# OVIDENTIA

# Developer's Guide

**Noureddine Ayachi**

**Ludo Rousseau**

| ***Version Documentation*** | ***Version OVIDENTIA / Module*** | ***Date*** |
|---|---|---|
| 1.4 | 5.6.3 | 16 September 2005 |
| 1.5 | 5.6.3 | 22 November 2005 |
| 1.6 | 5.6.3 | 23 Februar 2006 |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

# Table of contents

# Preface

This guide documents the development of Ovidentia and contains information for version 5.6.3 or later. Be aware that this is developer's documentation and some of the functions described here are planned in some faraway future, or will never make it in any distribution. And changes can frequently occur in functions that are in an early development status.

Today the Internet has become a business tool with added value for your organization. By making information and services easily available, companies and organizations can ensure that all the aspects of their business are working together. Portals are the way to communicate with partners, customers, employees and the public.

The enterprise portal becomes a unique entry point for services (applications, work group functions, work flow ...) and information (Notes, FAQ, Articles, News, information from the Internet ...). A portal site achieves the following goals:

- Stimulate cooperation;
- Support decision making functions;
- Ensure access to company applications.

Most available portal solutions don't offer all the possible answers and don't meet all the corporate needs. In general these products are very specialized in a chosen domain.

Therefore Koblix [http://www.koblix.com] has joined forces with Cantico [http://www.cantico.fr, http://www.ovidentia.org] for the development of Ovidentia [http://www.ovidentia.be], a powerful web portal software aimed at building a tool that allows for the fast development of a company portal according to the above mentioned criteria. We have decided to work according to the principle of "open source" in order to:

- ensure the future availability of the solution;
- multiply the resources for development;
- minimize the costs of implementation for the final customer.

# 1 - Introduction

The purpose of this part is to aid developers in developing add-ons for Ovidentia. An Ovidentia add-on is a packaged module that can be delivered with an Ovidentia distribution and that enhances Ovidentia with new functionalities.

Ovidentia add-ons use the same methods used to develop the Ovidentia product and inherit functions and global variables used by Ovidentia. In particular, add-ons can use Ovidentia templates to separate PHP code from HTML sources. And add-ons also use the multilanguage facilities available in Ovidentia.

Ovidentia templates are a facility for separating HTML from PHP code. Template files are ordinary text files that contain HTML tags interlaced with specific Ovidentia tags. Ovidentia parses template files and replaces Ovidentia tags with values. The idea behind this is to let the developer concentrate only on PHP code while the web designer can edit the template files in a WYSIWYG tool. Also, HTML developers can offer different skins by modifying only the template and style sheet files.

This guide provides learning by action and the developer is guided through the implementation of an Ovidentia add-on. In this way the developer becomes familiar with add-ons in Ovidentia and how to develop them.

Readers must be familiar with the PHP language and Ovidentia objects (sections, groups, users ...). You can find additional documentation on Ovidentia at the Ovidentia community site : http://www.ovidentia.org. The essential documents are the Administrator Guide, the User Guide, the Installation Guide and the OvML Language document.

Before going further, let’s review how to use Ovidentia templates.

# 2 - Templates

A template file can contain multiple templates. Each template is delimited and recognized by specific tags: *template delimiter tags*.

Their syntax is of the form:

```
<!--#begin template-name -->
  ....
<!--#end template-name -->
```

where template-name is a *template name* (the name of your template).

(Remark: there is a blank space before --> )

To print this template, the PHP developer must write a PHP class and call the function bab_printTemplate:

```
bab_printTemplate($class, $file, $template);
```

where:

$class: the PHP class that holds all variable data for the template

$file: the name of the template file, including the .html extension, such as mytemplatefile.html.

$template: *template name* as used in *template delimiter tags*

Ovidentia parses the template, makes a substitution with the value of the variables provided by the PHP class $class of the variables in the text between the template delimiter tags and prints the result.

First we provide a reference for Ovidentia tags and describe their syntax, and then we give an example to illustrate how to use them.

Utiliser des fichiers ovml propres au module :

L'url d'accès à un fichier ovml présent dans le skin courante est la

suivante :

index.php?tg=oml&file=nom du fichier.ovml

Il est possible d'ajouter des fichiers ovml propres à un module.

Chemin dans le fichier zip du module : skins\ovidentia\ovml.

A l'installation du module, les fichiers ovml présents dans le fichier zip

existeront dans ce chemin : noyau d'Ovientia\skins\ovidentia\addons\nom du

module\....ovml

Syntaxe pour lancer un fichier ovml propre à un module :

index.php?tg=oml&file=addons\nom du module\nom du fichier.ovml

**Use ovml files appropriate for the add-on:**

The url of access to a ovml file present in current skin is the following one:

index.php?tg=oml&file=name of ovml file

It is possible to add ovml files appropriate for a add-on. Path in the file zip of the add-on:

skins\ovidentia\ovml\

After the installation of the add-on, the ovml files present in the file zip will exist in this path:

core of Ovientia\skins\ovidentia\addons\name of add-on\

Syntax to launch an ovml file appropriate for a add-on:

index.php?tg=oml&file=addons\prefix of add-on\name of ovml file

## *2.1 - Ovidentia Tags Syntax*

The syntax used to indicate text substitution is of the form:

{ var1 }

where var1 identifies a variable named var1 in PHP code. This tag is used to substitute variable data into text.

As example look at this template:

```
<b>I say: <font color=red>{ var1 }</font>
```

If in PHP code, var1 contains "Hello World!", this will output:

```
<b>I say: <font color=red>Hello World!</font>
```

Sometimes, the text to be included is dependent upon the value of some other data. The *if* tag is provided to support insertion of text based on the value of a variable. An expression is evaluated and if the value is true, the text is inserted, otherwise nothing happens. The *if* syntax is as follows:

```
<!--#if var1 "expression" -->
    .....
<!--#else var1 -->
    .....
<!--#endif var1 -->
```

(Remark: always put a space before -->)

where expression has one of the following forms:

```
== value
!= value
>= value
<= value
> value
< value
```

For example:

```
<!--#if lang "== english" -->
    <b>I say: <font color=red>Hello World</font>
<!--#else lang -->
    <b>Je dis: <font color=red>Bonjour tout le monde</font>
<!--#endif lang -->
```

will output, if lang equals "english"

```
<b>I say: <font color=red>Hello World</font>
```

otherwise

```
<b>Je dis: <font color=red>Bonjour tout le monde</font>
```

You can also test a variable with true/false like this

```
<!--#if bClosed -->
      <b>bClosed variable is true</font>
  <!--#endif bClosed -->
```

Often, it is necessary to insert a sequence of text, like table rows or a selected list from an SQL request for example. For this, you use the *in* tag:

```
<!--#in function-to-call -->
.....
<!--#endin function-to-call -->
```

where function-to-call is the name of the *function to call*. At each call, if the function returns 'true', variable substitution is made and the result is printed, otherwise nothing happens.

As an example let's see how to list files in a folder:

In the template file test.html:

```
<!--#begin mytemplate -->
   <table>
          <!--#in getfile -->
                 <tr><td><b>{ filename }</b><br></td></tr>
          <!--#endin getfile -->
   </table>
<!--#end mytemplate -->
```

In the PHP code, a method getfile must be defined in a class like this:

```
class myClass
  {
  var $tab = array();
  var $filename;
  var $count;

  function myClass()
      {
      // fill $tab with file names
      $this->tab[] = "file1.txt";
      $this->tab[] = "file2.doc";
      $this->tab[] = "file1.gif";
      $this->tab[] = "file1.html";
      $this->count = sizeof($this->tab);
      }

  function getfile()
      {
      static $i = 0;
      if( $i < this->count)
          {
          $this->filename = $this->tab[$i];
          $i++;
          return true;
          }
      else
          return false;
      }
  }

  // instantiate class and call babPrintTemplate function
  $cl = new myClass();

  bab_printTemplate( $cl, "test.html", "mytemplate");
```

This will output:

```
<table>
    <tr><td><b>file1.txt</b><br></td></tr>
    <tr><td><b>file2.doc</b><br></td></tr>
    <tr><td><b>file1.gif</b><br></td></tr>
    <tr><td><b>file1.html</b><br></td></tr>
</table>
```

In this example, if for some reason the $tab array is empty, the previous code will produce an empty table:

```
<table>
</table>
```

To avoid displaying an empty table, you can use the *if* tag in your template and test the number of elements in the $tab array:

```
<!--#begin mytemplate -->
   <!--#if count "> 0" -->
          <table>
          <!--#in getfile -->
                 <tr><td><b>{ filename }</b><br></td></tr>
          <!--#endin getfile -->
          </table>

   <!--#else count -->
          There are no files !
   <!--#endif count -->
<!--#end mytemplate -->
```

## *2.2 - Example of how to create an add-on: the favorites add-on*

Now we know how to use Ovidentia templates, let's see how to create an add-on. Suppose we want to provide an add-on that allows registered users (users that have an account on the site) to store their favorite URLs. For this, we want to add a menu item Favorites in the "User's section". This menu item appears only for registered users and enables them to list their favorites and add URLs to their list of favorites. We will give our add-on the name "favorites".

