_YES', _CMN_YES); | moscomprofilerHTML::makeOption('_UE_YES', _UE_YES); but better with Int in DB: moscomprofilerHTML::makeOption('1', _UE_YES); |
| \$lists['_pg_AccessMode'] = mosHTML::selectList(\$accessMode, \$this->_getPagingParamName("cb_pgaccessmode"), 'class="inputbox" size="1" mosReq="0" mosLabel="'._pg_AccessMode.'"', 'value', 'text', \$user->cb_pgaccessmode); | \$lists['_pg_AccessMode'] = moscomprofilerHTML::selectList(\$accessMode, \$this->_getPagingParamName("cb_pgaccessmode"), 'class="inputbox" size="1" mosReq="0" mosLabel="'._pg_AccessMode.'"', 'value', 'text', \$user->cb_pgaccessmode, 2); (see the ,2 added at end to avoid blank choice at begin). |
| \$_VERSION | checkJVersion() |

```
$ CB framework->check_acl( 'canManageUsers', $ CB framework->myUserType() )
$ CB framework->check_acl( 'canManageUsers', $ CB framework->myUserType() )
: translation table for mambo/joomla 1.0:
```

```
array( 'canEditUsers' => array( 'administration', 'manage', 'users', null,
'components', 'com_users' ),
'canBlockUsers' => array( 'administration', 'edit', 'users', null, 'user
properties', 'block_user' ),
'canReceiveAdminEmails' => array( 'workflow', 'email_events', 'users',
null ),
'canEditOwnContent' => array( 'action', 'edit', 'users', null, 'content',
'own' ),
'canAddAllContent' => array( 'action', 'add', 'users', null, 'content', 'all'
),
'canEditAllContent' => array( 'action', 'edit', 'users', null, 'content',
'all' ),
'canPublishContent' => array( 'action', 'publish', 'users', null,
'content', 'all' ),
'canInstallPlugins' => array( 'administration', 'install', 'users', null,
'components', 'all' ),
'canManageUsers' => array( 'administration', 'manage', 'users', null,
'components', 'com_users' )
);
```

Backend

```
require_once( $_CB_joomla_adminpath . "/includes/pageNavigation.php" );
$pageNav = new mosPageNav( $total, $limitstart, $limit );
```

```
cbimport( 'cb.pagination' );
$pageNav = new cbPageNav( $total, $limitstart, $limit );
```

CB 1.2 Stable news

```
addCbHeadTag( $ui, "<script type=\"text/javascript\" src=\"".$_CB_framework-
>getCfg( 'live_site' ) . '/components/com_comprofiler/js/menubest' . (
$_CB_framework->getCfg( 'debug' ) ? ' : '.pack' ) . ".js\"></script>\n" );
$_CB_framework->document->addHeadScriptUrl(
'/components/com_comprofiler/js/menubest.js', true );
```

```
addCbHeadTag( $ui, '<link type="text/css" rel="stylesheet" href="' .
selectTemplate($ui) . $templateFile . '" . ( $media ? ' media="' . $media . '"
: ' ) . ' />' );
$_CB_framework->document->addHeadStyleSheet( selectTemplate( $ui ) .
$templateFile, false, $media );
```

4.2 Including CB Framework

You can use the CB Online module to see how the CB framework should be invoked from a module.

That's how the CB API (framework) is included natively cross-platforms:

```
/**
 * CB framework
 * @global CBframework $_CB_framework
 */
global $_CB_framework, $_CB_database, $ueConfig, $mainframe;
if ( defined( 'JPATH_ADMINISTRATOR' ) ) {
if ( ! file_exists( JPATH_ADMINISTRATOR .
'/components/com_comprofiler/plugin.foundation.php' ) ) {
echo 'CB not installed';
```

```
return;
}
include_once( JPATH_ADMINISTRATOR .
'/components/com_comprofiler/plugin.foundation.php' );
} else {
if ( ! file_exists( $mainframe->getCfg( 'absolute_path' ) .
'/administrator/components/com_comprofiler/plugin.foundation.php' ) ) {
echo 'CB not installed';
return;
}
include_once( $mainframe->getCfg( 'absolute_path' ) .
'/administrator/components/com_comprofiler/plugin.foundation.php' );
}
```

