Xaraya
Request for Comments: 0042
Category: Best Current Practice

M.R. van der Boom
Xaraya Development Group
Februari 2003

Module hooks system

Status of this Memo

This document specifies a Xaraya Best Current Practices for the Xaraya Community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

Copyright Notice

Abstract

This RFC documents the hooks system for Xaraya. We have a hooks system in place, but undocumented. This RFC should resolve that by at least documenting what we have now.

Table of Contents

1. Introduction

The hook system allows site administrators to extend the functionality of modules - with common features like comments, hitcounts, ratings, support for BBCode or Wiki etc.

Those features are provided by so-called hook modules : utility modules that offer a number of hook functions, that will be called automatically when the corresponding hooks are enabled.

The main difference between hook calls and regular API calls is that hook calls to a particular hook function (e.g. to display comments) are not hard-coded in the module itself, but you can turn them on and off again via the administration.

This gives you a much greater flexibility, both as a site administrator and as a module developer, in terms of the optional features that you want to provide in different parts of your site or in your module.

2. Types of hooks

There are 5 main types of hooks that are currently used to extend module functionality, plus some special-purpose hooks. Some hooks will provide part of the graphical user interface (GUI) to the user, while others will only operate on the API level.

2.1 Item Transform Hooks (API)

This type of hook allows you to transform some pieces of text of your module items (e.g. the summary and body of articles, or the description of a project, ...) in different ways, depending on the hook modules that are enabled for it. Hook modules that provide this type of hooks are for instance bbcode, autolinks, wiki, ...

2.2 Item Display Hooks (GUI)

This type of hook shows some additional information that relates to a particular module item when that item is being displayed. Typical examples include comments, hitcount, ratings, ...

2.3 Item Create/Update/Delete Hooks (API)

Some hook modules follow the whole life cycle of module items, from the moment the item is created, through any updates it may have, and up to the point where the item is deleted again.

In some cases, like the hitcount or ratings module, those hooks are only used to keep track of some internal value for each module item. In other cases, like for categories and dynamicdata, the hook module will also provide a graphical interface - see next type of hook.

2.4 Item New/Modify Hooks (GUI)

This type of hook extends the previous one by showing some additional input fields in forms where module items are added or modified. This allows you to e.g. assign a module item to a particular category, or to fill in some dynamic properties for that item, etc.

The corresponding values will then be saved or updated when the item is created/updated.

2.5 Module Configuration Hooks (GUI/API)

Some hook modules, like categories and dynamicdata, also offer specific configuration options for each module that is hooked to them. This type of hook will automatically insert those options within the rest of the module configuration screen, and update them when the module configuration itself is being updated.

2.6 Special-Purpose Hooks : Search, UserMenu, WaitingContent, ...

All the hook types described above provide some common add-on features that can be used to extend different modules. So a utility module like autolinks, that provides an Item Transform hook, can be used by any module that calls the standard Item Transform hooks, e.g. in articles, xproject, xarbb, ...

There is another type of hooks that works the other way around : those hooks are only called in one particular place, by one particular module, and other modules that provide this hook function will only extend this particular functionality.

One example of this is the Item Search hook, which is used by the search module to call any hook function (from any module) that provides a search interface. So the articles module may provide a search hook function, and the xarbb module may provide one, and the roles module as well, and it's up to the site administrator to decide which search functions he wants to allow by enabling the corresponding hooks.

Another example is the UserMenu hook, which is used by the roles module to call any hook function that provides a user menu, for integration in "Your Account". So the articles module may provide a user menu function, and the xarbb module may provide one, and the roles module as well, and it's up to the site administrator to decide which user menu functions he wants to allow by enabling the corresponding hooks.

The WaitingContent hook is another example, which is used by the adminpanels module to call any hook function that provides a list of waiting content, for use in the waitingcontent block. So the articles module may provide a list of waiting content, and the xarbb module may provide one, and the polls module as well, and it's up to the site administrator to decide which waiting content he wants to see by enabling the corresponding hooks.