To achieve this goal, the following tasks are needed:

- Create a database table to store the user's favorite URLs
- Implement the "Add-on Skeleton"
- Add functionalities to your add-on

# 3 - Create a database table to store the user's favorites

We need a database table to store the user's favorite URLs. Therefore we create a table favorites_list with four fields: id ( an index ), id_owner ( user id ), url and the URL's description. The user id (column id_owner) is the identification of the user who owns the URLs. This id is given by Ovidentia ( See later ). Use the following SQL request to create the favorites_list table:

```
CREATE TABLE `favorites_list` (
`id` INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
`id_owner` INT(11) UNSIGNED NOT NULL,
`url` TINYTEXT NOT NULL,
`description` TEXT NOT NULL,
PRIMARY KEY (`id`)
);
```

Remarks:

1. **You must prefix database table names used by your add-on**, by the name of your add-on. This will avoid name clashes. It also will automate the deletion of the database tables added by your add-on because when the administrator will click the icon to delete our add-on all tables prefixed with the name of your add-on will be deleted. See the chapter “How to package your add-on for distribution" for more details on this.
That's why our table favorites_list is prefixed with "favorites_". Ovidentia developers reserved the "bab_" prefix for themselves and Koblix uses the "ko_" prefix. See the appendix "Reserved prefixes used to avoid name clashes" for a complete list of reserved prefixes.

2. Create the tables for your add-ons in the Ovidentia database. This will make their backup easier.

# 4 - Implementation of a skeleton add-on

To do this, we need to create folders and some files for your add-on. For this you need to do the following:

- Create a folder for your add-on's program files
- Implement some initialization functions needed by Ovidentia
- Create a folder for your add-on's template files
- Add the code needed by your add-on

In the text that follows, $babInstallPath contains the path to the Ovidentia *distribution folder* as defined in config.php (See the Ovidentia Global Variables chapter).

Ovidentia knows about add-ons that can be loaded by looking at the presence of folders in its addons folder.

## *4.1 - Development and packaging*

The development of an add-on is done through the creation of the required files (.php, .html, .css …) in a subfolders of the Ovidentia folders.

See the following chapters:

- Create a folder for your add-on's PHP files
- Create a folder for the language files for the add-on
- Create folders for your add-on template files, style files and images

There are two possibilities to package an add-on for distribution. The preferred method is the first one as this method is more automated and can be done through the Ovidentia user interface.

Method 1: once the add-on has been developed the Ovidentia user interface allows to create the .zip file automatically by clicking “Download" in the Administration section, following the link Modules.

Method 2: after the add-on has been developed you can manually create the tree structure with your add-on files and put them in a .zip file. In order to test the distribution it is sufficient to load the add-on through the Ovidentia interface. See the chapter “How to package your add-on for more information.

## *4.2 - Create a folder for your add-on's PHP files*

Now we create a folder that houses our add-on's PHP source files. This folder must be located in $babInstallPath/addons/ folder and must have the same name as your add-on, such as used as prefix for our favorites_list table.

So in our example this is the favorites folder :

```
$babInstallPath/addons/favorites
```

We call this folder the *add-on program folder*.

We will create the following files in this folder:

- description.html
- addonini.php
- init.php
- main.php
- history.txt

In the next sections a description is given for the addonini.php, init.php and main.php script files. For a description of the description.html file and the history.txt file, see the chapter "How to document your add-on" further on in this guide.

### *4.3 - Make your add-on known to Ovidentia with the addonini.php script file*

The addonini.php file must be present and has the following structure:

```
; <?php DO NOT REMOVE THIS LINE
; Comments start with ';'
[general]
version="2.0"
description="Favorites example add-on"
delete="1"
db_prefix="favorites_"
ov_version="5.5.1"
author="unknown"
; DO NOT REMOVE THIS LINE ?>
```

For every add-on we write we can keep an exact copy of this file in which we only change the following lines:

version : Put the version number of your add-on here.

description : Put a short description of your add-on here. This description will appear on the list of available add-ons in the Administration section "Add-ons" link.

delete : If delete = 1, Administrators can delete add-on from Add-ons list. Otherwise, they can't delete add-on. See db_prefix.

db_prefix : Prefix used by your add-on in creation of database tables. This let Ovidentia remove all tables owned by your add-on, when your add-on is removed.

ov_version : indication from which Ovidentia version on this add-on may be installed. This information is taken into account from Ovidentia version 5.5.6.

author : author's name. Optional.

### *4.4 - Implement some initialization functions needed by Ovidentia in the init.php script file*

In the folder $babInstallPath/addons/favorites, create a file init.php that will handle the initialization tasks for your add-on.

The full path name of this file is obtained as the concatenation of the value of the $babInstallPath variable, the string "addons/", the name of the folder where your add-on is stored, and finally the name of the PHP file that contains your add-on initialization code init.php. For our example this results in:

ovidentia/addons/favorites/init.php

Copy the following code in this PHP file:

```
<?php
function favorites_getAdminSectionMenus(&$url, &$text)
{
return false;
}

function favorites_getUserSectionMenus(&$url, &$text)
{
return false;
}

function favorites_onUserCreate( $id )
{
}

function favorites_onUserDelete( $id )
{
}

function favorites_onGroupCreate( $id )
{
}

function favorites_onGroupDelete( $id )
{
}

function favorites_onSectionCreate(&$title, &$content)
{
return false;
}

function favorites_upgrade($version_base,$version_ini)
{

return false;
}

function favorites_onPackageAddon()

return true;
}

function favorites_onDeleteAddon()


return true;

}

?>
```

### 4.4.1 - Tip

To avoid name clashes with other add-on functions, and in order to make it possible for Ovidentia to retrieve your functions, you need to prefix your functions with your add-on name.

Let's explain how your add-on can use those functions.

If your add-on needs administration functions, you probably want to add a menu item to the Administration section. This menu item will let Ovidentia administrators configure your add-on, give access rights to groups, ... To do this, Ovidentia gives you a chance to add menu items in this section by calling your function favorites_getAdminSectionMenus(&$url, &$text) where $url and $text are passed by reference. Your application must initialize $url (the URL requested by the menu item) and $text ( text displayed for this menu item in the Administration section ). With this function you can add multiple menu items, as long as the function returns true. When there is no menu item to append anymore, the function must return false. If you don't need to add a menu item at all to the Administration section, simply return false.

For our example we don't need to add a menu item for administration purposes and so we don't change anything in this function.

In the same manner, you can add menu items in the User's Section by returning true from the favorites_getUserSectionMenus function and after initializing $url and $text.

For our example we need to add the menu item favorites to the User's Section menu:

```
function favorites_getUserSectionMenus(&$url, &$text)

{

static $nbMenus=0;

if( !$nbMenus && !empty($GLOBALS['BAB_SESS_USERID']))

 {

 $url = $GLOBALS['babAddonUrl']."main";

 $text = bab_translate("Favorites", "favorites");

 $nbMenus++;

 return true;

 }

return false;

}
```

First, remark how we use the $nbMenus variable to hold how many times Ovidentia calls this function. As we need to add only one menu item, the function returns true when $nbMenus is zero. Otherwise it returns false. Also, we use the $GLOBALS['BAB_SESS_USERID'] global variable to see if the current user is a registered user. It's empty if the user is not registered otherwise it contains the user id that Ovidentia uses to store the user's information in the Ovidentia database (See the Ovidentia global variables chapter).

Then we set $url to the value of the URL that will be requested when a user clicks our new menu item in the "Administration" section. All add-on URLs are formed in the same manner:

$GLOBALS['babAddonUrl']."main"

The $text variable is set to the text that will be displayed as menu item in the "Administration" section. But remember, Ovidentia is multilingual. Therefore our text is set with a call to the bab_translate function. This makes it possible to translate all text information for your add-on to the *user interface language*.

Your language files will be placed in the $babInstallPath/lang/addons/favorites folder. You should at least create this folder. If you don't put language files in the folder, Ovidentia will automatically create such files from the moment a user will use a new user interface language. They can then be translated later. But a better solution is of course to provide the necessary language files immediately in this folder.

The next four functions are called when Ovidentia creates or deletes a user or a group. When Ovidentia adds a new user to its database, it calls the favorites_onUserCreate($id) function with user id as an argument. This $id variable is the id used by Ovidentia to reference the user in its database. This id can be used by your add-on to store specific information by user in database tables.

When administrators delete users from the Ovidentia database, the favorites_onUserDelete($id) function is called with the id of the user to remove. This gives an opportunity to your add-on to clean data owned by this user ( in database tables in example ).

### 4.4.2 - Warning

Those functions must not output anything (such as a form that allows user data entry) and they must return as soon as possible to the caller.

For our example, we need to delete the user's data (her entries in the favorites_list database table) when this user is deleted from the Ovidentia database. To do this, change the favorites_onUserDelete() function as follows:

```
function favorites_onUserDelete( $id )

{

 $db = $GLOBALS['babDB'];

 $db->db_query("delete from favorites_list where id_owner='".$id."'");

}
```

First we declare a variable to represent our database connection and initialize it with the $GLOBALS['babDB'] variable witch let us make SQL requests to the Ovidentia database ( See the chapter on "Database Handling" further on in this guide on how to use this variable ), then make a request to delete all rows owned by the user with id_owner equal to $id.

