ConspireComponents

SDK Reference Manual

Welcome to ConspireComponents.
This document is designed to get you up to speed as quick as possible on how to use the components in development and product environments.

ConspireComponents is a PHP library and as such will require an environment capable of handling PHP scripts. Additionally the ionCube decoders (available free from http://www.ioncube.com/ if you have not already installed them), cURL and MCRYPT modules will need to be installed and configured for your PHP version.

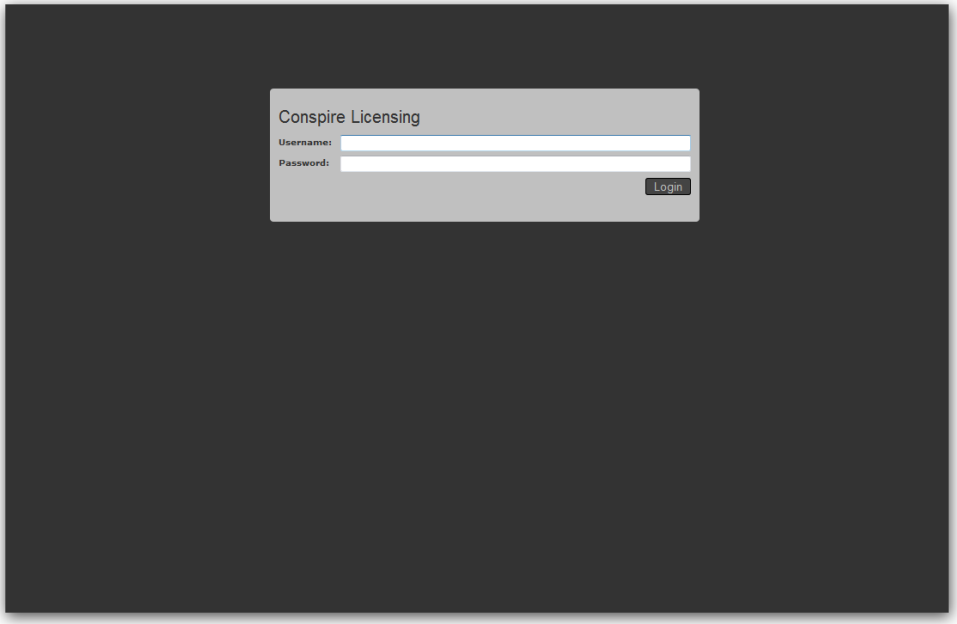
To begin using ConspireComponents, you'll need an active licensing manager account and a valid ConspireComponents license key. These details will have been emailed to you once they were activated and the email will contain information on how to access the license management control panel.

