WebAccess Plugin Hooks

The following document describes hooks placed in the webaccess as part of the Webaccess Plugin System.

Table of Contents

1 Hooks

1.1 Entry point scripts

This category describes the hooks that are placed in the entry point scripts as index.php, static.php and zarafa.php. These hooks are used to perform actions right before or right after the whole system has performed their native tasks. The static.php is used to combine and output the javascript and CSS files that are included by the webaccess main window and dialogs. The zarafa.php handles the AJAX requests done by the webaccess. The index.php handles the setup of the main window and dialogs, the download and upload attachment scripts, the outputting of the translations for the client-side and login.

server.index.load.dialog.after
This hook is triggered after all the code to load the dialog has been run. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far.

server.index.load.dialog.before
This hook is triggered before all the code to load the dialog is run. It can be used to initialize any object or load data that is needed.

server.index.load.download_attachment.after
This hook is triggered after all the code to download an attachment has been run. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far. In this case it can also be used keep track of the downloaded files. Note that you cannot do any outputting at this point when a file has been downloaded.

server.index.load.download_attachment.before
This hook is triggered before the code to download an attachment will be run. It can be used to initialize any object or load data that is needed.

server.index.load.jstranslations.after
This hook is triggered after all the code to load the translations for the client-side has been run. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far.

server.index.load.jstranslations.before
This hook is triggered before the the code to load the translations for the client-side will be run. It can be used to initialize any object or load data that is needed.

server.index.load.main.after
This hook is triggered after all the code to load the main window has been run. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far.

server.index.load.main.before
This hook is triggered before the code to load the main window is run. It can be used to initialize any object or load data that is needed.

server.index.load.upload_attachment.after
This hook is triggered when the script to upload an attachment is called through the index.php file. Normally the attachments dialog is used to upload an attachment and in that case this hook will not be triggered. This hook is not triggered when a dialog uploads attachments by refreshing the page.

This hook is triggered when the code to upload an attachment has been run. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far.

server.index.load.upload_attachment.before
This hook is triggered when the script to upload an attachment is called through the index.php file. Normally the attachments dialog is used to upload an attachment and in that case this hook will not be triggered. This hook is not triggered when a dialog uploads attachments by

refreshing the page.

This hook is triggered before the code to upload an attachment will be run. It can be used to initialize any object or load data that is needed.

server.index.login.success
This hook is triggered when the user has logged in successfully and will be redirected to the webaccess. The redirection will happen when the $_GET variable contains the key "logon".

1.2 Main window

This category describes the hooks that are triggered in the scripts that are run to build the main window.

client.core.viewcontroller.initview.addcustomview
This hook can be used to implement your own Views in the webaccess.

server.main.include.cssfiles
This hook allows a plugin to add paths to CSS files the main window should load. This is not meant for local CSS files, because those can be include by defining them in the manifest. This is for loading in external files. With this hook you can define the full URL to the file. Something that is not possible in the manifest as the path is always prepended with the path to the plugins directory.

server.main.include.jsfiles
This hook allows a plugin to add paths to Javascript files the main window should load. This is not meant for local Javascript files, because those can be include by defining them in the manifest. This is for loading in external files. With this hook you can define the full URL to the file. Something that is not possible in the manifest as the path is always prepended with the path to the plugins directory.

1.3 Dialogs

This category describes the hooks that are triggered in the dialogs.

server.dialog.gab_detail_distlist.tabs.htmloutput
This hook allows you to add extra tabs to the GAB distlist details dialog. This hook should be used in combination with the hook "server.dialog.gab_detail_distlist.tabs.setup". The setup hook registers the tabs to the TabBar and the htmloutput hook starts and ends the tab and outputs the HTML for content of the tab.

server.dialog.gab_detail_distlist.tabs.setup
This hook allows you to add extra tabs to the GAB distlist details dialog. This hook should be used in combination with the hook "server.dialog.gab_detail_distlist.tabs.htmloutput". The setup hook registers the tabs to the TabBar and the htmloutput hook starts and ends the tab and outputs the HTML for content of the tab.

server.dialog.gab_detail_mailuser.tabs.htmloutput
This hook allows you to add extra tabs to the GAB mailuser details dialog. This hook should be used in combination with the hook "server.dialog.gab_detail_mailuser.tabs.htmloutput". The setup hook registers the tabs to the TabBar and the htmloutput hook starts and ends the tab and outputs the HTML for content of the tab.

server.dialog.gab_detail_mailuser.tabs.setup
This hook allows you to add extra tabs to the GAB mailuser details dialog. This hook should be used in combination with the hook "server.dialog.gab_detail_mailuser.tabs.setup". The setup hook registers the tabs to the TabBar and the htmloutput hook starts and ends the tab and outputs the HTML for content of the tab.

client.dialog.general.javascript.onload.after
This hook allows the execution of Javascript code when the document is finished loading. The code is executed after the general dialog initialization code is run and after the onload-code specific for this type of dialog is executed. This last bit of code is added by the servers-side dialog setup function getJavascript_onload(). In case you only want to execute Javascript in one

specific type of dialog you can use the task identifier that is defined in the data object.

client.dialog.general.javascript.onload.before
This hook allows the execution of Javascript code when the document is finished loading. The code is executed just after the general dialog initialization code is run. After this hook the onload-code specific for this type of dialog is executed. This last bit of code is added by the servers-side dialog setup function getJavascript_onload(). In case you only want to execute Javascript in one specific type of dialog you can use the task identifier that is defined in the data object.

client.dialog.general.javascript.onresize.after
This hook allows the execution of Javascript code when the window is resized. The code is executed just after the onload-code specific for this type of dialog is executed. This last bit of code is added by the servers-side dialog setup function getJavascript_onresize(). In case you only want to execute Javascript in one specific type of dialog you can use the task identifier that is defined in the data object.

client.dialog.general.javascript.onresize.before
This hook allows the execution of Javascript code when the window is resized. The code is executed just before the onload-code specific for this type of dialog is executed. This last bit of code is added by the servers-side dialog setup function getJavascript_onresize(). In case you only want to execute Javascript in one specific type of dialog you can use the task identifier that is defined in the data object.

server.dialog.attachments.setup.getbody.uploadformhtml
This hook allows you to add HTML within the upload form that is used in the attachments dialog. The attachments dialog is a special dialog that submits the form on the page to upload files. When submitting the page only the content of the fields that are inside the FORM tags is sent to the server. Defining fields outside the FORM tags will result in the loss of the information when the entire page refreshes. With this hook you can add hidden fields that will be sent to the server along with the uploaded files.

server.dialog.general.include.cssfiles
This hook allows a plugin to add paths to CSS files the dialog should load. This is not meant for local CSS files, because those can be include by defining them in the manifest. This is for loading in external CSS files. With this hook you can define the full URL to the file. Something that is not possible in the manifest as the path is always prepended with the path to the plugins directory.

server.dialog.general.include.jsfiles
This hook allows a plugin to add paths to Javascript files the dialog should load. This is not meant for local Javascript files, because those can be include by defining them in the manifest. This is for loading in external Javascript files. With this hook you can define the full URL to the file. Something that is not possible in the manifest as the path is always prepended with the path to the plugins directory.

server.dialog.general.setup.getbody.before
This hook allows you to add HTML before the dialog setup function getBody() has called. This can be used to add hidden fields to the dialog. In case you only want to add HTML to one specific type of dialog you can use the task identifier that is also defined in the data object.

server.dialog.general.setup.getincludes
This hook allows you to add components to this dialog. This is primarialy used for native webaccess components. Plugin files can be added by defining them in the manifest. It is possible that you may only want to add certain plugin components for one specific type of dialog. In that case it is also possible to use this hook as the task identifier is also defined in the data object.

server.dialog.general.setup.getmenubuttons
When a dialog posts attachments to the server, it will post it to the dialog URL. The main example of this behavior is the attachments dialog that uploads the files by submitting a form that will refresh the entire dialog page. The dialog will post to the DIALOG URL so it will return to a user interface when the page is loaded.

One of the first actions that happen when the dialog URL is requested is checking if the upload *attachment script should be included. When this is the case this hook is triggered just before the include statement. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far.*

server.dialog.general.setup.getmodulename
This hook allows you to change the module that is used in any dialog. By checking the task of the dialog you can change a module for a specific dialog. The common practice is to extend the original modules and override methods to change the behavior. In this case it is important not to forget to include the superclass modules.

server.dialog.general.upload_attachment.after
When a dialog posts attachments to the server, it will post it to the dialog URL. The main example of this behavior is the attachments dialog that uploads the files by submitting a form that will refresh the entire dialog page. The dialog will post to the DIALOG URL so it will return to a user interface when the page is loaded.

One of the first actions that happen when the dialog URL is requested is checking if the upload attachment script should be included. When this is the case this hook is triggered just before the include statement. It can be used to initialize any object or load data that is needed.