There is nothing to do in our example when a new user is created, so we don't change the favorites_onUserCreate() function. The same is true for group creation or deletion, so we don't touch the functions favorites_onGroupCreate and favorites_onGroupDelete.

And as our "favorites" example doesn't need a specific section on the Ovidentia page, we don't modify the favorites_onSectionCreate.

If the add-on needs a dedicated section, you create a favorites_onSectionCreate(). When Ovidentia is creating its sections, favorites_onSectionCreate( &$title, &$content) will be called in order to get the title and content of its associated section. If this function returns "true" the section is created. And if the function returns false, the section is not displayed.

The favorites_upgrade() function is used when you upgrade your add-on to a new version or when the installation. Ovidentia calls this function with two parameters: $version_base and $version_ini. The former contains the last version and the last contains the new version. With the help of this function, you can upgrade your database tables or make other changes needed by the new version. See "Chapter 15. How to manage the installation and the updates ?" for more details. This function is optional.

The favorites_onDeleteAddon() function is called before the addon is deleted. If the function returns true, the system will delete all php, html and language files for the add-on. If the function returns false, the system will not delete all files. This function is optional.

The favorites_onPackageAddon() function is called before the addon is packaged. If the function returns true,

a zip file will be provided. If the function returns false, the system will not create a zip file. This function is optional.

### *4.5 - Create a folder for the language files for the add-on*

If we need to handle multiple languages for our add-on then we must always use the bab_translate($str, $name) function for each text to be displayed. In a distribution (.zip file) of an add-on we define a folder called langfiles where we put language files to be used by our add-on. These files must have a name structured as lang-xx.xml where xx is a language code.

The xx language code is composed of the two character ISO 639 language code in lower case for the language, optionally followed by –bb where bb is the two character ISO 3166 country code in lower case.

Some valid xx language codes are:

| | |
|---|---|
| de | German |
| de-ch | German, Switzerland |
| de-de | German, Germany |
| en | English |
| en-uk | English, United Kingdom |
| en-us | English, United States of America |
| fr | French |
| fr-be | French, Belgium |
| fr-ca | French, Canada |
| fr-ch | French, Switzerland |
| fr-fr | French, France |
| nl | Dutch |
| nl-be | Dutch, Belgium |
| nl-nl | Dutch, The Netherlands |

Example: lang-en-us.xml is the language file for an English user interface in the United States of America.

When you develop an add-on you place these files in a folder $babInstallPath/lang/addons/favorites.

Remark: in an Ovidentia installation these files are put in a folder named lang while in the .zip distribution file for an add-on these files are put in a folder named langfiles.

Warning: although these files have an .xml extension, they are not valid XML files.

Each language file starts and ends with language markers in agreement with the language code for the file. A language file lang-en-us.xml will start with the marker <en-us> and end with the marker </en-us> while a generic French language file lang-fr.xml will start with the marker <fr> and end with the marker </fr>.

Let's show some examples. First example is a generic lang-fr.xml French language file:

```
<fr>
<string id="N70">soixante-dix</string>
<string id="N80">quatre-vingts</string>
<string id="N90">quatre-vingt-dix</string>
</fr>
```

The content of this language file for usage in France could be identical, but the language identifier will be different as the file will be now named lang-fr-fr.xml:

```
<fr-fr>
<string id="N70">soixante-dix</string>
<string id="N80">quatre-vingts</string>
<string id="N90">quatre-vingt-dix</string>
</fr-fr>
```

But for the French language file lang-fr-be.xml as used in Belgium we have some differences:

```
<fr-be>
<string id="N70">septante</string>
<string id="N80">octante</string>
<string id="N90">nonante</string>
</fr-be>
```

And the language file lang-fr-ch.xml will be different also:

```
<fr-ch>
<string id="N70">soixante-dix</string>
<string id="N80">huitante</string>
<string id="N90">nonante</string>
</fr-ch>
```

The Ovidentia function bab_translate() will use the markers (N70, N80 and N90 and our examples) and the name of our add-on to find the text to be displayed. This function uses the language of the user, the marker text and the name of your add-on to retrieve the text to be displayed. Using our examples above, bab_translate("N80", "favorites") will return “quatre-vingts" if called when the user interface language is fr, octante" when the user interface language is fr-be and “huitante" if the user interface language is fr-ch.

If there is no translation available in the add-on for the user’s language then Ovidentia will look in the language files of the Ovidentia core application.

## *4.6 - Create folders for your add-on template files, style files and images*

Ovidentia uses skins to give different looks to an Ovidentia Web Site. Skins are differentiated by their names. Each skin is a group of template files, ".css" style files and images. Ovidentia is delivered with a skin named *ovidentia* and a style sheet *ovidentia.css*.

Skin is located in the $babInstallPath/skins/ovidentia folder.

In the same manner as for PHP files, we must store template files in the

`$babInstallPath/skins/ovidentia/templates/addons/add-on-name` folder.

In our case this folder is the

`ovidentia/skins/ovidentia/templates/addons/favorites folder.`

We call this folder *add-on template folder* and will add files to this folder in a later section.

In the same *add-on template folder* we will place our optional .css style sheet file for our add-on.

Additional images for our add-on can be placed in

`$babInstallPath/skins/ovidentia/images/addons/add-on-name folder.`

### *4.7 - Add the code needed by your add-on*

In your *add-on program folder*, create the main.php file and copy and paste this code into it:

```
<?php
/* main */
if( !isset($idx ))
$idx = "list";
     switch($idx)
     {
          case "list":
               default:
                    $babBody->title = bab_translate("Your favorites", "favorites");
                    $babBody->addItemMenu("list", bab_translate("List", "favorites"),
                         $GLOBALS['babAddonUrl']."main&idx=list");
                    $babBody->addItemMenu("new", bab_translate("Add", "favorites"),
                         $GLOBALS['babAddonUrl']."main&idx=new");
                    break;
     }


$babBody->setCurrentItemMenu($idx);
?>
```

$babBody is an Ovidentia global variable. This variable is a reference to an object that handles most of the output for the Ovidentia parts. In this object you can find some useful variables and functions:

Variables:

$babBody->title: The title text displayed on the top of the middle part of the Ovidentia web page (the content part).

$babBody->msgerror: If this variable is not empty, this text replaces the title text with a warning color.

Functions:

- $babBody->addItemMenu($idx, $text, $url )

Adds a menu item with:

- $idx: a keyword used as an index to identify the menu item. In the preceding program listing we added two menu items with "list" and "new" as keyword values for $idx.
- $text: the text that will be displayed as label for the menu item.
- $url: the url to be requested when this menu item is clicked.

- $babBody->setCurrentItemMenu($idx)

This function sets the menu item identified by the keyword value of $idx as active. This menu item will have no link and can be displayed in a different way than the other menu items.

- $babBody->babecho($text)

Prints $text in the Ovidentia output. Always use this function to output something to an Ovidentia page.

As you can see in the previous code example, we simply add two menu items: "List" and "Add" and as title, we choose “Your favorites". And we call the bab_translate function so that everything is prepared for translation to another *user interface language* if needed. The requested url and query string for these menu items are formed as shown in the example through the concatenation of the add-on url, the name of the program to be called and a query string specific to each menu item.

```
$GLOBALS['babAddonUrl']."main&idx=list"

$GLOBALS['babAddonUrl']."main&idx=new"
```

where

$GLOBALS['babAddonUrl'] is a global variable containing as value the path to your add-on program folder.

idx identifies the menu item that is selected

At this stage, you can already test your add-on. Log in as a user with administrator rights. In the "Administration" section click the "Add-ons" menu item. In the "Add-ons list" you will now see our "favorites" add-on. Click the "view" link for our add-on "favorites" to go to the page titled "Access to Add-on: favorites". In the drop down box select "Registered users" and click the "Modify" button.

**Figure 1. Add-on menu item in Administration section**


You must see that a menu entry favorites has been added to the "User's section" and if you click this menu item, you will see a title "Your favorites" and two menu items: List and Add. Further, the menu item List is active.

And if this is not the case, then you can start your first debugging session. Verify your config.php, add-on name and URLs.

As you can see the list is empty. But we want to display the user's favorites when a user clicks on the Favorites menu item in the User's section menu.

To achieve this we will start building the HTML code. In your *add-on template folder* you create a main.html file and copy and paste what follows:

```
<!--#begin bmlist -->

<table border = "0" width = "95%" cellpadding = "1" cellspacing = "0" align = "center">

<tr>
```

```
<td class = "BabLoginCadreBackground">
 <table class = "BabMailBackground" border = "0" width = "100%"
  cellpadding = "2" cellspacing = "0">
 <tr>
 <td class = "BabTitleBg" colspan = "2">
 <b>{ favorites }</b>
 </td>
 </tr>
 <!--#in getnext -->
 <tr>
 <td class = "BabTitleBg">
 <a class = "BabContentLinkColor" href = "{ bmurl }"
  target = "_blank">{ bmtext }</a>
 </td>
 <td class = "BabTitleBg" align = "right">
 <a class = "BabContentLinkColor" href = "{ bmdelurl }">{ bmdeltext }</a>
 </td>
 </tr>
 <!--#endin getnext -->
 </table>
</td>
</tr>
</table>
<!--#end bmlist -->
```

Here `<!--#begin bmlist -->` and `<!--#end bmlist -->` are our *template delimiter tags*.