Contents

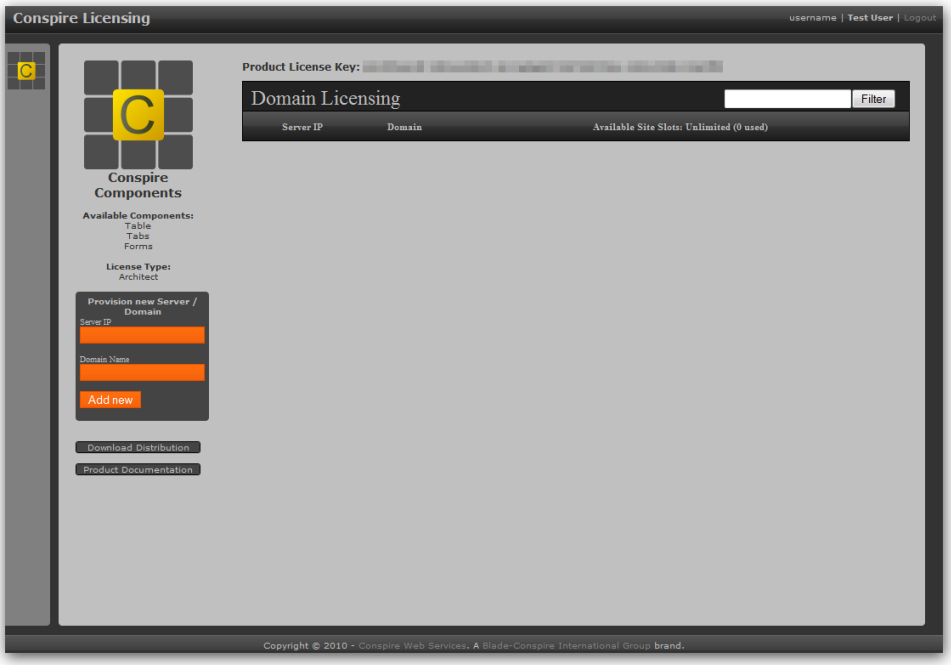
1.0 - Installation

A typical ConspireComponents installation can be built using the following method:

1. Log into your licensing management control panel using the details outlined in your activation email



2. Navigate to the ConspireComponents provision section by clicking on the ConspireComponents logo and click the 'Download Distribution' button and save the .zip file to your computer.

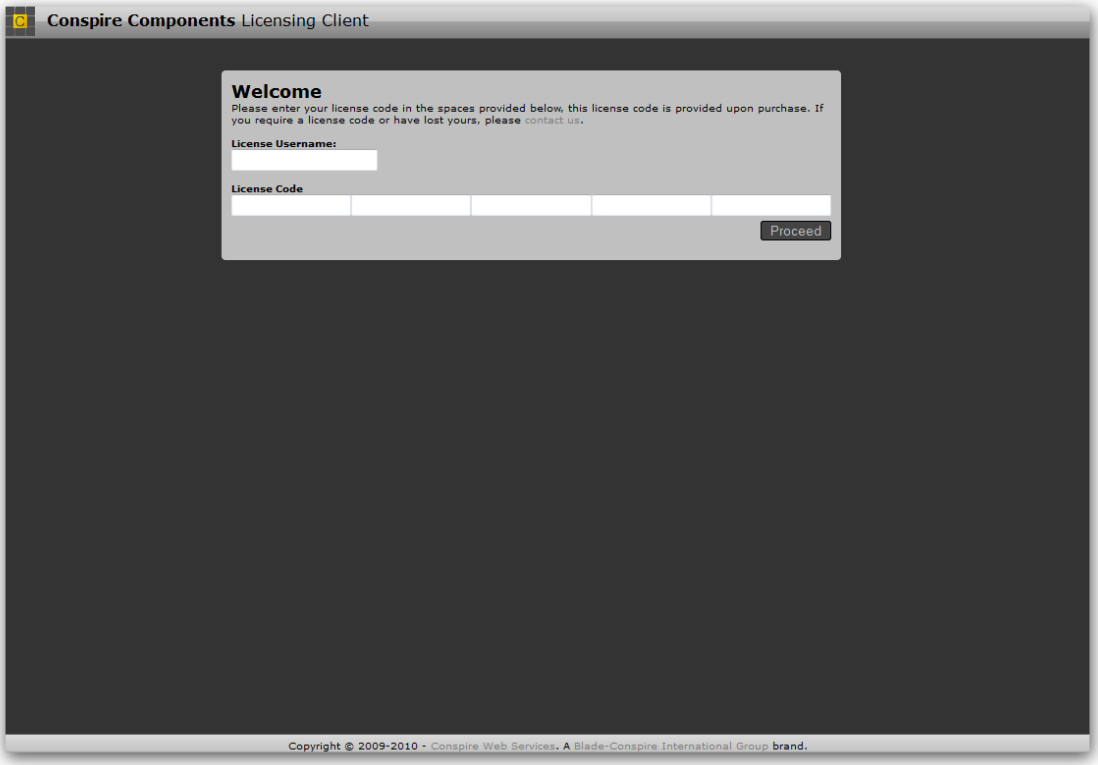


3. Unzip the distribution and upload the contents to your hosting environment using FTP.

4. While the upload completes, provision a new site by using the orange "provision new server / domain" box and ensure the site slot is active You will need to complete this step for every site the components will be installed on. For development and staging servers, you will need to add a site using each server IP address and host name - this process will use up a site slot against your license, however you may deactivate the development server once the site is placed in production. Additional Site Slots are available as an upgrade.