1.4 Modules

This category describes the hooks that are triggered in the modules.

client.module.addressbookitemmodule.item.after
This hook can be used to handle the response from the server with an entry for the addressbook details dialog. This hook can be used to modify the UI based on this data. It will take place after the default actions have been taken.

client.module.addressbookitemmodule.item.before
This hook can be used to handle the response from the server with an entry for the addressbook details dialog. This hook can be used to modify the UI based on this data. It will take place before the default actions have been taken.

client.module.contactlistmodule.contextmenu.buildup
This hook can be used to add an item to the contextmenu that is opened when you click with the right mouse button on a contact in any contact list view.

client.module.contactitemmodule.init.parsingObject
This hook can be used to add a custom Contact Parser Object. This Contact Parser Object will overwrite the default behavior of the address parsing, telephone number parsing and the full name parsing.

client.module.contactlistmodule.topmenu.buildup
This hook can be used to add an item to the top menu that is shown when the contactlistmodule is loaded.

client.module.hierarchymodule.contextmenu.buildup
This hook can be used to add an item to the contextmenu that is opened when you click with the right mouse button on a folder in the hierarchy list.

client.module.hierarchymodule.sharedFoldersPane.buildup
This hook can be used to quickly add links to the little pane below the hierarchylist in the webaccess interface. Note that this pane is not resized when extra content is added.

client.module.itemmodule.item.before
Some of the derived itemmodule classes have their own implementation of the item method, but they also call back to the item method of their superclass. This is usually done at the end of their method. So this BEFORE hook is not at the start of item method of the derived itemmodule class.

client.module.maillistmodule.contextmenu.buildup
This hook can be used to add an item to the contextmenu that is opened when you click with the right mouse button on an email in any mail list view.

client.module.maillistmodule.execute.before

The execute method for the maillistmodule is called when it receives data sent by the server.

client.module.previewreadmailitemmodule.item.before
The item method is called when the module receives data of one item from the server. Before the data is processed this hook is called.

client.module.readmailitemmodule.item.before
The item method is called when the module receives data of one item from the server. Before the data is processed this hook is called.

client.module.readmailitemmodule.setbody.postdisplay
This hook gives you the ability to interact with the IFRAME as a DOM object. The difference with the PREDISPLAY hook is that that hook only allows you to change the string that is added to the iframe as content. With this hook you can follow up on content added to the IFRAME by searching for them in the document and add events to links for example.

client.module.readmailitemmodule.setbody.predisplay
This hook allows you to change the content of the body before it is added to the IFRAME. Since it is only a string you cannot interact with the content as you would be able to do if it was a DOM tree. To do that you need the POSTDISPLAY hook.

server.module.addressbookitemmodule.open.props
This hook allows you to add and modify the list of properties that will be send to the client.

server.module.itemmodule.execute.after
This hook allows to run code in the open method of the itemmodule after all the default actions to open a message have taken place. With this hook you can still change the data that will be returned to the client.

server.module.maillistmodule.execute.after
This hook allows to run code before the requests to the maillistmodule are handled in the execute method.

server.module.maillistmodule.execute.before
This hook allows to run code after the requests to the maillistmodule have been handled in the execute method.

1.5 Server core

server.core.properties.addressbookitem.abobject
This hook allows you to add your own properties to the list of properties that is retrieved for all the addressbook objects shown in the addressbook details dialogs. The addressbook objects are the list of members, the manager, the owner, etc.

server.core.properties.addressbookitem.distlist
This hook allows you to add your own properties to the list of properties that is retrieved for the addressbook distlist (or AB container in the case of companies) that is loaded by the addressbookitemmodule in the addressbook details dialog.

server.core.properties.addressbookitem.mailuser
This hook allows you to add your own properties to the list of properties that is retrieved for the addressbook mailuser that is loaded by the addressbookitemmodule in the addressbook details dialog.

2 Hook reference

This chapter describes in detail the hooks that are placed in the webaccess.

2.1 client.core.viewcontroller.initview.addcustomview

Name	client.core.viewcontroller.initview.addcustomview
Location	/php-webclient/client/core/viewcontroller.js
Client/Server	Client-side
Category	Main

This hook can be used to implement your own Views in the webaccess.

Data	Static/Ref	Type	Description
view	static	string	Name of the selected view that the viewcontroller is expected to instantiate.
viewcontroller	reference	object	Reference to the viewcontroller object. In order to instantiate your own view you will need to store the view object in the viewObject property of this viewcontroller object.
moduleID	static	number	ID of the module that this viewcontroller instantiated.
element	reference	DOM element	HTML element that the selected view should use to output its data to.
events	reference	object	List of events that need to be defined for the rows of a list/table view.
data	reference	object	Optional argument passed to viewcontroller.initView method. This object can hold any extra data the View needs.
uniqueid	static	string	Optional argument passed to viewcontroller.initView method. It is used by listmodules to set the column that holds an unique ID for each row and can by used to identify rows.

The following code example shows how to use this hook to implement an own View.

```
data["viewcontroller"].viewObject =
      new ExampleView(data["moduleID"], data["element"], data["events"], data["data"]);
```

2.2 client.dialog.general.onload.after

Name	client.dialog.general.general.onload.after
Location	/php-webclient/client/layout/dialogs/window.php
Client/Server	Server-side
Category	Dialog

This hook allows the execution of Javascript code when the document is finished loading. The code is executed after the general dialog initialization code is run and after the onload-code specific for this type of dialog is executed. This last bit of code is added by the servers-side dialog setup function getJavascript_onload(). In case you only want to execute Javascript in one specific type of dialog you can use the task identifier that is defined in the data object.

Data	Static/Ref	Type	Description
task	static	string	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	string	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.

The following example shows the execution of Javascript in the onload event of a plugin defined dialog.

```
Pluginexample.prototype.exampleDialogOnLoad(data) {

      // Add hidden input fields to example dialog of example plugin
      if (data["task"] == "example_dialog" && data["plugin"] == "example"){

             var input = dhtml.getElementById("inputfieldX");
             input.value = "Hello world!";

      }

}
```

2.3 client.dialog.general.onload.before

Name	client.dialog.general.general.onload.before
Location	/php-webclient/client/layout/dialogs/window.php
Client/Server	Server-side
Category	Dialog

This hook allows the execution of Javascript code when the document is finished loading. The code is executed just after the general dialog initialization code is run. After this hook the onload-code specific for this type of dialog is executed. This last bit of code is added by the servers-side dialog setup function getJavascript_onload(). In case you only want to execute Javascript in one specific type of dialog you can use the task identifier that is defined in the data object.

Data	Static/Ref	Type	Description
task	static	string	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	string	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.

The following example shows the execution of Javascript in the onload event of a plugin defined dialog.

```
Pluginexample.prototype.exampleDialogOnLoad(data) {

      // Add hidden input fields to example dialog of example plugin
      if (data["task"] == "example_dialog" && data["plugin"] == "example"){

             var input = dhtml.getElementById("inputfieldX");
             input.value = "Hello world!";

      }

}
```

2.4 client.dialog.general.onresize.after

Name	client.dialog.general.general.onresize.after
Location	/php-webclient/client/layout/dialogs/window.php
Client/Server	Server-side
Category	Dialog

This hook allows the execution of Javascript code when the window is resized. The code is executed just after the onload-code specific for this type of dialog is executed. This last bit of code is added by the servers-side dialog setup function getJavascript_onresize(). In case you only want to execute Javascript in one specific type of dialog you can use the task identifier that is defined in the data object.

Data	Static/Ref	Type	Description
task	static	string	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	string	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.

The following example shows the execution of Javascript in the onresize event of a plugin defined dialog.

```
Pluginexample.prototype.exampleDialogOnResize(data) {

      // Add hidden input fields to example dialog of example plugin
      if (data["task"] == "example_dialog" && data["plugin"] == "example"){

             var input = dhtml.getElementById("inputfieldX");
             input.style.width = window.offsetWidth + "px";

      }

}
```

2.5 client.dialog.general.onresize.before

Name	client.dialog.general.general.onresize.before
Location	/php-webclient/client/layout/dialogs/window.php
Client/Server	Server-side
Category	Dialog

This hook allows the execution of Javascript code when the window is resized. The code is executed just before the onload-code specific for this type of dialog is executed. This last bit of code is added by the servers-side dialog setup function getJavascript_onresize(). In case you only want to execute Javascript in one specific type of dialog you can use the task identifier that is defined in the data object.

Data	Static/Ref	Type	Description
task	static	string	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	string	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.

The following example shows the execution of Javascript in the onresize event of a plugin defined dialog.

```
Pluginexample.prototype.exampleDialogOnLoad(data) {

      // Add hidden input fields to example dialog of example plugin
      if (data["task"] == "example_dialog" && data["plugin"] == "example"){

            var input = dhtml.getElementById("inputfieldX");
            input.value = "Hello world!";

      }

}
```

2.6 client.module.addressbookitem.item.after

Name	client.module.addressbookitemmodule.item.after
Location	/php-webclient/client/module/addressbookitemmodule .js
Client/Server	Client-side
Category	Module

This hook can be used to handle the response from the server with an entry for the addressbook details dialog. This hook can be used to modify the UI based on this data. It will take place after the default actions have been taken.

