



Blesta Module Documentation

Note: Last Updated August 20, 2007.

* signifies optional parameter

Constructor:

Module(tableName, *add, *edit, *delete)

Function List:

addDBField(type, name, friendlyName, *defaultValue)
addModuleHTMLField(type, friendlyName, dbFieldName, *regex, *errorMessage)
addModuleHTMLSelectOption(dbFieldName, optionName, optionValue)
addPackageHTMLField(type, friendlyName, dbFieldName, *regex, *errorMessage)
addPackageHTMLSelectOption(dbFieldName, optionName, optionValue)
addPackageNote(dbFieldName, value)
addServiceColumn(dbFieldName, friendlyName)
addServiceEditField(dbFieldName, friendlyName)
addServiceHTMLField(type, friendlyName, dbFieldName, *regex, *errorMessage)
addServiceHTMLSelectOption(dbFieldName, optionName, optionValue)
addServiceTag(dbFieldName, tagName)
addUserServiceHTMLField(type, friendlyName, dbFieldName, *regex, *errorMessage)
addUserServiceHTMLSelectOption(dbFieldName, optionName, optionValue)
createModule()
getDBFields()
getModuleHTMLFields()
getModuleRowSelect()
getPackageHTMLFields()
getRows(*conditions)
getServiceHTMLFields()
getServiceColumns()
getServiceEditFields()
getServiceTags()
getTableName()
getTableSettings()
getUserServiceHTMLFields()
setModuleRowSelect(*friendlyName, *dbFieldName)
updateDBFields(name, value, id)

Function Descriptions:

addDBField(type, name, friendlyName, *defaultValue)

Returns: void

Description:

Adds a database field to the table given by **name** that is of the given field **type**. **friendlyName** specifies the name shown whenever this database field is displayed. **defaultValue** is the default value used when a row is created with this module.

Usage:

```
myModule->addDBField("varchar(255)", "user", "User");
```

addModuleHTMLField(type, friendlyName, dbFieldName, *regex, *errorMessage)

Returns: void

Description:

Creates an HTML field that will be used to create new module rows, or edit existing module rows. **type** must be of a given set of HTML field types. For a complete listing see Appendix A. **dbFieldName** must match the **name** parameter of an element defined by the addDBField function. **regex** must be a regular expression compatible with the PHP defined function `ereg`. **errorMessage** is the message that is displayed if the input for this field does not match the regular expression defined by **regex**.

Usage:

```
myModule->addModuleHTMLField("text", "User Name", "user");
```

addModuleHTMLSelectOption(dbFieldName, optionName, optionValue)

Returns: void

Description:

Adds option fields to the HTML select field. **dbFieldName** must match a field that has been defined by the addModuleHTMLField function with **type** "select". **optionName** is the text that is displayed in the list for this select field. **optionValue** is the value returned when this option is selected.

Usage:

```
myModule->addModuleHTMLSelectOption("users", "Yes", "true");
```

addPackageHTMLField(type, friendlyName, dbFieldName, *regex, *errorMessage)

Returns: void

Description:

Creates an HTML field that will be used to create new packages, or edit existing packages using this module. **type** must be of a given set of HTML field types. For a complete listing see Appendix A. **dbFieldName** must be "mstring". **regex** must be a regular expression compatible with the PHP defined function ereg. **errorMessage** is the message that is displayed if the input for this field does not match the regular expression defined by **regex**.

Usage:

```
myModule : $module->addPackageHTMLField("select", "Plan Name", "mstring", "^.", "You must select a valid Plan Name.");
```

addPackageHTMLSelectOption(dbFieldName, optionName, optionValue)

Returns: void

Description:

Adds option fields to the HTML select field. **dbFieldName** must match a field that has been defined by the addPackageHTMLField function with **type** "select". **optionName** is the text that is displayed in the list for this select field. **optionValue** is the value returned when this option is selected.

Usage:

```
myModule : $module-> addPackageHTMLSelectOption ("mstring", "Plan1", "pl an1");
```

addPackageNote(dbFieldName, value)

Returns: void

Description:

Adds a note element that is associated with the HTML field (**dbFieldName**) created using the addPackageHTMLField function. **value** is the text that is displayed.