The Module Remove hook functions are called by Xaraya when a particular module is being removed, so that hook modules can perform some clean-up of their own table(s) if necessary.

3. Enabling or disabling hooks

Site administrators can enable hooks via the Modules administration :

- if you want to enable some hooks for a particular module (e.g. articles), you can do that via Modules -> View All -> extend that particular module
- if on the other hand, you want to enable/disable a particular hook (e.g. comments) for several modules at once (e.g. articles, example, polls, ...), you should do that via Modules -> Configure hooks -> choose that particular hook module

If you want to enable or disable hooks via code, see also below.

Note : some modules handle more than one type of items (e.g. articles has different publication types), and you may want to enable hooks for one particular item type in that module. You can do this in the Modules administration as well.

4. Designing your module to make use of hooks

Insert xarModCallHooks() in all the right places - cfr. example module. // TODO: improve the parameters

The syntax of the xarModCallHooks() function is as follows :

```
$outputarray = xarModCallHooks('<hookobject>', '<hookaction>', $objectid,
$extrainfo);
```

In general, API hooks expect an associative array containing the current item information as $extrainfo, and return that array with extended or transformed information. GUI hooks expect the same item information as $extrainfo, and return an associative array of hookmodule => hookoutput pairs

By default, hooks are always called for the current "main module". If you need to call the hooks for another module, e.g. in a block or API function, you can specify the caller module in $extrainfo['module']. If your module deals with different item types, you should pass on the currently relevant item type via $extrainfo['itemtype'].

4.1 Calling Item Transform Hooks (API)

```
Module code in xaruser.php :

function <module>_user_display($args)
{
    ...
    // get the item information
    $item = xarModAPIFunc('<module>', 'user', 'get',
                          array('someid' => $itemid, ...));
    ...
    // add some arguments for hooks
    $item['module'] = '<module>';
    $item['itemtype'] = $itemtype;
    ...
/* old-style transform hooks
    list($item['some'],
         $item['pieces'],
         $item['of'],
         $item['text']) = xarModCallHooks('item', 'transform', $itemid,
                                          // the pieces of text you want to transform
here
                                          array($item['some'],
                                                $item['pieces'],
                                                $item['of'],
                                                $item['text']));
*/
    // the fields (array keys) corresponding to the pieces of text you want to
transform here
    $item['transform'] = array('some', 'pieces', 'of', 'text');

    // transform item
    $item = xarModCallHooks('item', 'transform', $itemid, $item);
    ...
}
```

4.2 Calling Item Display Hooks (GUI)

```
Module code in xaruser.php :

function <module>_user_display($args)
{
    ...
    // get the item information
    $item = xarModAPIFunc('<module>', 'user', 'get',
                          array('someid' => $itemid, ...));
    ...
    // add some arguments for hooks
```

```
    $item['module'] = '<module>';
    $item['itemtype'] = $itemtype;
    ...
/* old-style display hooks
    $hookoutput = xarModCallHooks('item', 'display', $itemid,
                                  // the URL to return to e.g. after rating the item,
or commenting on it, or ...
                                  xarModURL('<module>', 'user', 'display',
                                            array('someid' => $itemid, ...))
                                 );
*/
    // the URL to return to e.g. after rating the item, or commenting on it, or ...
    $item['returnurl'] = xarModURL('<module>', 'user', 'display',
                                   array('someid' => $itemid, ...));
    // get display hook output
    $hookoutput = xarModCallHooks('item', 'display', $itemid, $item);

    if (empty($hookoutput)) {
       $data['hooks'] = '';
    } elseif (is_array($hookoutput)) {
        // $hookoutput['comments'] contains the hook output for comments,
        // $hookoutput['hitcount'] contains the hook output for the hitcount,
        // $hookoutput['ratings'] contains the hook output for ratings, and so on...

        // you can use them individually in the template, or join everything together
        $data['hooks'] = join('',$hookoutput);
    } else {
        // this will no longer be relevant
        $data['hooks'] = $hookoutput;
    }
    ...
}
```