the link to logged-in user's profile:

```
cbSef( 'index.php?option=com_comprofiler&task=userProfile' .
getCBprofileItemid( true, false ) )
```

this will produce an htmlspecialchars escaped string if you need it for redirect or javascript, you don't want htmlspecialchars in that case:

```
cbSef( 'index.php?option=com_comprofiler&task=userProfile' .
getCBprofileItemid( true, false ), false )
```

link to profile for a given user:

```
cbSef( 'index.php?option=com_comprofiler&task=userProfile&user=' . (
(int) $userid ) . getCBprofileItemid( true, false ) )
```

for registration:

```
cbSef( 'index.php?option=com_comprofiler&=registers' )
```

for login:

```
cbSef( 'index.php?option=com_comprofiler&task=login' )
```

for logout:

```
cbSef( 'index.php?option=com_comprofiler&task=logout' )
```

(this directly logs out)

for default users-list:

```
cbSef( 'index.php?option=com_comprofiler&task=usersList' )
```

joomla 1.5 will find the Itemid if there is a corresponding userslist defined in menus

(always add false for non-htmspecialchared).

4.2.1 Logging changes:

```
cbimport( 'cb.field' );
$myId = $_CB_framework->myId();
$targetUserId = 64; // put whatever message got replied to, or NULL
$oldValues = 2352; // could be id of the post to which user replies, or NULL
$newValues = 'New subject'; // would be new post subject.
$message = 'New post body'; // would be new post body.
$reason = 'forum';
$event = 'postnew'; // for new thread
$event = 'postreply'; // for a reply to an existing thread
$event = 'postedit'; // for editing an existing post
$external_id = 2354; // would be the unique post id
$external_url = 'index.php?...'; // the full url to the post, including the
#2354 part (not sefed)
if ( $myId ) {
    $cbUser =& CbUser::getInstance( $myId );
    if ( $cbUser !== null ) {
        $user =& $cbUser->getUserData();
        global $_PLUGINS;
        $_PLUGINS->loadPluginGroup('user');
        $_PLUGINS->trigger( 'onLogChange', array( 'update', 'forum', 'post',
&$user, null, null, $oldValues, $newValues, $reason, $message, $targetUserId,
$external_id, null, 'com_kunena', $external_url ) );
        $_PLUGINS->trigger( 'onLogStore', array( &$user ) );
    }
}
```

4.2.2 PMS to user link:

```
cbimport( 'cb.field' );
cbimport( 'language.front' );
outputCbTemplate( $_CB_framework->getUi() );
$toId = 64; // replace by to whom to PM
global $_CB_PMS;
$resultArray = $_CB_PMS->getPMSlinks( $toId, $_CB_framework->myId(), '', '', 1)
; // toid,fromid,subject,message,1: link to compose new PMS message for
$toId user.
$html = null;
if ( count( $resultArray ) > 0 ) {
    $pmIMG = '';

    foreach ( $resultArray as $res ) {
        if ( is_array( $res ) ) {
            $linkItem = getLangDefinition( $res["caption"] );
            // or:
```

```
$linkItem = $pmIMG;
$html .= '<a href="' . cbSef( $res["url"] ) . '" title="' .
getLangDefinition( $res["tooltip"] ) . '">' . $linkItem . '</a>';
}
}
}
echo $html;
```

4.2.3 Generate HTML code to display avatar of a user:

```
cbimport( 'cb.field' );
cbimport( 'language.front' );
outputCbTemplate( $_CB_framework->getUi() );
$myId = $_CB_framework->myId();
if ( $myId ) {
    $cbUser =& CbUser::getInstance( $myId );
    if ( $cbUser !== null ) {
        $thumbnailAvatarHtmlWithLink = $cbUser->getField( 'avatar', null, 'html',
'none', 'list' );
        $bigAvatarHtmlWithLink = $cbUser->getField( 'avatar' );
        echo $thumbnailAvatarHtmlWithLink . '<br />' . $bigAvatarHtmlWithLink;
    }
}
*/
```

5 Conclusions

The Community Builder 1.4 Plugin API is a generic API made for helping integrating user-centric components .

It will be further developed, with backwards compatibility in mind. The authors can't guarantee that this will always be the case.

Your feed-backs, improvement proposals and plugin developments are highly welcome in the Community Builder Community.

Have fun developing plugins

– The Community Builder Team.