The addressbookitemmodule can open an entry that is either a mailuser or a group (distlist/AB container). There are two separate dialogs for these types, but they both use the same module.

Data	Static/Ref	Type	Description
module	reference	Object	Reference to the addressbookitemmodule.
object_type	static	Number	The object type of the entry that has been returned by the server.
message	reference	XML DOM Node	Reference to the item portion of the XML that is sent to the client containing all the properties of an item. By changing any of these properties the module will have different data than on the server.

2.7 client.module.addressbookitem.item.before

Name	client.module.addressbookitemmodule.item.before
Location	/php-webclient/client/module/addressbookitemmodule.js
Client/Server	Client-side
Category	Module

This hook can be used to handle the response from the server with an entry for the addressbook details dialog. This hook can be used to modify the UI based on this data. It will take place before the default actions have been taken.

The addressbookitemmodule can open an entry that is either a mailuser or a group (distlist/AB container). There are two separate dialogs for these types, but they both use the same module.

Data	Static/Ref	Type	Description
module	reference	Object	Reference to the addressbookitemmodule.
object_type	static	Number	The object type of the entry that has been returned by the server.
message	reference	XML DOM Node	Reference to the item portion of the XML that is sent to the client containing all the properties of an item. By changing any of these properties the module will have different data than on the server.

2.8 *client.module.contactitemmodule.init.parsingObject*

Name	client.module.contactitemmodule.init.parsingObject
Location	/php-webclient/client/module/contactitemmodule.js
Client/Server	Client-side
Category	Module

This hook can be used to add a custom Contact Parser Object. This Contact Parser Object will overwrite the default behavior of the address parsing, telephone number parsing and the full name parsing.

Data	Static/Ref	Type	Description
parsingObject	reference	Boolean/Object	Set to false by default, but can be overwritten by a plugin to use a custom Contact Parser Object.

The following plugin implements the CustomContactParser object to be used in the contact dialog.

```
function Plugincontactparser(){}

Plugincontactparser.prototype.init = function(){
    this.registerHook("client.module.contactitemmodule.init.parsingObject");
}

Plugincontactparser.prototype.execute = function(eventID, data){

    switch(eventID){

        case "client.module.contactitemmodule.init.parsingObject":

                   data["parsingObject"] = new CustomContactParser() ;

                   break;

    }

}
```

2.9 client.module.contactlistmodule.contextmenu.buildup

Name	client.module.contactlistmodule.contextmenu.buildup
Location	/php-webclient/client/module/contactlistmodule.js
Client/Server	Client-side
Category	Module

This hook can be used to add an item to the contextmenu that is opened when you click with the right mouse button on a contact in any contact list view.

Data	Static/Ref	Type	Description
contextmenu	reference	array	Array of menu items that will be added to the contextmenu. Use the webclient.menu.createMenuItem() function to add a new item.

The following item uses the webclient.menu.createMenuItem() function to add a new item to the contextmenu.

```
var items = data["contextmenu"];

items.push(webclient.menu.createMenuItem(
      "example_save",                 // CSS class of the item in the menu will be
                                      // prepended with icon_. So example will become
                                      // icon_example. The icon in front of the text
                                      // will be added through CSS.

      dgettext("Example Save", "plugin_example"),
                                      // Displayed text in the item

      dgettext("Special example save", "plugin_example"),
                                      // Text that is displayed in the tooltip when
                                      // holding you mouse over the item. Set to false
                                      // when no text needs to be displayed.

      eventListContextMenuOpenMessage, // Event function called when clicked

      "E",                            // The shortcut key that is used to access this
                                      // menu item. Normally no keys are defined for
                                      // this and in that case it is set to false.

      false,                          // Toggle argument that when set to true
                                      // highlights the selected item after it has been
                                      // selected. When the user clicks on it again the
                                      // button is deselected. An example of where this
                                      // is used is the search button in the top menu
                                      // of the main window.

      data                            // JS Object that is added to the HTML DOM
                                      // element under the property element.data and
                                      // can be accessed in the event function.
));
```

To add a separator in the menu You can simply add this item.

```
items.push(webclient.menu.createMenuItem("seperator", ""));
```

2.10 client.module.contactlistmodule.topmenu.buildup

Name	client.module.contactlistmodule.topmenu.buildup
Location	/php-webclient/client/module/contactlistmodule.js
Client/Server	Client-side
Category	Module

This hook can be used to add an item to the top menu that is shown when the contactlistmodule is loaded.

Data	Static/Ref	Type	Description
topmenu	reference	array	Array of menu items that will be added to the top menu. Use the webclient.menu.createMenuItem() function to add a new item.

The following item uses the webclient.menu.createMenuItem() function to add a new item to the top menu.

```
var items = data["topmenu"];

items.push(webclient.menu.createMenuItem(
      "example_save",                  // CSS class of the item in the menu will be
                                       // prepended with icon_. So example will become
                                       // icon_example. The icon in front of the text
                                       // will be added through CSS.

      dgettext("Example Save", "plugin_example"),
                                       // Displayed text in the item

      dgettext("Special example save", "plugin_example"),
                                       // Text that is displayed in the tooltip when
                                       // holding you mouse over the item. Set to false
                                       // when no text needs to be displayed.

      eventListContextMenuOpenMessage, // Event function called when clicked

      "E",                             // The shortcut key that is used to access this
                                       // menu item. Normally no keys are defined for
                                       // this and in that case it is set to false.

      false,                           // Toggle argument that when set to true
                                       // highlights the selected item after it has been
                                       // selected. When the user clicks on it again the
                                       // button is deselected. An example of where this
                                       // is used is the search button in the top menu
                                       // of the main window.

      data                             // JS Object that is added to the HTML DOM
                                       // element under the property element.data and
                                       // can be accessed in the event function.
));
```

To add a separator in the menu You can simply add this item.

```
items.push(webclient.menu.createMenuItem("seperator", ""));
```

2.11 client.module.hierarchymodule.contextmenu.buildup

Name	client.module.hierarchymodule.contextmenu.buildup
Location	/php-webclient/client/module/hierarchymodule.js
Client/Server	Client-side
Category	Module

This hook can be used to add an item to the contextmenu that is opened when you click with the right mouse button on a folder in the hierarchy list.

Data	Static/Ref	Type	Description
items	reference	Array	Array of menu items that will be added to the contextmenu. Use the webclient.menu.createMenuItem() function to add a new item.
folder	static	Object	Object that contains the data of the folder the user clicked on. The format of the folder object is explained below.

The folder object contains the following data about the folder the user clicked on.

```
folder => (

      access => (                               // Value of property PR_ACCESS
             modify              => 1           // (pidTagAccess). When set to 0 that
             read                => 2           // specific operation is not available.
             delete              => 4
             create_hierarchy    => 8
      )

      container_class => "IPF.Note"             // Class of folder. For inbox this is
                                                // IPF.Note and other examples are
                                                // IPF.Apppointment, IPF.StickyNote,
                                                // IPF.Task, IPF.Journal.

      content_count => 101                      // Number of items in the folder.

      content_unread => 5                       // Number of unread items in the folder.

      display_name => "Inbox"                   // Displayed name of the folder.

      entryid => 0000000010fbc34......          // EntryID of the folder.

      parent_entryid => 0000000010fbc34......// EntryID of the parent folder.

      rights => (                               // Value of the property PR_RIGHTS is
             deleteany => "0"                   // (pidTagRights). When set to "0" the
             deleteowned => "16"                // user not allowed to delete their own or
      )                                         // any items from the folder.

      storeid => 0000000038a1bb1005e.......     // EntryID of the store.

      subfolders => "1"                         // Indicator whether there are subfolders
                                                // or not. If set to "1" there are
                                                // subfolders and when set to "-1" there
                                                // are no subfolders.

)
```

The following item uses the webclient.menu.createMenuItem() function to add a new item to the contextmenu.

```
var items = data["contextmenu"];

items.push(webclient.menu.createMenuItem(
      "example_save",                  // CSS class of the item in the menu will be
                                       // prepended with icon_. So example will become
                                       // icon_example. The icon in front of the text
                                       // will be added through CSS.

      dgettext("Example Save", "plugin_example"),
                                       // Displayed text in the item

      dgettext("Special example save", "plugin_example"),
                                       // Text that is displayed in the tooltip when
                                       // holding you mouse over the item. Set to false
                                       // when no text needs to be displayed.

      eventListContextMenuOpenMessage, // Event function called when clicked

      "E",                             // The shortcut key that is used to access this
                                       // menu item. Normally no keys are defined for
                                       // this and in that case it is set to false.

      false,                           // Toggle argument that when set to true
                                       // highlights the selected item after it has been
                                       // selected. When the user clicks on it again the
                                       // button is deselected. An example of where this
                                       // is used is the search button in the top menu
                                       // of the main window.

      data                             // JS Object that is added to the HTML DOM
                                       // element under the property element.data and
                                       // can be accessed in the event function.
));
```

To add a separator in the menu You can simply add this item.

```
items.push(webclient.menu.createMenuItem("seperator", ""));
```

2.12 client.module.hierarchymodule.sharedFoldersPane.buildup

Name	client.module.hierarchymodule.sharedFoldersPane.buildup
Location	/php-webclient/client/module/hierarchymodule.js
Client/Server	Client-side
Category	Module

This hook can be used to quickly add links to the little pane below the hierarchylist in the webaccess interface. Note that this pane is not resized when extra content is added.