4.3 Calling Item Create/Update/Delete Hooks (API)

```
Module code in xaradminapi.php :

function <module>_adminapi_create($args)
{
    ...
    // create the item first
    ...
    // then add some arguments for hooks, including the newly created itemid
    $args['module'] = '<module>';
    // adapt if you're dealing with several types of items in your module
    $args['itemtype'] = 0;
    $args['itemid'] = $itemid;
    // then call the create hooks
    $result = xarModCallHooks('item', 'create', $itemid, $args);
    ...
}

function <module>_adminapi_update($args)
{
    ...
    // update the item first
    ...
    // then add some arguments for hooks
    $args['module'] = '<module>';
    // adapt if you're dealing with several types of items in your module
    $args['itemtype'] = 0;
    $args['itemid'] = $itemid;
    // then call the update hooks
    $result = xarModCallHooks('item', 'update', $itemid, $args);
    ...
}

function <module>_adminapi_delete($args)
{
    ...
    // add some arguments for hooks
```

```
    $args['module'] = '<module>';
    // adapt if you're dealing with several types of items in your module
    $args['itemtype'] = 0;
    $args['itemid'] = $itemid;
    // then call the delete hooks
    $result = xarModCallHooks('item', 'delete', $itemid, $args);
    ...
    // then delete the item itself at last
    ...
}
```

Note: in some cases, you may want to call update hooks even when you're only updating part of a module item (e.g. the status of several articles, or the password of a user). In that case, you should signal the hook modules that they should not update any hooked values because they won't be available :

```
xarVarSetCached('Hooks.all','noupdate',1);
```

4.4 Calling Item New/Modify Hooks (GUI)

```
Module code in xaradmin.php :

function <module>_admin_new($args)
{
    ...
    // we have no item information here yet
    $item = array();
    // we don't have an item id yet either
    $itemid = '';
    ...
    $item['module'] = '<module>';
    // adapt if you're dealing with several types of items in your module
    $item['itemtype'] = 0;
    $item['itemid'] = $itemid;

    $hookoutput = xarModCallHooks('item', 'new', $itemid, $item);
    if (empty($hookoutput)) {
        $data['hooks'] = '';
    } elseif (is_array($hookoutput)) {
        // $hookoutput['categories'] contains the hook output for categories,
        // $hookoutput['dynamicdata'] contains the hook output for dynamicdata, and so
on...

        // you can use them individually in the template, or join everything together
        $data['hooks'] = join('',$hookoutput);
    } else {
        // this will no longer be relevant
        $data['hooks'] = $hookoutput;
    }
    ...
}

function <module>_admin_modify($args)
{
    ...
    // get the item information
    $item = xarModAPIFunc('<module>', 'user', 'get',
                          array('someid' => $itemid, ...));
    ...
    $item['module'] = '<module>';
    // adapt if you're dealing with several types of items in your module
    $item['itemtype'] = 0;
    $item['itemid'] = $itemid;

    $hookoutput = xarModCallHooks('item', 'modify', $itemid, $item);
    if (empty($hookoutput)) {
        $data['hooks'] = '';
    } elseif (is_array($hookoutput)) {
        // $hookoutput['categories'] contains the hook output for categories,
        // $hookoutput['dynamicdata'] contains the hook output for dynamicdata, and so
on...
```

```
        // you can use them individually in the template, or join everything together
        $data['hooks'] = join('',$hookoutput);
    } else {
        // this will no longer be relevant
        $data['hooks'] = $hookoutput;
    }
    ...
}
```