Usage:

```
myModule : $module-> addPackageNote ("mstring", "Plan Name is required.");
```

addServiceColumn(dbFieldName, friendlyName)

Returns: void

Description:

Creates a column associated with **dbFieldName** that is displayed for any serviced created using this module. **friendlyName** is the heading used to display the client's information stored under the **dbFieldName**. **dbFieldName** values may be any of the following: "user1", "user2", "pass", "opt1", "opt2", "notes".

Usage:

```
mymodule : $module->addServiceColumn("user2", "User Name");
```

addServiceEditField(dbFieldName, friendlyName)

Returns: void

Description:

Creates a field that may be used to perform a local edit on a service created using this module. **dbFieldName** is the service table database field that may be modified. **friendlyName** is the field identifier for this element. **dbFieldName** values may be any of the following: "user1", "user2", "pass", "opt1", "opt2", "datep", "dated", "dater", "notes".

Usage:

```
mymodule : $module->addServiceEditField("user2", "User Name");
```

addServiceHTMLField(type, friendlyName, dbFieldName, *regex, *errorMessage)

Returns: void

Description:

Creates an HTML field that will be used to add new services from the admin interface. **type** must be of a given set of HTML field types. For a complete listing see Appendix A. If **dbFieldName** is anything other than "user1", "user2", "pass", "opt1", "opt2" then that field will not be stored in the database under the services table and may only be used in this module file. **regex** must be a regular expression compatible with the PHP defined function `ereg`. **errorMessage** is the message that is displayed if the input for this field does not match the regular expression defined by **regex**.

If the third parameter is anything other than user1, user2, pass, opt1, opt2, it must be formatted as an element of the detail array (ex. detail[emaiaddress]). Any field other than user1, user2, pass, opt1, or opt2 supplied for the third parameter will not be stored in the database, and may only be used in this module file.

Usage:

```
mymodule : $module->addServiceHTMLField("text", "Password", "pass", "^. {4, 16}$",  
"Passwords must be between 4 and 16 characters.");
```

addServiceHTMLSelectOption(dbFieldName, optionName, optionValue)

Returns: void

Description:

Adds option fields to the HTML select field. **dbFieldName** must match a field that has been defined by the addServiceHTMLField function with **type** "select". **optionName** is the text that is displayed in the list for this select field. **optionValue** is the value returned when this option is selected.

Usage:

```
mymodule : $module-> addServiceHTMLSelectOption ("opt1", "Active", "true");
```

addServiceTag(dbFieldName, tagName)

Returns: void

Description:

Adds a tag that may be used in welcome e-mails for services that are created using this module. **dbFieldName** values may be any of the following: "user1", "user2", "pass", "opt1", "opt2", "term", "notes". **tagName** is the name of the tag that will be replaced with the value supplied for this **dbFieldName**.

Usage:

```
mymodule : $module->addServiceTag("user1", "domain");
```

addUserServiceHTMLField(type, friendlyName, dbFieldName, *regex, *errorMessage)

Returns: void

Description:

Creates an HTML field that will be used to add new services from the client interface. **type** must be of a given set of HTML field types. For a complete listing see Appendix A. If **dbFieldName** is anything other than "user1", "user2", "pass", "opt1", "opt2" then that field will not be stored in the database under the services table and may only be used in this module file. **regex** must be a regular expression compatible with the PHP defined function `ereg`. **errorMessage** is the message that is displayed if the input for this field does not match the regular expression defined by **regex**.

If the third parameter is anything other than user1, user2, pass, opt1, opt2, it must be formatted as an element of the detail array (ex. detail[emaiaddress]). Any field other than user1, user2, pass, opt1, or opt2 supplied for the third parameter will not be stored in the database, and may only be used in this module file.

Usage:

```
mymodule : $module->addUserServiceHTMLField("text", "Password", "pass", "^.{4,16}$", "Passwords must be between 4 and 16 characters.");
```

addUserServiceHTMLSelectOption(dbFieldName, optionName, optionValue)

Returns: void

Description:

Adds option fields to the HTML select field. **dbFieldName** must match a field that has been defined by the addUserServiceHTMLField function with **type** "select". **optionName** is the text that is displayed in the list for this select field. **optionValue** is the value returned when this option is selected.