Data	Static/Ref	Type	Description
sharedFoldersElement	reference	DOM element	HTML element of the pane that is displayed below the hierarchy list. This pane is used normally used to display the “Shared Folders” link.

2.13 client.module.itemmodule.item.before

Name	client.module.itemmodule.item.before
Location	/php-webclient/client/module/itemmodule.js
Client/Server	Client-side
Category	Module

Some of the derived itemmodule classes have their own implementation of the item method, but they also call back to the item method of their superclass. This is usually done at the end of their method. So this BEFORE hook is not at the start of item method of the derived itemmodule class.

Data	Static/Ref	Type	Description
message	reference	XML DOM Node	Reference to the item portion of the XML that is sent to the client containing all the properties of an item. By changing any of these properties the module will have different data than on the server.

2.14 *client.module.maillistmodule.contextmenu.buildup*

Name	client.module.maillistmodule.contextmenu.buildup
Location	/php-webclient/client/module/maillistmodule.js
Client/Server	Client-side
Category	Module

This hook can be used to add an item to the contextmenu that is opened when you click with the right mouse button on an email in any mail list view.

Data	Static/Ref	Type	Description
contextmenu	reference	array	Array of menu items that will be added to the contextmenu. Use the webclient.menu.createMenuItem() function to add a new item.

The following item uses the webclient.menu.createMenuItem() function to add a new item to the contextmenu.

```
var items = data["contextmenu"];

items.push(webclient.menu.createMenuItem(
      "example_save",                  // CSS class of the item in the menu will be
                                       // prepended with icon_. So example will become
                                       // icon_example. The icon in front of the text
                                       // will be added through CSS.

      dgettext("Example Save", "plugin_example"),
                                       // Displayed text in the item

      dgettext("Special example save", "plugin_example"),
                                       // Text that is displayed in the tooltip when
                                       // holding you mouse over the item. Set to false
                                       // when no text needs to be displayed.

      eventListContextMenuOpenMessage, // Event function called when clicked

      "E",                             // The shortcut key that is used to access this
                                       // menu item. Normally no keys are defined for
                                       // this and in that case it is set to false.

      false,                           // Toggle argument that when set to true
                                       // highlights the selected item after it has been
                                       // selected. When the user clicks on it again the
                                       // button is deselected. An example of where this
                                       // is used is the search button in the top menu
                                       // of the main window.

      data                             // JS Object that is added to the HTML DOM
                                       // element under the property element.data and
                                       // can be accessed in the event function.
));
```

To add a separator in the menu You can simply add this item.

```
items.push(webclient.menu.createMenuItem("seperator", ""));
```

2.15 *client.module.maillistmodule.execute.before*

Name	client.module.maillistmodule.execute.before
Location	/php-webclient/client/module/maillistmodule.js
Client/Server	Client-side
Category	Module

The execute method for the maillistmodule is called when it receives data sent by the server.

Data	Static/Ref	Type	Description
type	static	string	String identifying the type that is sent to by the server. This can be “list” for a list of items, “item” for updating or adding one item to the list, “delete” to delete an item. The previous examples are very basic types and there are more depending on which are implement in each different listmodule.
action	reference	XML DOM Node	Reference to the entire ACTION portion of the XML contain all the data for that set of data. By changing any of available data by the plugin, the module will have different data than on the server.

2.16 client.module.previewreadmailitemmodule.item.before

Name	client.module.previewreadmailitemmodule.item.before
Location	/php-webclient/client/module/previewreadmailitemmodule .js
Client/Server	Client-side
Category	Module

The item method is called when the module receives data of one item from the server. Before the data is processed this hook is called.

Data	Static/Ref	Type	Description
message	reference	XML DOM Node	Reference to the item portion of the XML that is sent to the client containing all the properties of an item. By changing any of these properties the module will have different data than on the server.

2.17 client.module.readmailitemmodule.item.before

Name	client.module.readmailitemmodule.item.before
Location	/php-webclient/client/module/readmailitemmodule .js
Client/Server	Client-side
Category	Module

The item method is called when the module receives data of one item from the server. Before the data is processed this hook is called.

Data	Static/Ref	Type	Description
message	reference	XML DOM Node	Reference to the item portion of the XML that is sent to the client containing all the properties of an item. By changing any of these properties the module will have different data than on the server.

2.18 client.module.readmailitemmodule.setbody.postdisplay

Name	client.module.readmailitemmodule.setbody.postdisplay
Location	/php-webclient/client/module/readmailitemmodule .js
Client/Server	Client-side
Category	Module

This hook gives you the ability to interact with the IFRAME as a DOM object. The difference with the PREDISPLAY hook is that that hook only allows you to change the string that is added to the iframe as content. With this hook you can follow up on content added to the IFRAME by searching for them in the document and add events to links for example.

Data	Static/Ref	Type	Description
iframedocument	reference	DOM Element	Reference to HTML DOM element DOCUMENT of the IFRAME where the body of the item will be displayed in. This is a DOM element so you can apply the DOM functionalities to this element.

This example code shows you a plugin class that wraps all occurrences of the text “TEST” in A-tags. After displaying it searches for them in the IFRAME DOCUMENT and adds event handlers to the links properly using the DHTML library.

```
Plugintest.prototype.execute = function(eventID, data){
      switch(eventID){
             // This hook adds links to specific matching pieces of text
             case "client.module.readmailitemmodule.setbody.predisplay":
                    this.processItemDataPreDisplay(data);
                    break;

             // This hook adds events to the links added by this plugin
             case "client.module.readmailitemmodule.setbody.postdisplay":
                    this.processItemDataPostDisplay(data);
                    break;
      }
}

Plugintest.prototype.processItemDataPreDisplay = function(data){
      // Adding <a> link tag arround any occurrence of the text "TEST".
      data["content"] = data["content"].replace(/(TEST)/g,
             "<a href=\"javascript:void(0);\" target=\"_self\" plugin=\"test\">$1</a>");
}

Plugintest.prototype.processItemDataPostDisplay = function(data){
      // Finding A-tags with the attribute plugin=test added.
      var linksNodes = data["iframedocument"].getElementsByTagName("a");
      for(var i=0;i<linksNodes.length;i++){
             if(linksNodes[i].getAttribute("plugin") == "test"){
                    dhtml.addEvent(-1, linksNodes[i], "click",
                                               eventPlugintestClickBodyLink, window);
             }
      }
}
```

2.19 client.module.readmailitemmodule.setbody.predisplay

Name	client.module.readmailitemmodule.setbody.predisplay
Location	/php-webclient/client/module/readmailitemmodule .js
Client/Server	Client-side
Category	Module

This hook allows you to change the content of the body before it is added to the IFRAME. Since it is only a string you cannot interact with the content as you would be able to do if it was a DOM tree. To do that you need the POSTDISPLAY hook.

Data	Static/Ref	Type	Description
content	reference	string	Reference to the content string that is added to the IFRAME.

This example code shows you a plugin class that wraps all occurrences of the text “TEST” in A-tags. After displaying it searches for them in the IFRAME DOCUMENT and adds event handlers to the links properly using the DHTML library.

```
Plugintest.prototype.execute = function(eventID, data){
      switch(eventID){
             // This hook adds links to specific matching pieces of text
             case "client.module.readmailitemmodule.setbody.predisplay":
                    this.processItemDataPreDisplay(data);
                    break;

             // This hook adds events to the links added by this plugin
             case "client.module.readmailitemmodule.setbody.postdisplay":
                    this.processItemDataPostDisplay(data);
                    break;
      }
}

Plugintest.prototype.processItemDataPreDisplay = function(data){
      // Adding <a> link tag arround any occurrence of the text "TEST".
      data["content"] = data["content"].replace(/(TEST)/g,
             "<a href=\"javascript:void(0);\" target=\"_self\" plugin=\"test\">$1</a>");
}

Plugintest.prototype.processItemDataPostDisplay = function(data){
      // Finding A-tags with the attribute plugin=test added.
      var linksNodes = data["iframedocument"].getElementsByTagName("a");
      for(var i=0;i<linksNodes.length;i++){
             if(linksNodes[i].getAttribute("plugin") == "test"){
                    dhtml.addEvent(-1, linksNodes[i], "click",
                                               eventPlugintestClickBodyLink, window);
             }
      }
}
```

2.20 server.core.properties.addressbookitem.abobject

Name	server.core.properties.addressbookitem.abobject
Location	/php-webclient/server/core/class.properties.php
Client/Server	Server-side
Category	Core

This hook allows you to add your own properties to the list of properties that is retrieved for all the addressbook objects shown in the addressbook details dialogs. The addressbook objects are the list of members, the manager, the owner, etc.