4.5 Calling Module Configuration Hooks (GUI/API)

```
Module code in xaradmin.php :

function <module>_admin_modifyconfig($args)
{
    ...
    $extrainfo = array();
    $extrainfo['module'] = '<module>';
    // adapt if you're dealing with several types of items in your module
    $extrainfo['itemtype'] = 0;

    $hookoutput = xarModCallHooks('module', 'modifyconfig', '<module>', $extrainfo);
    if (empty($hookoutput)) {
        $data['hooks'] = '';
    } elseif (is_array($hookoutput)) {
        // $hookoutput['categories'] contains the hook output for categories,
        // $hookoutput['dynamicdata'] contains the hook output for dynamicdata, and so
on...

        // you can use them individually in the template, or join everything together
        $data['hooks'] = join('',$hookoutput);
    } else {
        // this will no longer be relevant
        $data['hooks'] = $hookoutput;
    }
    ...
}

function <module>_admin_updateconfig($args)
{
    ...
    $extrainfo = array();
    $extrainfo['module'] = '<module>';
    // adapt if you're dealing with several types of items in your module
    $extrainfo['itemtype'] = 0;

    $result = xarModCallHooks('module', 'updateconfig', '<module>', $extrainfo);
    ...
}
```

4.6 Providing Special-Purpose Hooks : Search, UserMenu, WaitingContent, ...

In this case, you'll need to create a hook function and register it at initialisation - see next section

4.7 Other hook-related functions

You can check if a particular hook module is enabled for your module by using the xarModIsHooked() function :

```
   ...
   if (xarModIsHooked('<hookmodule>')) {
   }
   ...
```

If you want to use this in places where your module may not be the current "main module", e.g. in blocks or API functions, you should specify the caller module as an extra parameter :

```
   ...
   if (xarModIsHooked('<hookmodule>', '<yourmodule>')) {
   }
   ...
```

And if you want to check if a hook is enabled for a particular item type in your module, you should specify the item type as the third argument :

```
   ...
   if (xarModIsHooked('<hookmodule>', '<yourmodule>', $thistype)) {
   }
   ...
```

You can enable hooks to your module via code, e.g. at initialisation, by using one of the API functions provided by the 'modules' module :

```
   ...
    xarModAPIFunc('modules', 'admin', 'enablehooks',
                  array('callerModName' => '<yourmodule>', 'hookModName' =>
'<hookmodule>'));
   ...
```

To disable hooks via code, you can use the corresponding function as well :

```
   ...
    xarModAPIFunc('modules', 'admin', 'disablehooks',
                  array('callerModName' => '<yourmodule>', 'hookModName' =>
'<hookmodule>'));
   ...
```

To specify a particular item type in one of those two functions, you can use the extra parameter 'callerItemType' here.

5. Providing hook functions in your module

Decide what functionality you want to offer first, then see what type(s) of hooks you need to provide for that...

Please note that your hook function may be called by modules other than the current "main module", so you should check for any optional $extrainfo['module'] parameter if your hook module stores information regarding particular modules or module items. Some modules calling your hook functions might also be handling several item types, which they will pass as an optional $extrainfo['itemtype'] parameter. So your module should take into account that item type as well, if relevant.

5.1 Writing an API hook function

```
Module code in xaruserapi.php or xaradminapi.php :

/**
 * The hook function you want to provide, e.g. for 'item' 'transform' 'API'
 *
 * @param $args['objectid'] ID of the object
 * @param $args['extrainfo'] extra information
 * @returns array
 * @return extrainfo the updated extrainfo array
 */
function <module>_<type>api_<hookfunc>($args)
{
    ...
}
```

Note: in some cases, people may want to call update hooks even when they're only updating part of a module item (e.g. the status of several articles, or the password of a user). In that case, your hook module should not update any hooked values because they won't be available :

```
    if (xarVarIsCached('Hooks.all','noupdate')) {
        // return without updating any values
        return $args['extrainfo'];
    }
```