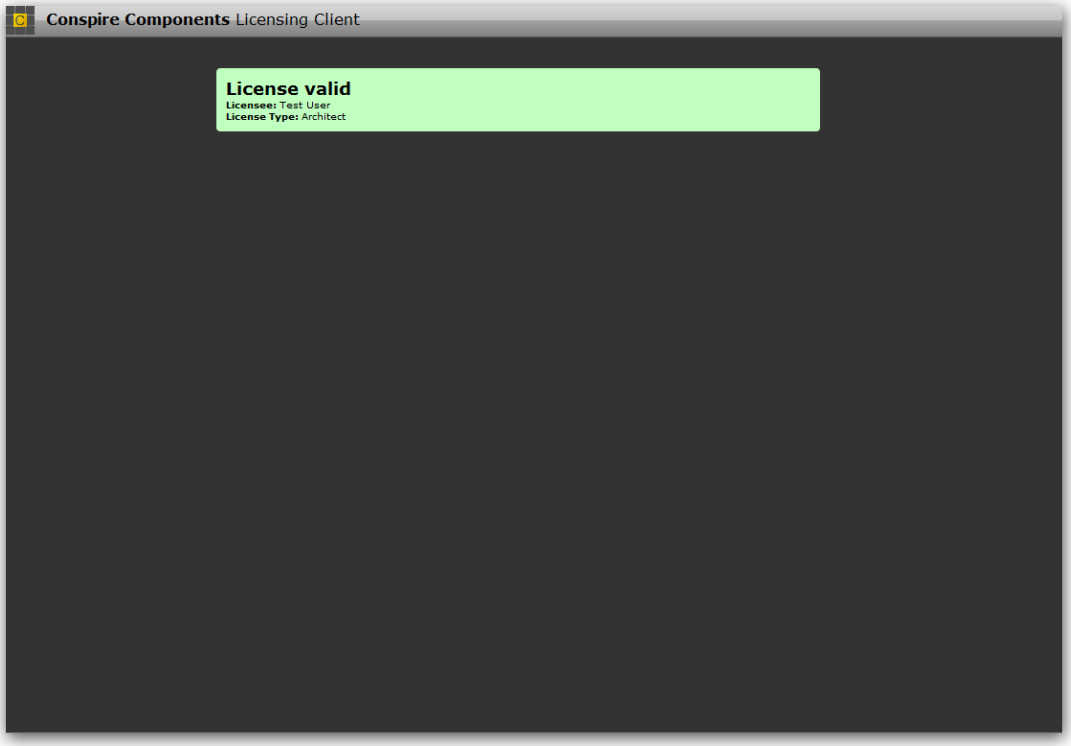
5. Once the files are uploaded to the server, use your web browser to navigate to: http://yourdomain/conspirecomponents/ or http://yourdomain/installationfolder/conspirecomponents (if the components were installed somewhere other than the base site folder)

6. Ensure PHP has write access and cURL has access to external servers and then enter your licensed username and the serial key for your ConspireComponents suite.



7. The installation will now contact the licensing server and create a license key file for the installation.

8. Once the license file is written, the licensing client will display the 'license valid' message and display your licensee information. The licensee name will be displayed in the HTML source output of the components. If for any reason, this process does not return a license valid message - please contact your account representative.


Conspire Components Licensing Client
License valid
Licensee: Test User
License Type: Architect

2.0 Basic Usage

Interfacing ConspireComponents is easy and all of the components follow a standard coding style for all core operations.

To get started with ConspireComponents, simply include the component you want to use in your PHP code.

```php
include 'conspirecomponents/conspire.table.php';
```

PHP Code 1: This example includes the table component

Once included, create an instance of the component of choice.

```php
$component = new conspire_table();
```

PHP Code 2: Create the component

If you have installed the components in a directory other than in the base working directory, the setWorkingDir() function can be used to change the working directory.

```php
$component->setWorkingDir('conspirecomponents');
```

PHP Code 3: Set the working directory

To access the components internal themes, simply call the loadBuiltInTheme() function with the name of the theme you would like to use. Supported themes and information on custom themes is available on the reference pages for each component.

```php
$component->loadBuiltInTheme('lime');
```

PHP Code 4: Load one of the built-in themes

Once you are ready to output to the browser: call the render() function.