Data	Static/Ref	Type	Description
properties	reference	Array	List of properties that will be loaded for this type of object.

The following example shows how to add properties to the properties array.

```
function addABObjectProps($data) {
      $data['properties']['propertyname'] = "PT_TSTRING:PSETID_Address:0x9999";
      $data['properties']['propertyname2'] = PR_PROPERTY_DEFINITION;
}
```

2.21 server.core.properties.addressbookitem.distlist

Name	server.core.properties.addressbookitem.distlist
Location	/php-webclient/server/core/class.properties.php
Client/Server	Server-side
Category	Core

This hook allows you to add your own properties to the list of properties that is retrieved for the addressbook distlist (or AB container in the case of companies) that is loaded by the addressbookitemmodule in the addressbook details dialog.

Data	Static/Ref	Type	Description
Properties	reference	Array	List of properties that will be loaded for this type of object.

The following example shows how to add properties to the properties array.

```
function addABObjectProps($data) {
      $data['properties']['propertyname'] = "PT_!STRING:PSETID_Address:0x9999";
      $data['properties']['propertyname2'] = PR_PROPERTY_DEFINITION;
}
```

2.22 server.core.properties.addressbookitem.mailuser

Name	server.core.properties.addressbookitem.mailuser
Location	/php-webclient/server/core/class.properties.php
Client/Server	Server-side
Category	Core

This hook allows you to add your own properties to the list of properties that is retrieved for the addressbook mailuser that is loaded by the addressbookitemmodule in the addressbook details dialog.

Data	Static/Ref	Type	Description
Properties	reference	Array	List of properties that will be loaded for this type of object.

The following example shows how to add properties to the properties array.

```
function addABObjectProps($data) {
      $data['properties']['propertyname'] = "PT_!STRING:PSETID_Address:0x9999";
      $data['properties']['propertyname2'] = PR_PROPERTY_DEFINITION;
}
```

2.23 server.dialog.attachments.setup.getbody.uploadformhtml

Name	server.dialog.attachments.setup.getbody.uploadformhtml
Location	/php-webclient/client/layout/dialogs/modal/attachments.php
Client/Server	Server-side
Category	Dialog

This hook allows you to add HTML within the upload form that is used in the attachments dialog. The attachments dialog is a special dialog that submits the form on the page to upload files. When submitting the page only the content of the fields that are inside the FORM tags is sent to the server. Defining fields outside the FORM tags will result in the loss of the information when the entire page refreshes. With this hook you can add hidden fields that will be sent to the server along with the uploaded files.

Data	Static/Ref	Type	Description
html	reference	string	String that can be used to append your own HTML that will be added within the FORM-tags of the upload form in the attachments dialog.

The following example shows the appending of some hidden fields to the attachments dialog.

```
function addDialogHTML($data) {
	$data["html"] .= '<input type="hidden" id="plugin_example_traceid" value="" />';
	$data["html"] .= '<input type="hidden" id="plugin_example_barcode" value="" />';
}
```

2.24 server.dialog.gab_detail_distlist.tabs.htmloutput

Name	server.dialog.gab_detail_distlist.tabs.htmloutput
Location	/php-webclient/client/layout/dialogs/standard/gab_detail_distlist.php
Client/Server	Server-side
Category	Dialog

This hook allows you to add extra tabs to the GAB distlist details dialog. This hook should be used in combination with the hook “server.dialog.gab_detail_distlist.tabs.setup”. The setup hook registers the tabs to the TabBar and the htmloutput hook starts and ends the tab and outputs the HTML for content of the tab.

There is no data in the data object.

The following example shows how to add an extra tab in the GAB distlist details dialog.

```
class PluginABDetailsTab extends Plugin {

      function init(){

            $this->registerHook('server.dialog.gab_detail_distlist.tabs.setup');
            $this->registerHook('server.dialog.gab_detail_distlist.tabs.htmloutput');

      }

      function execute($eventID, &$data){

            switch($eventID){

                  case 'server.dialog.gab_detail_distlist.tabs.setup':
                        $this->setupDialogTab($data);
                        break;

                  case 'server.dialog.gab_detail_distlist.tabs.htmloutput':
                        $this->outputDialogTab($data);
                        break;

            }

      }

      function setupDialogTab(&$data){

            $data['tabs']['tabID'] = dgettext('plugin_abdetailstab', 'Extra data');

      }

      function outputDialogTab(&$data){
            global $tabbar, $tabs;

            $tabbar->beginTab("tabID");

            // Output HTML content for our brand new tab
            echo '<div id="proxy_addresses_table"></div>';

            $tabbar->endTab();

      }

}
```

2.25 server.dialog.gab_detail_distlist.tabs.setup

Name	server.dialog.gab_detail_distlist.tabs.setup
Location	/php-webclient/client/layout/dialogs/standard/gab_detail_distlist.php
Client/Server	Server-side
Category	Dialog

This hook allows you to add extra tabs to the GAB distlist details dialog. This hook should be used in combination with the hook “server.dialog.gab_detail_distlist.tabs.htmloutput”. The setup hook registers the tabs to the TabBar and the htmloutput hook starts and ends the tab and outputs the HTML for content of the tab.

Data	Static/Ref	Type	Description
tabs	reference	Array	Array that you have to use to register your tab. This array is fed into the TabBar. For each new tab you add one element to the array. The key is the tab ID and the value will be the title displayed in the tab.

The following example shows how to add an extra tab in the GAB distlist details dialog.

```
class PluginABDetailsTab extends Plugin {

      function init(){

             $this->registerHook('server.dialog.gab_detail_distlist.tabs.setup');
             $this->registerHook('server.dialog.gab_detail_distlist.tabs.htmloutput');

      }

      function execute($eventID, &$data){

             switch($eventID){

                    case 'server.dialog.gab_detail_distlist.tabs.setup':
                          $this->setupDialogTab($data);
                          break;

                    case 'server.dialog.gab_detail_distlist.tabs.htmloutput':
                          $this->outputDialogTab($data);
                          break;

             }

      }

      function setupDialogTab(&$data){

             $data['tabs']['tabID'] = dgettext('plugin_abdetailstab', 'Extra data');

      }

      function outputDialogTab(&$data){
             global $tabbar, $tabs;

             $tabbar->beginTab("tabID");

             // Output HTML content for our brand new tab
             echo '<div id="proxy_addresses_table"></div>';

             $tabbar->endTab();

      }
}
```

2.26 server.dialog.gab_detail_mailuser.tabs.htmloutput

Name	server.dialog.gab_detail_mailuser.tabs.htmloutput
Location	/php-webclient/client/layout/dialogs/standard/gab_detail_mailuser.php
Client/Server	Server-side
Category	Dialog

This hook allows you to add extra tabs to the GAB mailuser details dialog. This hook should be used in combination with the hook “server.dialog.gab_detail_mailuser.tabs.setup”. The setup hook registers the tabs to the TabBar and the htmloutput hook starts and ends the tab and outputs the HTML for content of the tab.

There is no data in the data object.

The following example shows how to add an extra tab in the GAB mailuser details dialog.

```
class PluginABDetailsTab extends Plugin {
      function init(){
            $this->registerHook('server.dialog.gab_detail_mailuser.tabs.setup');
            $this->registerHook('server.dialog.gab_detail_mailuser.tabs.htmloutput');
      }
      function execute($eventID, &$data){
            switch($eventID){
                  case 'server.dialog.gab_detail_mailuser.tabs.setup':
                        $this->setupDialogTab($data);
                        break;
                  case 'server.dialog.gab_detail_mailuser.tabs.htmloutput':
                        $this->outputDialogTab($data);
                        break;
            }
      }
      function setupDialogTab(&$data){
            $data['tabs']['tabID'] = dgettext('plugin_abdetailstab', 'Extra data');
      }
      function outputDialogTab(&$data){
            global $tabbar, $tabs;
            $tabbar->beginTab("tabID");
            // Output HTML content for our brand new tab
            echo '<div id="proxy_addresses_table"></div>';
            $tabbar->endTab();
      }
}
```

2.27 server.dialog.gab_detail_mailuser.tabs.setup

Name	server.dialog.gab_detail_mailuser.tabs.setup
Location	/php-webclient/client/layout/dialogs/standard/gab_detail_mailuser.php
Client/Server	Server-side
Category	Dialog

This hook allows you to add extra tabs to the GAB mailuser details dialog. This hook should be used in combination with the hook “server.dialog.gab_detail_mailuser.tabs.htmloutput”. The setup hook registers the tabs to the TabBar and the htmloutput hook starts and ends the tab and outputs the HTML for content of the tab.