5.2 Writing a GUI hook function

```
Module code in xaruser.php or xaradmin.php :

/**
 * The hook function you want to provide, e.g. for 'item' 'display' 'GUI'
 *
 * @param $args['objectid'] ID of the object
 * @param $args['extrainfo'] extra information
 * @returns string
 * @return the HTML output for this hook function
 */
function <module>_<type>_<hookfunc>($args)
{
    ...
}
```

5.3 Registering hook functions

```
Module code in xarinit.php :

function <module>_init()
{
    ...
    xarModRegisterHook(// the type of hook you want to register
                       'item', 'display', 'GUI',
                       // the hook function to be called
                       '<module>', '<type>', '<hookfunc>');
    ...
```

```
}

function <module>_delete()
{
    ...
    xarModUnregisterHook(// the type of hook you want to unregister
                         'item', 'display', 'GUI',
                         // the hook function to be called
                         '<module>', '<type>', '<hookfunc>');
    ...
}
```

5.4 Creating a Search hook function (GUI)

A search hook function is only going to be invoked by the search module. Next to the standard $args['objectid'] and $args['extrainfo'] arguments, which are not going to contain anything interesting here, your search hook function will be able to retrieve the current search query via the input variable 'q'.

```
Module code in xaruser.php :

/**
 * The hook function you want to provide for 'item' 'search' 'GUI'
 *
 * @param $args['objectid'] ID of the object
 * @param $args['extrainfo'] extra information
 * @returns string
 * @return the HTML output for this hook function
 */
function <module>_user_search($args)
{
    list($q,
         $author,
         $...) = xarVarCleanFromInput('q',
                                      'author',
                                      '...');
    if (empty($q)) {
        // return part of a search form, with input fields relevant to your module
        ...
    } else {
        // return the results for your module
        ...
    }
}
```

You can add other input fields of your own in the GUI output, but make sure they don't create a conflict with input fields from other modules. Note that the same function will be invoked by the search module to show (part of) the search form, but also to show (part of) the result page.

5.5 Creating a User Menu hook function (GUI)

A user menu hook function is only going to be invoked by the roles module, to generate some user menu in the "Your Account" page. You can use this e.g. to allow users to change their preferences for your module from within the "Your Account" page.

```
Module code in xaruser.php :

/**
 * The hook function you want to provide for 'item' 'usermenu' 'GUI'
 *
 * @param $args['objectid'] ID of the object
 * @param $args['extrainfo'] extra information
 * @returns string
 * @return the HTML output for this hook function
 */
function <module>_user_usermenu($args)
{
    list($update,
```

```
         $bold,
         $itemsperpage,
         $...) = xarVarCleanFromInput('update',
                                      'bold',
                                      'itemsperpage',
                                      '...');
    if (!empty($update)) {
        // update the user preferences
        if (isset($bold)) {
            xarModSetUserVar('<module>', 'bold', 1);
        } else {
            xarModSetUserVar('<module>', 'bold', 0);
        }
        ...
    } else {
        // get the current user preferences
        $bold = xarModGetUserVar('<module>', 'bold');
        ...
    }
    // show an input form where the user can change his preferences for your module
    // Tip : include a hidden field called 'update' in the form, so that you know
    //       when to update the user preferences, and when to simply show them
    ...
}
```

Note that the same function will be invoked by the roles module to show the current preferences of the user, and also to update them.

5.6 Creating a Module Remove hook function (API)

If your hook module keeps track of some information for module items of different modules, you may want to do some clean-up in your own table(s) when some hooked module gets deleted by the site administrator. In this case, you can define a 'module' 'remove' 'API' function that will be automatically invoked by Xaraya.

```
Module code in xaradminapi.php :

/**
 * The hook function you want to provide for 'module' 'remove' 'API'
 *
 * @param $args['objectid'] ID of the object
 * @param $args['extrainfo'] extra information
 * @returns array
 * @return extrainfo the updated extrainfo array
 */
function <module>_adminapi_removehook($args)
{
    extract($args);
    ...
    // in this case, $objectid will contain the name of the module that is being
removed
    $modid = xarModGetIDFromName($objectid);
    // do some clean-up in your table(s) for that module
    ...
}
```

6. Hints on using different item types in your modules

Many of the more complex modules deal with different types of items. For instance : articles, calendar, cast, contact, events, issueareapub, nascar, shopping, todolist, workflow, xarbb or xproject.