And

```
{ favorites }, { bmurl }, { bmtext }, { bmdelurl }, { bmdeltext }
```

are variables.

`<!--#in getnext -->` and `<!--#endin getnext -->` delimit a block that will be repeated as many times as there are favorites.

To output this HTML code, we must write a PHP function favorites_list() that will initialize variables for the Ovidentia parser. Modify main.php as follows:

```
<?php
function favorites_list()
```

```
{
global $babBody;

class temp
 {
 var $db;
 var $res;
 var $count;
 var $favorites;
 var $bmurl;
 var $bmtext;
 var $bmdelurl;
 var $bmdeltext;

 function temp()
  {
  $this->favorites = bab_translate("Favorites", "favorites");
  $this->bmdeltext = bab_translate("Delete", "favorites");
  $this->db = $GLOBALS['babDB'];
  $this->res = $this->db->db_query("select * from favorites_list
   where id_owner='".$GLOBALS['BAB_SESS_USERID']."'");
  $this->count = $this->db->db_num_rows($this->res);
  }

 function getnext()
  {
  static $i = 0;
  if( $i < $this->count)
   {
   $arr = $this->db->db_fetch_array($this->res);
   $this->bmtext = $arr['description'];
   $this->bmurl = $arr['url'];
   $this->bmdelurl =
    $GLOBALS['babAddonUrl']
    ."main&idx=del&id=".$arr['id'];
```

```
   $i++;
   return true;
   }
  else
   return false;
  }
 }

$temp = new temp();
$babBody->babecho(bab_printTemplate($temp, $GLOBALS['babAddonHtmlPath']
 ."main.html", "bmlist"));
}

function favorites_add()
{
global $babBody;
class temp
 {
 var $url;
 var $description;
 var $add;

 function temp()
  {

  $this->url = bab_translate("Url", "favorites");
  $this->description = bab_translate("Description", "favorites");
  $this->add = bab_translate("Add", "favorites");
  }
 }

$temp = new temp();
$babBody->babecho(bab_printTemplate($temp, $GLOBALS['babAddonHtmlPath']
 ."main.html", "bmadd"));
}

/* main */
if( !isset($idx ))
```

```
 $idx = "list";
if( isset($add) && $add == "bm")
{
$db = $GLOBALS['babDB'];
if ("http://" != substr($url, 0, 7))
 {
  $url = "http://".$url;
 }
$res = $db->db_query("insert into favorites_list (url, description, id_owner)
 values ('".$url."', '".$description."', '".$GLOBALS['BAB_SESS_USERID']."')");
}

if( $idx == "del")
 {
 $db = $GLOBALS['babDB'];
 $res = $db->db_query("delete from favorites_list where id='".$id."'");
 $idx = "list";
 }
switch($idx)
{
case "new":
 $babBody->title = bab_translate("Add favorite", "favorites");
 favorites_add();
 $babBody->addItemMenu("list", bab_translate("List", "favorites"),
  $GLOBALS['babAddonUrl']."main&idx=list");
 $babBody->addItemMenu("new", bab_translate("Add", "favorites"),
  $GLOBALS['babAddonUrl']."main&idx=new");
 break;

case "list":
default:
 $babBody->title = bab_translate("Your favorites", "favorites");
 favorites_list();
 $babBody->addItemMenu("list", bab_translate("List", "favorites"),
  $GLOBALS['babAddonUrl']."main&idx=list");
 $babBody->addItemMenu("new", bab_translate("Add", "favorites"),
```

```
  $GLOBALS['babAddonUrl']."main&idx=new");
 break;
}
$babBody->setCurrentItemMenu($idx);
?>
```

We add a call to the favorites_list() and favorites_add() functions in the main part of our script. The favorites_list() function defines and instantiates a temp class that contains variables and a function getnext() used in the template file to loop through a list of values to be displayed.

We initialize the { favorites }and { bmdeltext } variables:

```
$this->favorites = bab_translate("Favorites", "favorites");
$this->bmdeltext = bab_translate("Delete", "favorites");
```

and make an SQL request to get rows from the database table favorites_list:

```
$this->res =
 $this->db->db_query("select * from favorites_list where id_owner='"
  .$GLOBALS['BAB_SESS_USERID']."'");
$this->count = $this->db->db_num_rows($this->res);
```

We remark here that we use the $GLOBALS['BAB_SESS_USERID'] global variable witch contains the id of the current logged user.

And

```
$this->count
```

now contains the number of rows.

The function getnext() is used to loop over rows. The parser engine calls this function when it reaches <!--#in getnext --> in main.html. For each row, the function initializes { bmtext }, { bmurl } and { bmdelurl } variables and returns true to tell the parser engine that there are still rows to handle. When there is no row to handle any more, the function returns false.

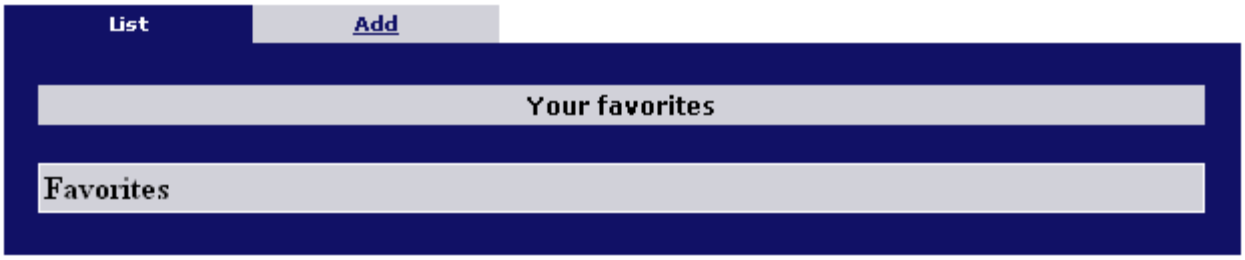
To use this class with templates, we call the

```
bab_printTemplate($temp, $GLOBALS['babAddonHtmlPath']."main.html", "bmlist")
```

function with an instance of class temp as first argument, a template file name as second argument and template name as last argument. This function makes a call to the parser engine and returns a result ( a string of HTML code ). To output the result, use the $babBody->babecho() function. This function inserts the add-on HTML in the rest of the Ovidentia page HTML.

Now, you can test the first part of your add-on.

**Figure 2. Favorites add-on empty list**




As you can see the favorites list is still empty. This is because we didn't add any favorites to our database table yet. This is what we will add now.

Change main.php as follows to handle requests for adding new favorites to the list ($idx = "new"):

Implement the favorites_add() function to output a form that lets the user enter the url and description of a new favorite:

```
function favorites_add()
 {
 global $babBody;
 class temp
  {
  var $url;
  var $description;
  var $add;

  function temp()
   {
   $this->url = bab_translate("Url", "favorites");
   $this->description = bab_translate("Description", "favorites");
   $this->add = bab_translate("Add", "favorites");
   }
  }

 $temp = new temp();
 $babBody->babecho(bab_printTemplate($temp,"addons/favorites/main.html",
 "bmadd"));
 }
```

and add a new template bmadd in our template file main.html:

```
<!--#begin bmadd -->
<form name="bmcreate" method="post" action="{ babPhpSelf }">
<input type="hidden" name="tg" value="{ babAddonTarget }/main">
<input type="hidden" name="idx" value="list">
```

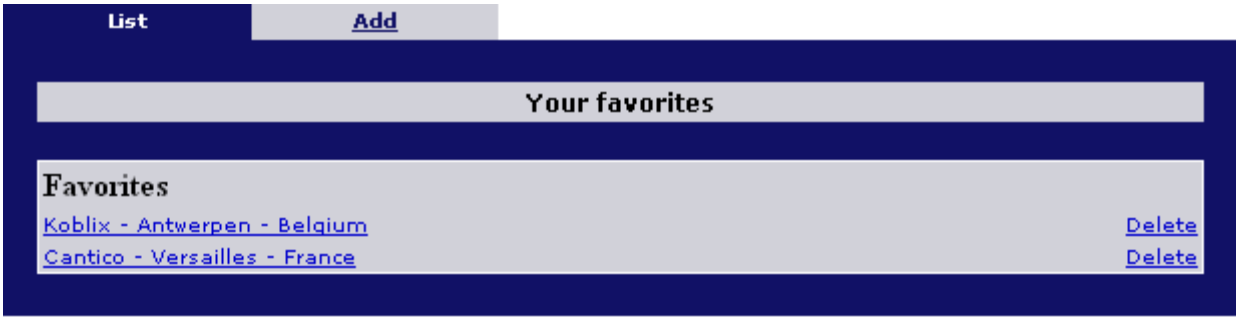
```
<input type="hidden" name="add" value="bm">
<table class="BabLoginCadreBackground" width="80%" border="0" cellspacing="0"
 cellpadding="2" align="center">
<tr>
<td>
<table class="BabLoginMenuBackground" width="100%" border="0" cellspacing="0"
 cellpadding="5" align="center">
<tr>
<td> </td>
</tr>
<tr align="center" valign="middle">
<td align="left" valign="middle"> { url }:</td>
</tr>
<tr>
<td align="center" valign="middle">
<input type="text" name="url" size="80">
</td>
</tr>
<tr>
<td align="left" valign="middle">{ description }:</td>
</tr>
<tr>
<td width="91%" align="center" valign="middle">
<input type="text" name="description" size="80">
</td>
</tr>
<tr>
<td align="center" valign="middle">
<input type="submit" name="submit" value="{ add }">
<br>
</td>
</tr>
</table>
</td>
</tr>
</table>
```

```
</form>

<!--#end bmadd -->
```

Remark how we use hidden fields to store the $tg, $idx variable values to store the request parameters. Also pay attention on how to handle "add" when the user submits the form and how to delete favorites when the user clicks on the "Delete" link.