Data	Static/Ref	Type	Description
tabs	reference	Array	Array that you have to use to register your tab. This array is fed into the TabBar. For each new tab you add one element to the array. The key is the tab ID and the value will be the title displayed in the tab.

The following example shows how to add an extra tab in the GAB mailuser details dialog.

```
class PluginABDetailsTab extends Plugin {

      function init(){

            $this->registerHook('server.dialog.gab_detail_mailuser.tabs.setup');
            $this->registerHook('server.dialog.gab_detail_mailuser.tabs.htmloutput');

      }

      function execute($eventID, &$data){

            switch($eventID){

                  case 'server.dialog.gab_detail_mailuser.tabs.setup':
                        $this->setupDialogTab($data);
                        break;

                  case 'server.dialog.gab_detail_mailuser.tabs.htmloutput':
                        $this->outputDialogTab($data);
                        break;

            }

      }

      function setupDialogTab(&$data){

            $data['tabs']['tabID'] = dgettext('plugin_abdetailstab', 'Extra data');

      }

      function outputDialogTab(&$data){
            global $tabbar, $tabs;

            $tabbar->beginTab("tabID");

            // Output HTML content for our brand new tab
            echo '<div id="proxy_addresses_table"></div>';

            $tabbar->endTab();

      }

}
```

2.28 server.dialog.general.include.cssfiles

Name	server.dialog.general.include.cssfiles
Location	/php-webclient/client/layout/dialogs/window.php
Client/Server	Server-side
Category	Dialog

This hook allows a plugin to add paths to CSS files the dialog should load. This is not meant for local CSS files, because those can be include by defining them in the manifest. This is for loading in external CSS files. With this hook you can define the full URL to the file. Something that is not possible in the manifest as the path is always prepended with the path to the plugins directory.

Data	Static/Ref	Type	Description
task	static	string	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	string	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.
files	reference	array	Array of files that will be included as CSS files when the dialog HTML page is loaded by the browser.

The following example code shows the including of an external CSS file.

```
function onDialogLoadIncludeCSSFiles($data, $apiKey){
   $data['files'][] = 'http://www.zarafa.com/specialstyles.css;
}
```

2.29 server.dialog.general.include.jsfiles

Name	server.dialog.general.include.jsfiles
Location	/php-webclient/client/layout/dialogs/window.php
Client/Server	Server-side
Category	Dialog

This hook allows a plugin to add paths to Javascript files the dialog should load. This is not meant for local Javascript files, because those can be include by defining them in the manifest. This is for loading in external Javascript files. With this hook you can define the full URL to the file. Something that is not possible in the manifest as the path is always prepended with the path to the plugins directory.

Data	Static/Ref	Type	Description
task	static	string	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	string	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.
files	reference	array	Array of files that will be included as Javascript files when the dialog HTML page is loaded by the browser.

The following example code shows the including of an external Javascript file: The Google Maps API.

```
function onDialogLoadIncludeJsFiles($data, $apiKey){
   $data['files'][] = 'http://maps.google.com/maps?file=api&amp;v=2&amp;key='.$apiKey;
}
```

2.30 server.dialog.general.setup.getbody.before

Name	server.dialog.general.setup.getbody.before
Location	/php-webclient/client/layout/dialogs/window.php
Client/Server	Server-side
Category	Dialog

This hook allows you to add HTML before the dialog setup function getBody() has called. This can be used to add hidden fields to the dialog. In case you only want to add HTML to one specific type of dialog you can use the task identifier that is also defined in the data object.

Data	Static/Ref	Type	Description
task	static	string	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	string	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.
html	reference	string	String that can be used to append your own HTML that will be added before the getBody() function adds the HTML.

The following example shows the appending of some hidden fields to a plugin defined dialog.

```
function addDialogHTML($data) {

	// Add hidden input fields to example dialog of example plugin
	if ($data["task"] == "example_dialog" && $data["plugin"] == "example"){

		$data["html"] .=
			'<input type="hidden" id="plugin_example_traceid" value="" />';

		$data["html"] .=
			'<input type="hidden" id="plugin_example_barcode" value="" />';

	}

}
```

2.31 server.dialog.general.setup.getincludes

Name	server.dialog.general.setup.getincludes
Location	/php-webclient/client/layout/dialogs/window.php
Client/Server	Server-side
Category	Dialog

This hook allows you to add components to this dialog. This is primarialy used for native webaccess components. Plugin files can be added by defining them in the manifest. It is possible that you may only want to add certain plugin components for one specific type of dialog. In that case it is also possible to use this hook as the task identifier is also defined in the data object.

Data	Static/Ref	Type	Description
task	static	string	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	string	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.
includes	reference	array	Array of files that will be included when the dialog HTML page is loaded by the browser. It is possible to add both Javascript files and CSS files.

The following example shows the loading of native webaccess components in a plugin defined dialog.

```
function dialogGetIncludes($data) {

      // Load suggestion functionality in example dialog of example plugin
      if ($data["task"] == "example_dialog" && $data["plugin"] == "example"){

            $data["includes"][] = "client/widgets/suggestionlist.js";
            $data["includes"][] = "client/modules/suggestemailaddressmodule.js";
            $data["includes"][] = "client/layout/css/suggestionlayer.css";

      }

}
```

The following example shows the loading of two plugin files for the “createmail” dialog only. Note the use of the getPluginPath method.

```
function dialogGetIncludes($data) {

      // ignore all dialogs except "createmail"
      if ($data["task"] != "createmail"){
            return;
      }

      $data["includes"][] = $this->getPluginPath(). "plugin.example.js";
      $data["includes"][] = $this->getPluginPath(). "plugin.example.css";

}
```

2.32 server.dialog.general.setup.getmenubuttons

Name	server.dialog.general.setup.getmenubuttons
Location	/php-webclient/client/layout/dialogs/window.php
Client/Server	Server-side
Category	Dialog

This hook allows you to add buttons to the top menu in all the dialogs.

Data	Static/Ref	Type	Description
task	static	string	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	string	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.
buttons	reference	array	Array that contains the information of the menu items that need to be added to the top menu in the dialogs.

The following example shows how to add menu items to the dialogs.

```
function addDialogMenuItems($data) {

      // ignore all dialogs except "createmail"
      if ($data["task"] != "createmail") return;
      $data["buttons"][] = array(
             'id' => "seperator",             // This is how a seperator is added.
             'name' => "",
             'title' => "",
             'callback' => ""
      );

      $data["buttons"][] = array(
             'id' => "examplelink",           // HTML ID and CSS class of the menu item.
                                              // The CSS class will be prepended with
                                              // icon_. Sp example will become
                                              // icon_example. The icon in front of the
                                              // text will be added through CSS.

             'name' => dgettext("plugin_example", "Example Link"),
                                              // Displayed text in the item.

             'title' => dgettext("plugin_example", "Link to example"),
                                              // Text that is displayed in the thooltip
                                              // when holding your mouse over the item.
                                              // Leave string empty ("") when no text
                                              // needs to be displayed.

             'callback' => "eventPluginexampleClicked"
                                              // Event function that is called when
                                              // clicking on this item.
      );

}
```

The following example shows that it is also possible to add javascript code to the callback. This code opens first flags the message as changed and then opens a modal dialog.

```
      'id'=>"options",
      'name'=>dgettext("Options", "plugin_example"),
      'title'=>"",
      'callback'=>"function(){setMessageChanged();webclient.openModalDialog(module,
'options', DIALOG_URL+'task=mailoptions_modal', 310, 220, mailOptionsCallBack);}"
```

2.33 *server.dialog.general.setup.getmodulename*

Name	server.dialog.general.setup.getmodule
Location	/php-webclient/client/layout/dialogs/window.php
Client/Server	Server-side
Category	Dialog

This hook allows you to change the module that is used in any dialog. By checking the task of the dialog you can change a module for a specific dialog. The common practice is to extend the original modules and override methods to change the behavior. In this case it is important not to forget to include the superclass modules.

Also when you want to communicate to the server you need to have a corresponding server-side module. To make this work you need to extend that server-side module as well.

Data	Static/Ref	Type	Description
task	static	String	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	String	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.
moduleName	reference	String	Contains the name of the module that will be instantiated for this dialog. By changing this you can change the module that will be used.