Usage:

```
myModule : $module-> addUserServiceHTMLSelectOption ("opt1", "Active", "true");
```

createModule()

Returns: void

Description:

Adds the module to the database if it does not already exist.

Usage:

```
myModule : $module->createModule();
```

getDBFields()

Returns: Array

Description:

Returns an array containing all fields defined using the addDBField function.

Usage:

```
$dbFields = myModule : $module->getDBFields();
```

getModuleHTMLFields()

Returns: Array

Description:

Returns an array containing all fields defined using the addModuleHTMLField function.

Usage:

```
$moduleFields = myModule : $module->getModuleHTMLFields();
```

getModuleRowSelect()

Returns: Array

Description:

Returns an array containing the field defined to be the module row select field by the setModuleRowSelect function.

Usage:

```
$moduleRowSelect = mymodule : $module->getModuleRowSelect();
```

getPackageHTMLFields()

Returns: Array

Description:

Returns an array containing all fields defined using the addPackageHTMLField function.

Usage:

```
$packageFields = mymodule : $module->getPackageHTMLFields();
```

getRows(*conditions)

Returns: Array, FALSE if there are no rows to report

Description:

Returns all rows in this module that match the conditions option field and value pairs. **conditions** must be a two-dimensional array of the form array(array("field" => "", "value" => ""), ...). If **conditions** is not defined all rows will be returned.

Usage:

```
$conditions = array(array("field" => "id", "value" => "2"));  
$data = mymodule : $module->getRows($conditions);
```

getServiceHTMLFields()

Returns: Array

Description:

Returns an array containing all fields defined using the addServiceHTMLField function.

Usage:

```
$serviceFields = mymodule : $module->getServiceHTMLFields();
```

getServiceColumns()

Returns: Array

Description:
Returns an array containing all fields defined using the addServiceColumn function.

Usage:
`$serviceColumns = mymodule : $module->getServiceColumns();`

getServiceEditFields()

Returns: Array

Description:
Returns an array containing all fields defined using the addServiceEditField function.

Usage:
`$serviceEditFields = mymodule : $module->getServiceEditFields();`

getServiceTags()

Returns: Array

Description:
Returns an array of all tags defined using the addServiceTag function.

Usage:
`$serviceTags = mymodule : $module->getServiceTags();`

getTableName()

Returns: String

Description:
Returns the name of the table for this module. Note: The actual name for the table in the database begins with the prefix "m_". For example, the table name for the cpanel module returned from getTableName() is "cpanel", however the actual table name is "m_cpanel".

Usage:
`$tableName = mymodule : $module->getTableName();`

getTableSettings()

Returns: Array

Description:

Returns an array containing the table settings defined by the initialization of this module through the module constructor.

Usage:

```
$tableSettings = mymodule : $module->getTableSettings();
```

getUserServiceHTMLFields()

Returns: Array

Description:

Returns an array containing all fields defined using the addUserServiceHTMLField function.

Usage:

```
$userServiceFields = mymodule : $module->getUserServiceHTMLFields();
```

setModuleRowSelect(*friendlyName, *dbFieldName)

Returns: TRUE on success

Description:

Sets the friendly drop down for adding/editing services that allows the user to select a row from the module database. **friendlyName** is the name displayed when selecting to add a service using this module. **dbFieldName** must be a field defined by the addDBField function, and should have the unique identification property.

Usage:

```
mymodule : $module->setModuleRowSelect("Server", "hostname");
```

updateDBFields(name, value, id)

Returns: FALSE on failure, TRUE on success

Description:

Updates the field **name** with **value** for a row with an id of **id**.

Usage:

```
mymodule : $module->updateDBFields("cur", $data[0]['cur']+1, $serviceInfo['mid']);
```

Creating a Module

Module Requirements:

- Must be placed in the /inc/modules/ directory.
- Must have the name: modulename.class.php.
- Must be a PHP 5 class file.
- Must contain the member function getModule(). An example is below:

```
public function getModule() {  
    return cpanel::$module;  
}
```

- In order to create services remotely the public member function promodule(action, serviceInfo, packageInfo) must be defined. See /inc/modules/cpanel.class.php for an example on implementing the promodule member function.

Minimum Structure

For an example of a minimal module file see /inc/modules/none.class.php.

Appendix A

HTML Types: checkbox, password, radio, select, text, textarea