Unfortunately, they often don't make use of the notion of item types, which means that they (and your future module users) can't really benefit from Xaraya's hook system. Wouldn't it be nice if you could enable categories just for one item type, or extend your item types in distinct ways via dynamicdata, or enable transformations for some item types but not for others, etc. ? :-)

Here's a little overview on how to adapt your modules to use item types, so that you can make full use of the Xaraya hook system.

6.1 Identify your item types

This is generally pretty easy - just see how many different module tables that you have in your module. Each of those is probably a good candidate for becoming a separate item type.

Or if you have lots of similar functions like create_this, create_that, etc., that's another indicator for spotting item types.

6.2 Make them known to Xaraya

All you need for that is to provide a user API function getitemtypes(), in modules/<mymodule>/xaruserapi/getitemtypes.php or in your global modules/<mymodule>/xaruserapi.php file.

This way, the hook system will know your module handles different item types, and it will present them as separate checkboxes to the site admin in the hooks configuration screens.

Your getitemtypes() function should return an array of

```
$itemtypes[$itemtype] = array('label' => 'some label to display',
                              'title' => 'some link title to use',
                              'url' => <url to view them (if relevant)>);
```

In some cases, this array will be static (e.g. one per module table), or in others this may be built up dynamically - like for articles pubtypes or dynamicdata objects for instance.

6.3 Call the right hooks with the right arguments

Many modules still don't specify the calling module or item type in their hook calls, or don't even call hooks at all. This makes it impossible for Xaraya to figure out which hooks it should actually call, so you'll want to go through your code and adapt the xarModCallHooks() calls if necessary.

This RFC explains the different hook calls and what parameters they expect. In most cases, adaptation is pretty straight-forward : just create an array with the item information, the module and the itemtype, and pass that as $extrainfo argument instead of (whatever).

6.4 Sit back and relax :-)

========== extra credit ==========

There are a few other things you can do to make life easier for your module users later on :

6.5 Provide links to your items in top hits, top ratings, categories etc.

For this, you'll need to have a user API function getitemlinks(), that accepts the itemtype and an array of itemids as input, and returns an array of

```
$itemlinks[$itemid] =  array('label' => 'some text/title to display',
                             'title' => 'some link title to use',
                             'url' => <url to display it (if relevant)>);
```

6.6 Working with categories

Now that the categories module supports different item types per module, you can re-use it more easily to let your users navigate through your items by category.

The easiest approach would be to define a dynamic object for each of your item types, and to let DD handle the selection for you. For that, all you need to do is fetch the 'catid' variable from the input, and pass it along to your DD function call or BL tag, together with the item type that you're dealing with. You'll find some different ways to do that in the dyn_example module.

Slightly more difficult is when you're doing your own database queries etc. In that case, you can retrieve the relevant pieces of SQL from the categories module, and include them in your queries. For instance :

```
if (xarModIsHooked('categories','mymodule',$thistype)) {
    // Get the LEFT JOIN ... ON ...  and WHERE parts from categories
    $categoriesdef = xarModAPIFunc('categories','user','leftjoin',
                                   array('modid' =>
                                         xarModGetIDFromName('mymodule'),
                                         'itemtype' => $type,
                                         'catid' => $catid));
    $query = "SELECT xar_this, xar_that
              FROM ( $mytable
              LEFT JOIN $categoriesdef[table]
              ON $categoriesdef[field] = xar_myitemid )
              $categoriesdef[more]
              WHERE xar_whatever = 'something'
              AND $categoriesdef[where]";
} else {
    $query = "SELECT xar_this, xar_that
              FROM $mytable
              WHERE xar_whatever = 'something'";
}
```