The example below shows how to extend the client-side addressbookitemmodule for the addressbook detail dialogs for a mailuser and a distlist. These are two separate dialogs.

```
class PluginSpecialABDetails extends Plugin {

	function init(){

		$this->registerHook('server.dialog.general.setup.getmodule');
		$this->registerHook('server.dialog.general.include.jsfiles');

	}

	function execute($eventID, &$data){

		switch($eventID){

			case 'server.dialog.general.setup.getmodule':
				$this->getDialogModuleName($data);
				break;

			case 'server.dialog.general.include.jsfiles':
				$this->onDialogLoadIncludeJsFiles($data);
				break;

		}

	}

	function getDialogModuleName(&$data){

		switch($data['task']){
			case 'gab_details_mailuser':
			case 'gab_details_distlist':
				$tabs['moduleName'] = 'specialabitemmodule';
				break;
		}
	}

	function onDialogLoadIncludeJsFiles($data){

		// Include the superclass
		$data['files'][] = 'client/modules/addressbookitemmodule.js';

		/*
		 * The dialog code looks for the module in the modules directory of the
		 * webaccess. He will not find it in there as it is in the plugin
		 * directory. Therefor we have to include it properly here.
		 */
		$data['files'][] = $this->getPluginPath() . 'mods/specialabitemmodule.js';

	}

}
```

2.34 server.dialog.general.upload_attachment.after

Name	server.dialog.general.upload_attachment.after
Location	/php-webclient/client/dialog.php
Client/Server	Server-side
Category	Dialog

When a dialog posts attachments to the server, it will post it to the dialog URL. The main example of this behavior is the attachments dialog that uploads the files by submitting a form that will refresh the entire dialog page. The dialog will post to the DIALOG URL so it will return to a user interface when the page is loaded.

One of the first actions that happen when the dialog URL is requested is checking if the upload attachment script should be included. When this is the case this hook is triggered just before the include statement. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far.

This hook will also be triggered when a wrong dialog task is opened. That means that if no dialog can be found the attachments will be properly uploaded, but the dialog page will display an error message telling the user that the dialog does not exist.

Data	Static/Ref	Type	Description
task	static	string	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	string	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.

2.35 server.dialog.general.upload_attachment.before

Name	server.dialog.general.upload_attachment.before
Location	/php-webclient/client/dialog.php
Client/Server	Server-side
Category	Dialog

When a dialog posts attachments to the server, it will post it to the dialog URL. The main example of this behavior is the attachments dialog that uploads the files by submitting a form that will refresh the entire dialog page. The dialog will post to the DIALOG URL so it will return to a user interface when the page is loaded.

One of the first actions that happen when the dialog URL is requested is checking if the upload attachment script should be included. When this is the case this hook is triggered just before the include statement. It can be used to initialize any object or load data that is needed.

This hook will also be triggered when a wrong dialog task is opened. That means that if no dialog can be found the attachments will be properly uploaded, but the dialog page will display an error message telling the user that the dialog does not exist.

Data	Static/Ref	Type	Description
task	static	string	Contains the task of the dialog identifying the type of window that has been opened.
plugin	static	string	Contains the name of the plugin that has added this dialog. If set to false it is a native webaccess dialog.

2.36 server.index.load.dialog.after

Name	server.index.load.dialog.after
Location	/php-webclient/index.php
Client/Server	Server-side
Category	Index

This hook is triggered after all the code to load the dialog has been run. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far.

There is no data in the data object.

2.37 server.index.load.dialog.before

Name	server.index.load.dialog.before
Location	/php-webclient/index.php
Client/Server	Server-side
Category	Index

This hook is triggered before all the code to load the dialog is run. It can be used to initialize any object or load data that is needed.

There is no data in the data object.

2.38 server.index.load.download_attachment.after

Name	server.index.load.download_attachment.after
Location	/php-webclient/index.php
Client/Server	Server-side
Category	Index

This hook is triggered after all the code to download an attachment has been run. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far. In this case it can also be used keep track of the downloaded files. Note that you cannot do any outputting at this point when a file has been downloaded.

There is no data in the data object.

2.39 server.index.load.download_attachment.before

Name	server.index.load.download_attachment.before
Location	/php-webclient/index.php
Client/Server	Server-side
Category	Index

This hook is triggered before the code to download an attachment will be run. It can be used to initialize any object or load data that is needed.

There is no data in the data object.

2.40 server.index.load.jstranslations.after

Name	server.index.load.jstranslations.after
Location	/php-webclient/index.php
Client/Server	Server-side
Category	Index

This hook is triggered after all the code to load the translations for the client-side has been run. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far.

There is no data in the data object.

2.41 server.index.load.jstranslations.before

Name	server.index.load.jstranslations.before
Location	/php-webclient/index.php
Client/Server	Server-side
Category	Index

This hook is triggered before the the code to load the translations for the client-side will be run. It can be used to initialize any object or load data that is needed.

There is no data in the data object.

2.42 server.index.load.main.after

Name	server.index.load.main.after
Location	/php-webclient/index.php
Client/Server	Server-side
Category	Index

This hook is triggered after all the code to load the main window has been run. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far.

There is no data in the data object.

2.43 server.index.load.main.before

Name	server.index.load.main.before
Location	/php-webclient/index.php
Client/Server	Server-side
Category	Index

This hook is triggered before the code to load the main window is run. It can be used to initialize any object or load data that is needed.

There is no data in the data object.

2.44 server.index.load.upload_attachment.after

Name	server.index.load.upload_attachment.after
Location	/php-webclient/index.php
Client/Server	Server-side
Category	Index

This hook is triggered when the script to upload an attachment is called through the index.php file. Normally the attachments dialog is used to upload an attachment and in that case this hook will not be triggered.

The attachment dialog posts the uploaded file to the dialog URL. This will reload the entire dialog and in order to return to the attachments dialog user interface it cannot call the upload attachment script. That is why this script is included when files are posted in a dialog.

This hook is currently only triggered when the the ZarafaDnD Firefox extension is used. The extension allows the user to drag a file from their desktop to the browser to upload it. The URL that is used to post the file calls the upload attachment script from the index.php file.

This hook is triggered when the code to upload an attachment has been run. It can be used to clean up any data that has to be discarded or to process information that has been added to the session so far.

There is no data in the data object.

2.45 server.index.load.upload_attachment.before

Name	server.index.load.upload_attachment.before
Location	/php-webclient/index.php
Client/Server	Server-side
Category	Index

This hook is triggered when the script to upload an attachment is called through the index.php file. Normally the attachments dialog is used to upload an attachment and in that case this hook will not be triggered.

The attachment dialog posts the uploaded file to the dialog URL. This will reload the entire dialog and in order to return to the attachments dialog user interface it cannot call the upload attachment script. That is why this script is included when files are posted in a dialog.

This hook is currently only triggered when the the ZarafaDnD Firefox extension is used. The extension allows the user to drag a file from their desktop to the browser to upload it. The URL that is used to post the file calls the upload attachment script from the index.php file.

This hook is triggered before the code to upload an attachment will be run. It can be used to initialize any object or load data that is needed.

There is no data in the data object.

2.46 server.index.login.success

Name	server.index.login.success
Location	/php-webclient/index.php
Client/Server	Server-side
Category	Index

This hook is triggered when the user has logged in successfully and will be redirected to the webaccess. The redirection will happen when the $_GET variable contains the key “logon”.

There is no data in the data object.

2.47 server.main.load.include.cssfiles

Name	server.main.load.include.cssfiles
Location	/php-webclient/client/webclient.php
Client/Server	Server-side
Category	Main

This hook allows a plugin to add paths to CSS files the main window should load. This is not meant for local CSS files, because those can be include by defining them in the manifest. This is for loading in external files. With this hook you can define the full URL to the file. Something that is not possible in the manifest as the path is always prepended with the path to the plugins directory.

Data	Static/Ref	Type	Description
files	reference	array	Array of files that will be included as CSS files when the main window HTML page is loaded by the browser.

The following example code shows the including of an external CSS file.

```
function onMainLoadIncludeCSSFiles($data){
   $data['files'][] = 'http://www.zarafa.com/example_styles.css;
}
```

2.48 server.main.load.include.jsfiles

Name	server.main.load.include.jsfiles
Location	/php-webclient/client/webclient.php
Client/Server	Server-side
Category	Main

This hook allows a plugin to add paths to Javascript files the main window should load. This is not meant for local Javascript files, because those can be include by defining them in the manifest. This is for loading in external files. With this hook you can define the full URL to the file. Something that is not possible in the manifest as the path is always prepended with the path to the plugins directory.

Data	Static/Ref	Type	Description
files	reference	array	Array of files that will be included as Javascript files when the main window HTML page is loaded by the browser.

The following example code shows the including of an external Javascript file: The Google Maps API.

```
function onMainLoadIncludeJsFiles($data, $apiKey){
   $data['files'][] = 'http://maps.google.com/maps?file=api&amp;v=2&amp;key='.$apiKey;
}
```

2.49 server.module.addressbookitemmodule.open.props

Name	server.module.addressbookitemmodule.open.props
Location	/php-webclient/server/module/class.addressbookitemmodule.php
Client/Server	Server-side
Category	Module

This hook allows you to add and modify the list of properties that will be send to the client.

Data	Static/Ref	Type	Description
moduleObject	reference	object	Reference to the itemmodule object.
abentry	static	MAPI object	The MAPI object that is opened from the addressbook.
object_type	static	Integer	The PR_OBJECT_TYPE of the opened addressbook entry.
messageprops	static	Array	The list of properties retrieved for this addressbook entry.
props	static	Array	The list of properties that will be returned to the client. This list is already in the proper format to be converted to the protocol used in the transport to the client.