The first parameter of the render() function on each component is a boolean which specifies whether the component should echo the rendered output directly to the browser or return the output for further processing. Direct output is the default behavior;

```php
$component->render();
```

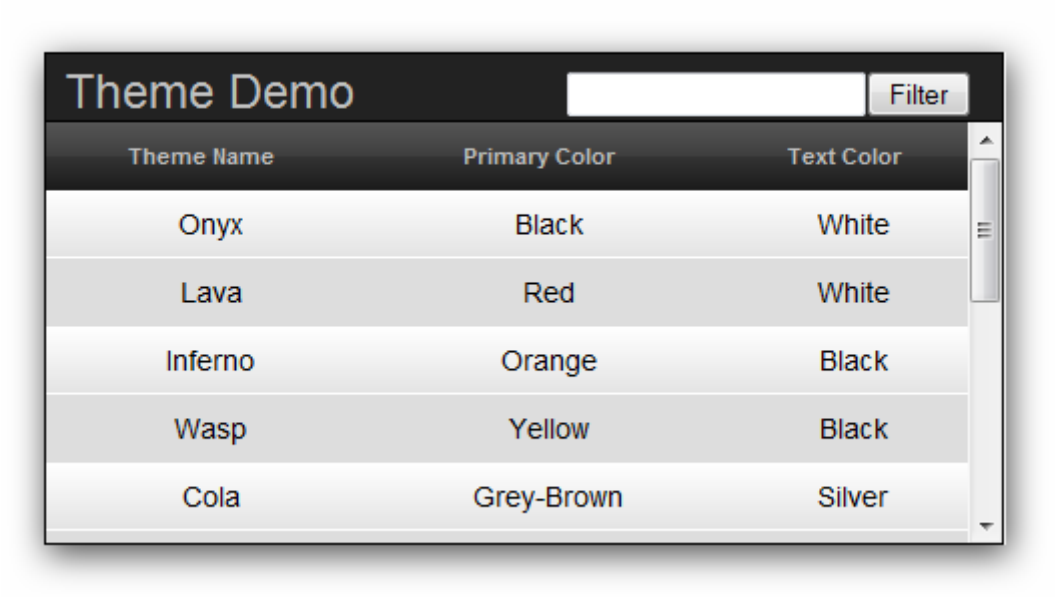
PHP Code 5a: echo directly to browser

```php
$output = $component->render(false);
```

PHP Code 5b: return output to variable

3.0 – Table Component

The table component allows you to easily display large amounts of data while keeping the code to a minimum.



In a typical usage scenario, the table component is constructed as an object then filled with data from a file, database, string or array. The output above is generated by a few basic commands.

```
include 'conspirecomponents/conspire.table.php';
$component = new conspire_table();
$component->loadBuiltInTheme('onyx');
$component->setTitle('Theme Demo');
$component->addHeaders('string', 'Name, Primary Color, Text');
$component->addDataSource('CSVFile', 'data.csv');
$component->render();
```

In this example, the table object is constructed. The default 'Onyx' theme is applied and the title is set by a string. The headers are then loaded as a comma separated string and the data source is loaded from a data.csv file before being rendered to the browser.

The output is rendered pre-styled and is ready to be sent to the users browser. The user can the search / filter the data by using the filter box and can sort the data by column (ascending and descending) by clicking the column header. Alternating rows are also styled differently to help make the data easier to read.

Each cell can be filled with a string value - and any valid HTML can be rendered provided all tags opened are closed before the end of the cell.

The table component supports all the standard built in theme styles and fully supports custom styling.

3.1 – Table – Code Reference

class_conspire_table();

function __construct();
Constructs the table component.

function setTitle(string $title);
Sets the title of the table component.
This is normally set to a short string title such as 'My Title' but can also be any valid HTML markup provided any tags opened in the markup are closed before the end of the string.

function setHeaders(string $type, string or array $input);
Sets the column headers. Rendering a table component without a call to setHeaders will produce a header-less output - suitable for data grids or where the columns are unknown before building the table.