**Figure 3. Favorites add-on list page**



# 5 - Ovidentia Global variables

| | |
|---|---|
| $BAB_SESS_LOGGED | if set and true, the current user is a registered user and is logged in. Otherwise she is an anonymous visitor. |
| $BAB_SESS_USER | contains the full name of the current user. If empty, the user is an anonymous visitor. |
| $BAB_SESS_NICKNAME | contains the nickname of the current user. If empty, the user is an anonymous visitor. |
| $BAB_SESS_USERID | contains the id of the current user. If empty, the user is an anonymous visitor. |
| $BAB_SESS_EMAIL | contains the email address of the current user. If empty, the user is an anonymous visitor. |
| $babAddonUrl | gives the full URL, including the end slash, to your *add-on program folder*. |
| $babAddonPhpPath | contains the path to your *add-on program folder*. |
| $babAddonTarget | contains "addon/x" where x is the id of your add-on in the Ovidentia database. |
| $babAddonHtmlPath | contains the path to your *add-on template folder*. |
| $babAddonFolder | this variable contains the folder name of your add-on. This is also your add-on name. |
| $babAddonUpload | this is the folder name where your add-on can store documents or files. This folder is a sub-directory of $babUploadPath folder. |
| $babCurrentDate | today date in timestamp format. |
| $babInstallPath | contains the path where the Ovidentia PHP files are installed. This path is relative to the *site document root* folder that contains the config.php script file.The path is terminated by a slash. |
| $babSkin | contains the name of the currently used skin. |
| $babSkinPath | $babInstallPath."skins/".$babSkin."/" |
| $babCssPath | “skins/".$babSkin."/styles/name of file choosen by user.css". |
| $babScriptPath | $babInstallPath."scripts/" |
| $babLanguage | holds a string initialized to the current language for the user interface ( en, fr, nl, nl-be, ... ). You can use this variable to translate text making the language compatible with the rest of the user interface. Ovidentia type language files lang-en.xml and others must be completed for your add-on. |
| $babStyle | current CSS file used ( with .css extension ). |
| $babSiteName | contains the site name. |
| $babUrl | URL to access the current site (http://www.mydomain.com/ [ http://www.mydomain.com/] ). This is your *site document root* where config.php and index.php reside. |

# 6 - The $babAddon variables

The different $babAddon variables are here to help us to write an add-on. Let's illustrate their content with an example. Imagine an add-on with the name "my-add-on" that is identified in the Ovidentia database as the add-on with id = 2. And that our site has a $babUrl = "http://www.ovidentia.org/". Then we will have the following values for the $babAddon variables:

- `$babAddonUrl = "http://www.ovidentia.org/index.php?tg=addon/2/"`
- `$babAddonPhpPath = "ovidentia/addons/my-add-on/"`
- `$babAddonTarget = "addon/2"`

By using this variable you don't have to worry about add-on's id numbers.

- `$babAddonHtmlPath = "addons/my-add-on/"`
- `$babAddonFolder = "my-add-on"`
- `$babAddonUpload = $babUploadPath. "/addons/my-add-on/"`

# 7 - Ovidentia functions

In this chapter you can find some useful functions that you can use in your add-on. You don't need to include any script file, this has already been done for you. Parameters between [ ] are optional.

| | |
|---|---|
| bab_mktime( $time ) | Returns the UNIX timestamp for MySQL dates. Uses PHP mktime() function. The date/time information returned by the database access functions is in the format yyyy-mm-dd hh:mm:ss . Given date/time information stored in this format as a parameter, this function returns a long integer Unix timestamp. Daylight savings time is not taken into account. |
| bab_ strftime($time, [false]) | Returns a date for a timestamp given in $time. If the optional second parameter is set to true the return value includes time ( hours and minutes ). This function uses the current language to format the date. If the second parameter is set to false only "day of the week", "day of the month", "month" and "year" are returned. |
| bab_ longDate($time, [true]) | Returns a long format of date for a timestamp given in $time. If the optional second parameter is set to true the return value includes time ( hours and minutes ). This function uses user's preferences. |
| bab_ shortDate($time, [true]) | Returns a short format of date for a timestamp given in $time. If the optional second parameter is set to true the return value includes time ( hours and minutes ). This function uses user's preferences. |
| bab_ time($time) | Returns a format of time for a timestamp given in $time. This function uses user's preferences. |
| bab_browserOS() | Returns the Operating System of the client browser. Possible returned values are: windows, linux, macos, or an empty string. |
| bab_browserAgent() | Returns the current client browser. Possible returned values are: konqueror, opera, msie ( Microsoft IE ), nn4 ( Netscape 4 ), nn6 ( Netscape 6) or an empty string. |
| bab_browserVersion() | Returns the current client browser version. |
| bab_translate( $str, $name ) | Returns the translated string of $str using the current user interface language. If $str is not found, $str is returned unmodified. $name must contain the name of you add-on. |
| bab_isUserAdministrator() | Returns true if the current user is an Ovidentia administrator, otherwise returns false. |
| bab_isUserGroupManager( [$grpid] ) | Without parameter this function returns true if the current user is a group manager, otherwise this function returns false. If $grpid is given, the function returns true if the current user is a group manager for the group identified by $grpid, otherwise false is returned. |
| bab_getUserName( $id ) | Returns the name of the user who has $id as identifier in the database. An empty string is returned if the user can't be found. |
| bab_getUserEmail( $id ) | Returns the email address of the user who has $id as identifier in the database. An empty string is returned if the user can't be found. |
| bab_getUserNickname( $id ) | Returns the nickname of the user who has $id as identifier in the database. An empty string is returned if the user can't be found. |

bab_getUserSetting( $id, $what ) This function returns information on the preference settings for the user identified by $id. An empty string is returned if the user can't be found. The variable $what can have following values:

- "lang": the returned value is the user's preferred language
- "skin": the returned value is user's preferred skin
- "style": the returned value is user's preferred style

bab_getGroupName( $id ) Returns the name of the group that has $id as identifier in the database. An empty string is returned if the group can't be found.

bab_getPrimaryGroupId( $id ) Returns the name of the primary group for the user who has $id as identifier in the database. An empty string is returned if the user or the group can't be found or if the user doesn't have an assigned primary group. This function is deprecated.

# 8 - Database handling

You can create your own database or use the Ovidentia database. You must to prefix your tables with your add-on name to avoid name clashes. See 'Appendix A. Reserved prefixes used to avoid name clashes' for more details.

To create your own database, use the bab_database class. You can also extend this class to provide more functionality or override some functions.

To use the Ovidentia database, use the $GLOBALS["babDB"] global variable. This variable holds a bab_database extended class which frees you from calling db_connect() and removes the link identifier from all calls.

For example, use this variable as follows:

```
$db = $GLOBALS['babDB'];

$res = $db->query("select * from mytable");

while( $row = $db->db_fetch_row($res))

 {

 ...

 }
```

You don’t need to close the connection to the database server.

## *8.1 - The bab_database class*

bab_database($die = false, $dbtype = "MySQL") Constructor. If $die = true and there is an error, the script will die. The variable $dbtype is the type of database. In the current Ovidentia version, only “MySQL" is allowed.

db_connect($host, $login, $password, $dbname) Connects to the database server using $host, $login, $password, $dbname. Return false if error, or a link identifier if success.

db_query($id, $query) Executes an SQL query where $id is the link identifier.

db_num_rows($result) Returns the number of rows in a result set.

db_fetch_array($result) Returns an array that corresponds to the fetched row, or false if there are no more rows. You can then address a column's value by using the column's name as a key index for the array. Such as:

$my_result = $my_array['column_a'];

db_fetch_row($result) Returns an array that corresponds to the fetched row, or false if there are no more rows. In the resulting array the database columns corresponds to numeric indices.

db_result($result, $row, $field) Returns the contents of one cell from a result set.

db_affected_rows($id) Gets the number of affected rows in a previous SQL operation.

| | |
|---|---|
| db_insert_id($id) | Returns the ID generated for an AUTO_INCREMENT column by the previous INSERT query using the given link identifier $id. |
| db_data_seek($res, $row) | Moves the internal row pointer of the SQL result associated with the specified result identifier $res to point to the specified row number. |

If you have a need for some other database functions, you can always extend this class and implement them in your extended class

# 9 - Group access rights to Ovidentia objects

Sometimes we will want to restrict access to some user groups for certain functions or objects in Ovidentia.