2.50 server.module.itemmodule.open.after

Name	server.module.maillistmodule.execute.after
Location	/php-webclient/server/module/class.itemmodule.php
Client/Server	Server-side
Category	Module

This hook allows to run code in the open method of the itemmodule after all the default actions to open a message have taken place. With this hook you can still change the data that will be returned to the client.

Data	Static/Ref	Type	Description
moduleObject	reference	object	Reference to the itemmodule object.
store	static	MAPI object	MAPI Message Store object of the opened message.
entryid	static	string	The entryid of the opened message.
action	static	array	The action data sent by the client.
message	reference	MAPI object	The MAPI Message object of the opened message.
data	reference	array	The data that will be sent to the client after the hook has finished.

2.51 server.module.maillistmodule.execute.after

Name	server.module.maillistmodule.execute.after
Location	/php-webclient/server/module/class.maillistmodule.php
Client/Server	Server-side
Category	Module

This hook allows to run code before the requests to the maillistmodule are handled in the execute method.

Data	Static/Ref	Type	Description
moduleObject	reference	object	Reference to the maillistmodule object at the moment the execute is called. This object will contain all the request information under the data property.

2.52 server.module.maillistmodule.execute.before

Name	server.module.maillistmodule.execute.before
Location	/php-webclient/server/module/class.maillistmodule.php
Client/Server	Server-side
Category	Module

This hook allows to run code after the requests to the maillistmodule have been handled in the execute method.

Data	Static/Ref	Type	Description
moduleObject	reference	object	Reference to the maillistmodule object at the moment the execute is called. This object will contain all the request information under the data property.

3 How to implement your own hooks

It is also possible to define hooks in your own plugins. This way you can let other plugins extend your functionality. Hooks can only be defined in code that is run after the Plugin Manager has initialized the plugins. This means that hooks that are not placed inside a function or class are run before the plugins can register to them. The code outside functions is execute the moment the files are included.

3.1 Server-side hook

The following code triggers a hook on the server-side using PHP code.

```
$data = Array(
      'moduleObject' =>& $this,        // Reference to module object
      'properties'   =>& $properties   // Reference to variable used in module
      'actiontype'   => $actiontype    // Static data
);

$GLOBALS['PluginManager']-
>triggerHook("server.plugin.alfresco.module.alfrescomodule.execute.before", $data);
```

The first part prepares the data that will be supplied to the Plugin Class. This must always be an array. The “moduleObject” data is a reference to the module object and the Plugin Class is able to change that data. To learn more on how to pass variables as a reference in PHP go to http://www.php.net/references. The “properties” data is passed as reference as well only this is not an object, but just a variable used in one of the methods of the module object. When this variable is changed by the Plugin Class it is also changed in this method. The “actiontype” data is copied and when changed by the Plugin Class the data is not changed in the module.

The second part contains a call to the triggerHook function of the Plugin Manager. Once instantiated the Plugin Manager can be accessed everywhere on the server using the *$GLOBALS['PluginManager']* variable. The first argument is the eventID that identifies the hook that you are triggering. It is important to keep inline with the naming conventions of the hooks. Later on in this chapter you can read more on how to name your hook. The second argument is the data array that is passed on to the Plugin Class. This must always be an array except when you do not want to pass on any data. In that case you can omit the second argument.

3.2 Client-side hook

The following code triggers a hook on the client-side using Javascript code.

```
var data = new Object();
data["type"] = type;            // Static (String)
data["action"] = action;        // Reference (Object)
data["status"] = status;        // Documented as reference (String)

webclient.pluginManager.triggerHook('client.plugin.alfresco.module.alfrescomodule.execu
te.before', data);

status = data["status"];        // String cannot be referenced so retrieve it by hand
```

The first part is preparing the data that will be supplied to the Plugin Class. This must always be an Object. As described for the server-side hook there is static and there is referenced data. In Javascript only objects, arrays and functions are passed as reference by default. All other types are cloned. The “type” variable is static as it is a string and the “action” is referenced as it is an XML DOM object. The third variable “status” is a string as well, but unlike “type” we want the Plugin Class to be able to change it. In order to do that you have to retrieve the “status” value from the *data* object and put it in the *status* variable by hand as is done after the call to the *triggerHook* method.

The call to the triggerHook function of the Plugin Manager is done in the second part. Once instantiated the Plugin Manager can be accessed everywhere on the client using the *webclient.pluginManager* variable. The first argument is the eventID that identifies the hook that you are triggering. It is important to keep inline with the naming conventions of the hooks. Later on in this chapter you can read more on how to name your hook. The second argument is the data array that is passed on to the Plugin Class. This must always be an object except when you do not want to pass on any data. In that case you can omit the second argument.

3.3 Hook naming conventions

Hooks are identified by their eventID. This is a string identifier and is formatted like the following examples.

```
client.module.hierarchymodule.contextmenu.buildup
client.module.readmailitemmodule.setbody.postdisplay
server.module.maillistmodule.execute.before
client.plugin.alfresco.module.alfrescospacemodule.execute.before
server.plugin.alfresco.module.alfrescospacemodule.execute.before
```

Every eventID is written in small caps and starts with the either *server* or *client*. For the sake of stating the obvious: this indicates whether the hook is triggered on the server-side or on the client-side.

The second part indicates what component the hook is triggered in.

Component		
Name	**Used when**	**Server/Client**
core	Hook is triggered in core library of the webaccess.	Both
dialog	Hook is triggered inside a dialog.	Both
download_attachment	Hook is triggered in the script that downloads an attachment.	Both
index	Hook is triggered in the entry script index.php.	Server
main	Hook is triggered when loading the main window.	Both
module	Hook is triggered inside a module.	Both
plugin	Hook is triggered in a plugin.	Both
pluginclass	Hook is triggered in sidea Plugin Class.	Both
upload_attachment	Hook is triggered in the script that uploads an attachment.	Server
view	Hook is triggered inside a view.	Client
widget	Hook is triggered inside a widget	Client

The naming of the the rest of the hooks depends on the structure of the selected component. When you want to place hooks in your own plugin this means the component your will be using is: plugin. Even if your plugin adds a module and you trigger a hook in that class.

Let's say your plugin is called *alfresco* and you add a server-side module called *alfrescospacemodule*. To visualize this the following code shows that module.

```
class AlfrescoSpaceModule extends Module {

      /**
       * Constructor
       * @param int $id unique id.
       * @param array $data list of all actions.
       */

      function AlfrescoSpaceModule($id, $data){
            $this->properties = Array();
            $this->sort = array();
            parent::Module($id, $data);
      }

      /**
       * Executes all the actions
       * Triggers a hook to be processed in plugin.alfresco.php
       *
       * @return boolean true
       */
      function execute(){
            $data = Array('moduleObject' =>& $this);
            $GLOBALS['PluginManager']->triggerHook(
                   "server.plugin.alfresco.module.alfrescospacemodule.execute.before",
                    $data);

            // (...)

            return true;
      }

}
```

In the execute method in the *alfrescospacemodule* a hook is triggered. The hook starts with *server*. The next part tells that it is a plugin. The third part says it is the *alfresco* plugin. The fourth part describes it as a *module* and the fifth element specifies it is the *alfrescospacemodule*. The second to last element describes the function where the hook is triggered: *execute*. The example above shows the hook is triggered before the rest of the code in the execute function is run. Therefor hook eventID ends with *before* indicating just that.

When adding hooks to other components in your own plugin, the following identifiers show how the naming convention must be applied here.

```
cient.plugin.alfresco.view.alfrescofilemanagerview.updateitem

server.plugin.alfresco.dialog.displayfilecontents.setup.getbody.addspecialhtml

client.plugin.alfresco.widget.alfrescofilemanagertree.addItem

// There is always only one plugin class defined for each plugin, so it is
// not required add the name of your Plugin Class to the eventID. By just
// adding the component name pluginclass there is no need to add the name
// as well since there are never two Plugin Classes per plugin. The
// dosomethingmethod mentioned below is a method and not the name of the
// Plugin Class.
server.plugin.alfresco.pluginclass.dosomethingmethod
```

When adding a hook to a dialog you can must also include the name of the dialog within the name of the hook. The following example shows how this works.

```
server.plugin.alfresco.dialog.displayfilecontents.setup.getbody.addspecialhtml
```

When placing a hook that is triggered right before or after the code inside a function is run, it is preferred to use the words *before* and *after* in the name of the eventID. This could mean that you use *before* right before the function is called or as one of the first lines inside the function. By using these keywords instead of start/end or other variations on that we can ensure the consistency of the hook names.

```
server.plugin.alfresco.module.alfrescospacemodule.execute.after

server.plugin.alfresco.module.alfrescospacemodule.execute.before
```