> **$type can be set to either an array or a string depending on the format of $input**
> *Note: there is no default behavior for $type and an invalid input will not set the column headers in the output*
>
> **$input can be either a standard PHP array or a comma separated string.**
> Headers can be simple strings or valid HTML where any opened tags are closed.
> *Bear in mind however, the headers in the final output render are encapsulated in a standard 'a' (anchor) tag so they produce a link for the user to use to sort the table.*

Note: when using comma separated strings, there is no way to escape the ',' character. Should you require column headers with commas in them - please use an array.

function addDataRow(string $type, string or array $input);
Adds a single row to the output.

> **$type can be set to either an array or a string depending on the format of $input**
> *Note: there is no default behavior for $type and an invalid input will not add a row to the table.*
>
> **$input can be either a standard PHP array or a comma separated string.**
> Row cells can be simple strings or valid HTML where any opened tags are closed.

Note: when using comma separated strings, there is no way to escape the ',' character. Should you require rows with commas in them - please use an array.

function addDataSource(string $type, string or array $input);

Adds multiple rows to the output after it's data is retrieved from it's source location.

$type can be set to either an array or a string depending on the format of $input

Note: there is no default behavior for $type and an invalid input will not add a row to the table.

$input can be either a standard PHP array or a comma separated string.

Row cells can be simple strings or valid HTML where any opened tags are closed.

> **When adding an array as a data source**, each indexed object is treated as a new row.
> If that object is an array itself, the values of the sub-array is used for the row cells.
> *Note: input arrays will always be treated as multi-dimensional arrays.*
>
> **The input can also be a string** and can be used in the same way as addDataRow, except any new lines ("") will be treated as a new row.
> *Note: there is no way of escaping ',' - if you require data with commas in the text, please use another method.*
>
> **If $type is set to 'csvfile'** the component will attempt to load the path defind by $input and load it's data into the table component. Each new line will be treated as a new row and each comma (',') will be treated as a new column.
>
> **If the $type is set to 'mysql'** then $input can be set to a valid MySQL query - see registerDatabase(). If the query is valid the the data returned will be added to the table output.
>
> Each column in the MySQL table is treated as a column in the table and the number of queried columns should match the number of headers set using setHeaders()
> *Note: the query should be read-only in nature. Any deletion attempts will result in script termination. Additionally, if the database has not been set or cannot be connected to in registerDatabase(), the component will cease execution.*
>
> Finally, if the query results in an error in any way, the script will stop execution and output an error message.

function registerDatabase(string $type, string $host, string $user, string $pass, string $db);

Registers a database with the table component.
This is required before calling the addDataSource() function with MySQL as the type.
Note: $type can only be set to 'MySQL'.

function fullSort(number $col);

Sorts the internal arrays.
Useful for sorting the table before displaying the output to the user. This should be called after all data has been added to the table and before rendering the output.

> **$col is the left index of the column**
> (Where 0 equals furthest left)

function getThemes();

Returns an array of all the supported themes for the component.
The returned array is not indexed.

function loadBuiltInTheme(string $theme);

Loads and sets a theme from one of the built-in theme styles.

> **$theme names one of the built in themes**
> Call getThemes() to see a list of valid theme names

function setThemeInformation(string $element, string $class, string $style, boolean $inlineCSS);

Overrides or sets the theme style for the $element.

> ***$element can be one of the following:***
>
> **table:** applies to the outer container of the component.
> **tr:** applied to each row
> **tr_alt:** applied to each alternating row when alternating rows are to be rendered
> **td:** applied to each cell
> **tr_header:** each column header
> **a_sort:** applies to each column header sort link
> **title_bar:** the block container that holds the title and search containers
> **title_container:** the main title display area
> **search_container:** the filter input container
> **table_frame:** the container for the data row collection
>
> ***If $class is defined*** every instance of $element will be tagged with a class attribute in the TML markup with $class as it's value.
>
> ***If $style is defined and $inlineCSS is true*** every instance of $element will have the $style applied to it in a style attribute in the HTML markup.
>
> ***$inlineCSS*** determines whether the component should allow inilne CSS styling
> *Note: if set to false, only $class will be used when rendering the output.*

function render(boolean $echo, boolean $altRows, boolean $allowSort, boolean $allowSearch);

Prepares and sends the output to the browser.