The following method to handle group access rights is used in Ovidentia to give groups access to topics, forums, sections …

In order to use the Ovidentia method to grant group access, do the following:

- Create a database table to store the group access rights for your object.
- Create the code needed for the Administration section to create a link to the access control function.
- Insert a test to check if the current user has access to a given function or object.

## *9.1 - Database table for group access rights*

Ovidentia uses a database table to store the access rights for groups to functions or objects.

Use the following SQL request to create this table:

```
CREATE TABLE xxx_groups (
   id int(11) unsigned NOT NULL auto_increment,
   id_object int(11) unsigned DEFAULT '0' NOT NULL,
   id_group int(11) unsigned DEFAULT '0' NOT NULL,
   PRIMARY KEY (id)
);
```

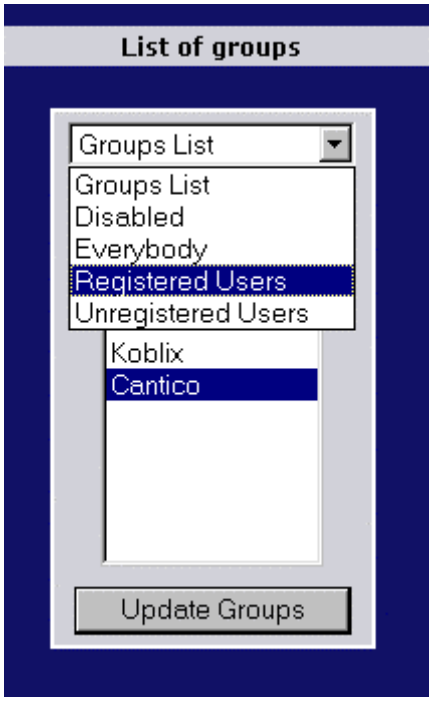
In this SQL statement we have:

| | |
|---|---|
| xxx_groups | The table name for your group access rights. You can give any name to your table, but the field names must be as indicated. Be aware of possible name clashes. It is recommended to register a prefix for your needs and use this prefix in a consistent fashion. For add-on modules you can use your add-on name as a prefix. |
| id_object | Identification of the function or object for which you want to grant access to certain groups. |
| id_group | The group id of the group for which you grant access to the function or object identified by the id_object field. |

### 9.1.1 - Code for the Administration section

The administrator can grant access rights to groups using the following selection list box:

**Figure 4. Set Group Access**




To display this selection list box for the Administrator you must call:

```
aclGroups("myscriptpath/myscript", "myidx", "xxx_groups", $id, "acl");
```

A call to the function aclGroups must be preceded by the inclusion of the acl.php file:

```
include ( $babInstallPath."admin/acl.php"  );
```

where

| | |
|---|---|
| "myscriptpath/myscript" | "myscript" is the name of your program script file that handles the browser request when an Administrator submits the selection list box form. And "myscriptpath" is the path from $babInstallPath to your program script. This will be expanded as: $babInstallPath/myscriptpath/myscript.php |
| "myidx" | is the value of the $idx variable after submission. |
| "xxx_groups" | is the name of your database table. |
| $id | is the object id for which you want to grant the access rights. |
| "acl" | is the name of the variable that contains the return value after submission. This value is always "update". |

When an Administrator submits the form, your script myscript.php receives 6 variables:

| | |
|---|---|
| $idx | contains the value passed as second argument in the preceding call. In this case $idx contains "myidx". |
| $acl | this is the variable that has the same name as the last argument in the function aclGroups() call. |
| $table | contains the name of your table. In this example this is "xxx_groups". |

| | |
|---|---|
| $item | contains the same value as $id in the function call. |
| $groups | an array with group id numbers selected by the user. |
| $what | additional internal information. |

You must use those variables as follows:

```
if( isset($acl))
 {
 aclUpdate($table, $item, $groups, $what);
 }
```

This lets Ovidentia update your table with the wanted access rights to your function or object for groups.

### *9.2 - Test current user's access rights*

To see if the current user has access to the object with a given ID, use the following function call:

```
bab_isAccessValid("xxx_groups", ID);
```

If the return value is true, the current user has access. Otherwise this user is not authorized to use the object that is identified by ID.

# 10 - How to use workflow approbation schemas

Your add-on can use approbation schemas to let users approve content.

To use this Ovidentia built-in function, you must include the following file:

```
include_once $babInstallPath."utilit/wfincl.php";
```

To let an Administrator choose which approbation schema he will use, you must give him the list of available approbation schemas. For this use the function:

```
bab_WFGetApprobationsList()
```

This function returns a multi-dimensional array:

```
return_value[0]["name"] = first schema name

return_value[0]["id"] = first schema id

return_value[n]["name"] = nth schema name

return_value[n]["id"] = nth schema id
```

To use an approbation schema, you must create an instance of the schema. For this use the function:

```
bab_WFMakeInstance ($idsch, $extra)
```

This function creates an instance of an approbation schema with id $idsch and return the id of the instance.

$extra is additional information and is not used by Ovidentia. You can use this field for debug purpose for example.

All next calls uses the instance Id returned by this function.

To delete an instance, use the function:

```
bab_WFDeleteInstance ($idschi)
```

where $idschi is the Id of the instance returned by the function bab_WFMakeInstance().

The function

```
bab_WFGetWaitingApproversInstance($idschi, $notify)
```

returns an array with user ids who are waiting for the approbation schema instance with id $idschi.

$notify is a boolean. If it's value is true, the function marks waiting users as to be notified.

To update an instance with user's response use the function:

```
bab_WFUpdateInstance($idschi, $iduser, $bool)
```

where $idshi is the id of the instance, $iduser is the id of the user and $bool is a boolean with true if the user accepts the approbation and false if the user declines the approbation.

This function returns the new result of the schema approbation:

0 if the approbation is declined and then the subject of the approbation must be revoked.

1 if the approbation is accepted and hence the subject of the approbation must be accepted.

Otherwise the approbation can't be evaluated at this moment and you must call bab_WFGetWaitingApproversInstance()to see which users needs to approve the instance.

You can also use the following functions:

```
bab_WFCheckInstance($idsa, $iduser)
```

This function returns true if there is a waiting instance approbation for the user with id $iduser . $idsa is the id of the approbation schema.

```
bab_WFCheckInstance($idsa, $iduser)
```

This function returns true if there is a waiting approbation instance for the user with id $iduser . $idsa is the id of the approbation schema.

```
bab_WFGetWaitingInstances($iduser)
```

This function returns an array of all waiting instances.

# 11 - How to add a search function to your add-on

If you wish to let users perform search operations on data maintained by your add-on, Ovidentia helps you with two functions:

```
function favorites_searchinfos()
```

This function is called by Ovidentia to get the name to insert in the Search listbox and that will be used for search operations within your add-on. This listbox contains all sub-categories for which the user can perform search operations.

```
function favorites_searchresults($str1, $str2, $option, $pos, $nb_result)
```

This function performs a search based on information in $str1, $str2 and $option and returns one result.

Positions are numbered starting from 0.

| | |
|---|---|
| $str1 | the string to search |
| $str2 | optional. Used with $option parameter. |
| $option | OR, AND or NOT. This tell your add-on to search for “$str1 $option $str2". For example “foo or bar" |
| $pos | Position in the list result |
| $nb_result | Max number of entries needed by Ovidentia |

For example, if the search results gives 25 results and if Ovidentia displays results by pages with 5 rows per page, calls are made as follow:

5 calls with $pos =0 and nb_results=5

5 calls with $pos = 5 and nb_result = 5

and so on …

This function is called each time Ovidentia need to add one item for the search result.

It must return an array structured as follows:

| | |
|---|---|
| `Return_array[0]` | Description of the item |
| `Return_array[1]` | Total number of the search result |
| `Return_array[2]` | An array |
| `Return_array[2][0]` | Url to access the result entry |
| `Return_array[2][1]` | Description of the entry |
| `Return_array[3]` | An array |

| | |
|---|---|
| `Return_array[3][0]` | Url to access the result entry using a popup window |
| `Return_array[3][1]` | Description of the entry |

# 12 - How to add mail functionality to your add-on

You can use the Ovidentia built-in mail functionality by including

```
include_once $babInstallPath."utilit/mailincl.php";
```

and using the object returned by the constructor bab_mail():

```
$mail = bab_mail();
```

Different methods allow you to compose and send e-mails:

`$mail->mailFrom($email, $name)`

Adds a “From" address

`$mail->mailTo($email, $name)`

Adds a “To" address. Can be called many times

`$mail->mailCc($email, $name)`

Adds a “CC" address. Can be called many times

`$mail->mailBcc($email, $name)`

Adds a “Bcc" address. Can be called many times

`$mail->ClearTo()`

Clears all recipients assigned in the “To" array

`$mail->ClearCc()`

Clears all recipients assigned in the “Cc" array

`$mail->ClearBcc()`

Clears all recipients assigned in the “Bcc" array

`$mail->mailReplyTo($email, $name)`

Adds a “Reply-To" address

`$mail->clearReplyTo()`

Clears a “Reply-To" address

`$mail->mailSubject ($subject)`

Sets the Subject of the message

`$mail->setPriority($priority)`

Set the priority of the message (1 = High, 3 = Normal, 5 = Low).