And retrieving the categories for some items can be done with :

```
$cids = xarModAPIFunc('categories','user','getlinks',
                      array('iids' => array($item1, $item2, ...),
                            'itemtype' => $mytype,
                            'modid' => xarModGetIDFromName('mymodule'),
                            'reverse' => 1));
```

Now all you need to do is add some categories navigation tag in your templates, and people will be able to browse through your items too :

```
<xar:if condition="xarModIsHooked('categories','mymodule',$thistype)">
  <xar:categories-navigation layout="trails" showchildren="1"
  module="mymodule" itemtype="$thistype" catid="$catid" />
</xar:if>
```

Note: all of that is done automagically by the dynamicdata module if you let it handle the selection and display :-)

That's about all I can think of regarding item types, and how you can benefit from Xaraya's hook system by using them...

7. Notes on the roles module

The roles module uses hooks among other things in the delete, purge and recall API calls.

Delete: A deleted role is still present but not visible, and can be recalled to become visible again. The roles module ignores deleted roles in all of its operations.

Recall: A recalled role is one that was previously deleted (and therefore ignored), but is put back into circulation again..

Purge: A purged role is for all practical purposes completely gone from Xaraya. References to it are maintained for data integrity purposes, but it cannot be accessed or recalled.

The API calls that perform delete, recall and purge in the roles module each have a hook call that lets other modules know that the action has taken place. Since other modules or external applications may or may not have equivalent concepts, the hook calls are variants of conventional "delete" and "create" hook calls. Each call has an additional bit of information that specifies it. For example the hook call in the recall API function is:

```
// Let any hooks know that we have recalled this user.
$item['module'] = 'roles';
$item['itemid'] = $item['uid'];
$item['method'] = 'recall';
xarModCallHooks('item', 'create', $uid, $item);
```

It is up to modules to analyze these calls and take appropriate action based on their own state.

8. When (not) to use hooks

Comparison with API calls and the event system...

9. Implementation Details

Current stuff in the core and modules module...

10. Future Evolution

One major drawback of the current hook types is that most of the hooked functionality can only be used when you're dealing with individual module items, not when you're dealing with several items at once. So you can't easily select module items based on some hooked information, or show some hooked information when viewing a list of items.

In order to provide the same kind of functionality when dealing with more than one module item, we can't simply add some GUI output in a list - we need some way to integrate the data retrieval from those "loosely coupled" hook modules with the data retrieval for the module items themselves, or work on the API and/or BL level to achieve the same thing.

A first approach to address this issue -used by the articles module at the moment (March 2003)- consisted of making the module "aware" of several commonly used hooks, and to include the data retrieval and GUI integration for those hooks in the module code. This solves the immediate problem of integrating certain hooks when dealing with lists of articles, and still allows you to enable or disable those hooks via administration, but it goes against the idea of "loose coupling" and requires extensive code in the module itself.

A more generic approach -for use by any module- is being investigated on the data access level. Whether this will be limited to some kind of "SQL hooks", or will cover the whole issue of data access to module items together with their "related" information is still being explored at the moment. Either way, some changes are to be expected in the hook calls in the future.

11. Revision history

2003-10-24: Add Hints on using different item types in your modules

2003-06-28: Add comments on how to call hooks on behalf of a particular module, and how to specify the item type (if any)

2003-03-27: Start filling in some sections

2003-02-27: Created

Author's Address

Marcel van der Boom
Xaraya Development Group
EMail: marcel@hsdev.com
URI: http://www.xaraya.com

Intellectual Property Statement

The DDF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the DDF's procedures with respect to rights in standards-track and standards-related documentation can be found in RFC-0.

The DDF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the DDF Board of Directors.

Acknowledgement

Funding for the RFC Editor function is provided by the DDF