`$mail->mailBody($babBody, $format)`

Sets the “Body" of the message. This can be either an HTML ( $format ="html") or text body ($format ="plain").

`$mail->mailAltBody($babAltBody)`

Sets the text-only body of the message. This automatically sets the email to multipart/alternative. This body can be read by mail clients that do not have HTML e-mail capability. Clients that can read HTML will view the normal Body.

`$mail->mailTemplate ($message)`

Use Ovidentia template mail to embed your message.

`$mail->send()`

Sends the message. Returns true on success or false otherwise

# 13 - Calendar API

Add-ons can interact with calendars by using the Calendar API.

To use this API, add-ons must include the calapi.php script file:

```
include_once $babInstallPath."utilit/calapi.php";
```

**bab_calGetCategories():**

Get all categories.

Returns complete results on calendar category information in a multi-dimensional array:

| | |
|---|---|
| `return_value[i]` | refers to the details of the ith entry. |
| `return_value[i]['id']` | id of the category. |
| `return_value[i]['name']` | name of the category. |
| `return_value[i]['description']` | description of the category. |
| `return_value[i]['color']` | background color of the category. |

**bab_getPersonalCalendar():**

Get the id of the personal calendar on success and 0 on error.

**bab_getResourceCalendars():**

Get all resources calendars.

Returns complete results on resource calendar information in a multi-dimensional array:

| | |
|---|---|
| `return_value[i]` | refers to the details of the ith entry. |
| `return_value[i]['id']` | id of the calendar. |
| `return_value[i]['name']` | name of the calendar. |
| `return_value[i]['description']` | description of the calendar. |
| `return_value[i]['rights']['view']` | true if current user has view rights for this calendar |
| `return_value[i]['rights']['add']` | true if current user has rights to add events to this calendar |

**bab_getPublicCalendars():**

Get all public calendars.

Returns complete results on public calendar information in a multi-dimensional array:

| | |
|---|---|
| `return_value[i]` | refers to the details of the ith entry. |
| `return_value[i]['id']` | id of the calendar. |

| | |
|---|---|
| `return_value[i]['name']` | name of the calendar. |
| `return_value[i]['description']` | description of the calendar. |
| `return_value[i]['rights']['view']` | true if current user has view rights for this calendar |
| `return_value[i]['rights']['add']` | true if current user has rights to add events to this calendar |

**`bab_calGetEvents($params):`**

Get all events for a specific calendar and specified dates.

Returns detailed results about calendar events in a multi-dimensional array:

Parameter $params is an array:

| | |
|---|---|
| `params['id_cal']` | id of the calendar for which you want to get events. Can be an array of ids. |
| `params['begindate']` | start date ( sql DATETIME ) |
| `params['enddate']` | end date ( sql DATETIME ) |
| `params['id_category']` | id of the category if you want to fetch only events that have this category |
| `params['order']` | “asc" for ascendant and “desc" for descendant. |

Return value:

| | |
|---|---|
| `return_value[i]` | refers to the details of the ith entry. |
| `return_value[i]['id_calendar']` | id of the calendar. |
| `return_value[i]['id_event']` | id of the event. |
| `return_value[i]['title']` | title of the event. |
| `return_value[i]['description']` | description of the event. |
| `return_value[i]['location']` | location of the event. |
| `return_value[i]['begindate']` | start date of the event. ( sql DATETIME ) |
| `return_value[i]['enddate']` | end date of the event. ( sql DATETIME ) |
| `return_value[i]['id_category']` | id of the category or 0 |
| `return_value[i]['name_category']` | name of the category |
| `return_value[i]['dsecription category']` | description of the category |
| `return_value[i]['id_creator']` | id of the author of this event |
| `return_value[i]['backgroundcolor']` | color used if any |
| `return_value[i]['private']` | true if the event is a private event |
| `return_value[i]['lock']` | true if the event is locked |
| `return_value[i]['free']` | true if the event is free |
| `return_value[i]['status']` | 1 if this event is waiting for approbation. Otherwise zero. |
| `return_value[i]['quantity']` | if the event is related to a vacation, this field contains the |

number of days

**bab_calGetFreeEvents($idcals, $sdate, $edate, $gap, $bopt):**

Get all free events for a specific calendar on specific dates.

Returns complete result information on free events in a multi-dimensional array:

Parameters:

| | |
|---|---|
| $id_cals | an array of calendar ids. |
| $sdate | start date ( sql DATE format ). |
| $edate | end date ( sql DATE format ). |
| $gap | in seconds. Minimum duration of event |
| $bopt | “Y" if the system must use calendar options of the user or “N". |

Return value:

| | |
|---|---|
| return_value[i] | refers to the detail of the $i^{th}$ entry. |
| return_value[i][0] | start time ( timestamp ). |
| return_value[i][1] | end time ( timestamp ).. |

**bab_newEvent ($idcals, $args, &$msgerror):**

Create a new event in the calendars with ids in $idcals ( array) and based on information in $args.

Returns True on success or False on error ( if error $msgerror contains additional information ):

Parameter $args is an array:

| | |
|---|---|
| args['startdate'] | array : start date of the event. |
| args['startdate']['day'] | day of the start date. |
| args['startdate']['month'] | month of the start date. |
| args['startdate']['year'] | year of the start date. |
| args['startdate']['hours'] | hours of the start date. |
| args['startdate']['minutes'] | minutes of the start date. |
| args['enddate'] | array : end date of the event. |
| args['startdate']['day'] | day of the end date. |
| args['startdate']['month'] | month of the end date. |
| args['startdate']['year'] | year of the end date. |
| args['startdate']['hours'] | hours of the end date. |
| args['startdate']['minutes'] | minutes of the end date. |

| | |
|---|---|
| args['owner'] | id of the author. |
| args['category'] | id of the category |
| args['private'] | true if the event is a private event. Or false |
| args['lock'] | true if the event must be locked. Or false |
| args['free'] | true if the event has to be marked free. Or false |
| args['rrule'] | this property defines a repeating pattern for recurring events.<br>0 no recurrence<br>BAB_CAL_RECUR_DAILY daily recurrence<br>BAB_CAL_RECUR_WEEKLY weekly recurrence<br>BAB_CAL_RECUR_MONTHLY monthly recurrence<br>BAB_CAL_RECUR_YEARLY yearly recurrence |
| args['ndays'] | if BAB_CAL_RECUR_DAILY is set the repetition occurs every args ['ndays']days. |
| args['nweeks'] | if BAB_CAL_RECUR_WEEKLY is set the repetition occurs every args['nweeks']weeks. |
| args['nmonths'] | if BAB_CAL_RECUR_MONTHLY is set the repetition occurs every args['nmonths']months. |
| args['nyears'] | if BAB_CAL_RECUR_YEARLY is set the repetition occurs every args ['nyears']years. |
| args['rdays'] | if BAB_CAL_RECUR_WEEKLY is set the repetition occurs for days listed in this array *( 0* (for Sunday) through *6* (for Saturday) ) |
| args['until'] | this property defines the end date of the repetition. |
| args['until']['day'] | day of the date. |
| args['until']['month'] | month of the date. |
| args['until']['year'] | year of the date. |

# 14 - How to add ovml functions to your add-on

In order to understand what follows in this chapter you need a working knowledge of OvML, the **Ov**identia **M**arkup **L**anguage. See the OvML documentation for more details.

The OvML language has been incorporated in Ovidentia in order to allow users to retrieve information that will be displayed as part of HTML documents. This information is often stored in the database. In order to retrieve information specific to an add-on it is sometimes necessary to create new OvML functions. In example, using a file manager, OvML functions will allow you to build yourself the HTML interface to display the information managed by your file manager.

## *14.1 - OCAddon Container*

You will find in the OvML documentation the general syntax to be used with add-ons:

```
<OCAddon name="" param1="" param1="" ...>
        <OVvar1>
        <OVvar2>
</OCAddon>
```

name : name of the add-on where the function can be found (prefix of the add-on)

param1 : first parameter to send to the add-on, generally the function name to be used

param2 : second parameter to be send to the add-on, generally this is the first parameter for the OvML function of the add-on

…

When this syntax is used the Ovidentia OvML parser will send some specific data to the add-on, as well as the parameters of the <OCAddon> container. The add-on wil then return the values corresponding with the variables (<OVvar1> and <OVvar2>) used by the function.

### 14.1.1 - Add the code needed by your add-on

In order to handle new OvML functions in our add-on we need to add an ovml.php file in the add-on program folder $babInstallPath/addons/add-on/add-on-name where add-on-name is the name of our add-on.

This file must have the following content:

```
function add-on-name_ovml($args) {}
```

where add-on-name is the name of our add-on (its prefix) and $args is an associative array with the names and values of the parameters needed for the function. The internal working of this function are fully controlled

by the programmer.

Example of parameter restitution:

```
if (isset($args['param1'])) {
	switch($args['param1']){
		case '1': ...
		case '2': ...
	}
}
```

It is important that you always return an indexed array with in it associative arrays that contain the variable names (without the OV prefix) and their values. The indexed array will multiple entries (meaning multiple associative arrays) when more than one result is returned. The OvML container <OCAddon> will loop as many times over the array as there are indexes in it.

When there is no result to be returned you must return an empty array.

Example of a returned result:

```
$tab = array();
$feed['var1'] = "value var1";
$tab[0] = $feed;
return $tab;
```

### *14.2 - OFAddon Function*

You will find in the OvML documentation the general syntax to be used with add-ons:

```
<OFAddon name="" function="" param1="" ...>
```

name: name of the add-on

function: name of the function

param1: first parameter to send to the function

...

#### 14.2.1 - Add the code needed by your add-on

In order to handle new OvML functions in our add-on we need to add an ovml.php file in the add-on program folder $babInstallPath/addons/add-on/add-on-name where add-on-name is the name of our add-on.

This file must have the following content:

```
function add-on-name_function-name($arg1,$arg2 ...) {}
```

where add-on-name is the name of our add-on (its prefix) and $arg1, $arg2 are the first and the second argument.

The internal working of this function are fully controlled by the programmer. Contrary to the container OCAddon, the use of OFAddon does not send variables (<OV...>). OFAddon send a string which is going to replace the function <OFAddon> in the ovml code.

Remark :

With OCAddon container (function add-on-name_ovml($args)), $args is an associative array with the names and values of the parameters needed for the function.

With OFAddon, there is no array. Every function's parameter ($arg1, $arg2) is associated to the parameter.

The order of writing of the parameters is very important.

Ex:

```
function favorites_translate($lang,$var) {}
```

The syntax of use will be:

<OFAddon name="favorites" function="translate" lang="fr" var="hello">

Inside the function favorites_translate, $lang is going to receive the string "fr" et $var is going to receive "hello".

This syntax is false:

<OFAddon name="favorites" function="translate" var="hello" lang="fr">

car $lang is going to receive "hello" et $var is going to receive "fr".

### *14.3 - Use ovml files appropriate for the add-on*

The url of access to a ovml file present in current skin is the following one:

index.php?tg=oml&file=name of ovml file

It is possible to add ovml files appropriate for a add-on. Path in the file zip of the add-on:

skins\ovidentia\ovml\

After the installation of the add-on, the ovml files present in the file zip will exist in this path:

core of Ovientia\skins\ovidentia\addons\name of add-on\

Syntax to launch an ovml file appropriate for a add-on:

index.php?tg=oml&file=addons\prefix of add-on\name of ovml file

# 15 - How to manage the installation and the updates?

The complete management of add-ons is done through the Ovidentia user interface, Administration section, link Modules. Here the administrator will install add-ons, update them, download them in .zip format and deletes them.

When you upload a .zip file containing an add-on you will see an update icon associated with it. This makes it possible to execute other operations that are needed, such as the creation of new tables in the database. When you upload an add-on all files are created and already existing files are overwritten. Before you click the Update icon the add-on application is suspended and unavailable for users because the database has not yet been installed or updated.

In order to create tables or to perform other operations you ned to create a function such as

```
function add-on-name_upgrade($version_base, $version_ini) {}
```

This function is optional and if present the function must be found in the init.php of the add-on. The function will be called automatically at installation or update time.

## *15.1 - Database specifics*

In the function add-on-name_upgrade() of the init.php function you must handle all update tasks, taking into account not to create already existing tables. You can do whatever is needed, including moving content or renaming tables.

Remark: An automatic update will replace all existing files with those found in the .zip file. In all updates published by Cantico updates are not complements to the old version but they are complete replacements. So you can install an add-on for the first time or update an existing add-on with the same .zip file.

## *15.2 - Handling the deletion of an add-on*

See "Implement some initialization functions needed by Ovidentia in the init.php script file" and the description of the function add-onPrefix_onDeleteAddon().

### *15.3 - Handling download requests for the add-on*

See "Implement some initialization functions needed by Ovidentia in the init.php script file" and the description of function add-onPrefix_onPackageAddon().

# 16 - How to document your add-on?

In order to correctly document your add-on you should respect the following guidelines.

The distribution package that you will prepare in the chapter "How to package your add-on for distribution?" has two files named description.html and history.txt. In the following paragraphs we will describe what should be in this description.html document and history.txt.

Remark:

By using the automatic method of creation of a zip file, the description.html file is automatically informed with the information of the addonini.php file.

description.html :

- Add-on name. This is the name of the add-on folder in your distribution package that contains all files need to properly install your add-on. See the chapter "How to package your add-on for distribution?" for more information about the content of this package.
- Add-on version.
- Add-on prefix. This is the prefix used by your add-on to avoid name clashes.
- Author information. Credits to authors, author company information, a link to a web site where more information is available, an E-mail address for support, ...
- Functionality of the add-on. Give a short description of the functions of your add-on. What does it do? Who should use it? How does it work?
- Prerequisites. Describe whatever is prerequisite for your add-on to function correctly. Does your add-on need a specific Ovidentia version? Are other add-ons required? Are there specific PHP modules needed? Or any other server-side software? Client side?
- Administrator section entries created by the add-on.
- User's section entries created by the add-on.
- Database modifications. Describe any tables you add and their usage.

history.txt :

When you upload an Ovidentia add-on then you will discover some information concerning the application: version, history, delete option …

These icons are only added when files addonini.php and history.txt are present in the programs folder of the .zip file or in the folder $babInstallPath/addons/favorites.

Simply putting a history.txt text file in the appropriate folder will make the history icon available. Although the history.txt file is optional it is an excellent way to illustrate the product evolutions without having to consult the description.html file.

# 17 - How to package your add-on for distribution?

In order to make it easy to distribute your add-on you should follow the packaging guidelines below.

1. Create a folder with the name of your add-on.
2. Put your description file description.html in this folder. See the "How to document your add-on ?" chapter for more information about the contents of this document.
3. Create a folder programs in your add-on folder.
4. Put your PHP script files in this programs folder.
5. Create a folder templates in your add-on folder and a folder styles inside.
6. Put your HTML template files in this templates folder and your styles files in this styles folder.
7. Create a folder langfiles in your add-on folder.
8. Put your language files in this langfiles folder.
9. Create a folder ovml in your add-on folder.
10. Put your ovml files in this ovml folder.
11. Create a zip file (not compressed) that contains all files and directories This .zip file must not contain the folder with the name of your add-on.

Remark: check that the folders programs, skins and langfiles are effectively at the root of the .zip file. This is important for the automatic installation facility that will copy the files directly is the corresponding Ovidentia folders. A filename should be structured as add-onName-version. No spaces are allowed in the filename.

Example: favorites-1-1.zip

Loading a file named favorites-1-1.zip (or with any other version suffix such as favorites-2-34.zip) will result in the creation of a folder named favorites in the Ovidentia addons folder. Each function that needs to be prefixed must have that same name favorites as its prefix, such as favorites_onPackageAddon().

Warning: the version as indicated in the filename is not used. The version that will be used to identify your add-on must be given in the addonini.php file. But you can use version suffixes in our .zip files in order to clearly identify each version. But then you have to be careful that each version file starts with the same prefix (the add-on name) to avoid the creation of more than one add-on!

So if we look back to our "favorites" add-on earlier in this guide, we should have a zip file as follows:

```
ZIP FILE
|
|-langfiles
| |-lang-de.xml
| |-lang-en.xml
| |-lang-fr.xml
| |-lang-nl.xml
| |-lang-nl-be.xml
|
|-programs
| |-addonini.php
| |-init.php
| |-main.php
| |-ovml.php
| |-xxx.php
|
|-skins
| |-ovidentia
| | |-templates
| | | |-main.html
| | | |-xxx.html
| | | |-styles
| | | | |-xxx.css
| | |-images
| | | |-xxx.jpeg
| | | |-xxx.gif
| | |-ovml
| | | |-xxx.ovml
|
|-description.html
|
```

# Appendix A. Reserved prefixes used to avoid name clashes

In order to avoid name clashes prefixes are used for add-ons and customer specific modifications. These prefixes are used with table names, function and class names, script variables. The following prefixes have been reserved:

- bab_ is the prefix used by the Ovidentia developers for the core product.
- bro_ is a customer specific prefix.
- ko_ is the prefix used by Koblix for its add-ons and some customer specific modifications.

To request the reservation of a prefix please send an e-mail to support@koblix.com where we are maintaining the Ovidentia documentation.