The render function will output a comment explaining what the component is and does, copyright information and licensee information in the form of a 'Licensed by: {Company Name}' string. The company name is set in the licensing control panel. These comments cannot be removed or altered as per the license agreement.

Following these comments, the component outputs the HTML markup for the table.

If you have not supplied theme information or loaded a built-in theme, the component will simply output plain HTML without any style information included.

> ***$echo*** if set to false, the output will be returned instead of echoed directly to the browser This defaults to true.
>
> ***$altRows*** defines whether the component should use a different theme for alternating rows, or the same row style for each row. This defaults to true.
>
> ***$allowSort*** determines whether the user should be able to sort the table by clicking the headers. This defaults to true.
>
> ***$allowSearch*** determines wther the user should be allowed to filter / search the table. This defaults to true.

function setSortIconPath(string $direction, string $path);

Sets the path for the icon to be displayed when the user sorts by a column.

> ***$direction*** can be either 'up' or 'down' and determines which direction to apply $path to.
> *Note: there is no default behavior for $direction and any invalid input will be passed directly to the browser.*

function __destruct();

Destroys the table and removes any reference to itself in memory and the user session.
This function is normally called at the end of scope and should not need to be called otherwise

3.2 – Table Examples

This section contains examples on how specific functionality is handled using the table component.

Create a table from a CSV (Comma Seperated Values) file

```
include 'conspirecomponents/conspire.table.php';
$component = new conspire_table();
$component->loadBuiltInTheme('onyx');
$component->setHeaders('string', 'Column 1, Column 2, Column 3');

//where 'data.csv' is the path to your csv file.
$component->addDataSource('CSVFile', 'data.csv');
$component->render();
```

Note: The CSV should contain the same amount of columns as the setHeaders() call specifies, or the table may not render correctly.

Create a table from data retrieved from a MySQL database query

```
include 'conspirecomponents/conspire.table.php';
$component = new conspire_table();
$component->loadBuiltInTheme('onyx');
$component->setHeaders('string', 'Column 1, Column 2, Column 3');
/*
 Register the database for use with the table component.
 *MUST* be called before addDataSource();
*/
$component->registerDatabase('mysql', 'server', 'username', 'password', 'database');
$component->addDataSource('mysql', 'SELECT * FROM table');
$component->render();
```

Note: The result should contain the same amount of columns as the setHeaders() call specifies, or the table may not render correctly.

Add individual rows of data to a table component

```
include 'conspirecomponents/conspire.table.php';
$component = new conspire_table();
$component->loadBuiltInTheme('onyx');
$component->setHeaders('string', 'Column 1, Column 2, Column 3');
//add row from string
$component->addDataRow('string', 'Data 1, Data 2, Data 3');
//add row from array
$component->addDataRow('array', array( 'Data 1', 'Data 2', 'Data 3') );
$component->render();
```

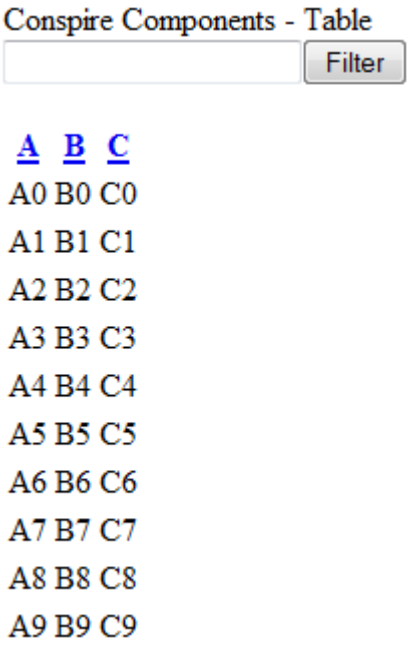
Hint: You can call addDataRow from a loop to add multiple values to the table before rendering.

Sort by a column before display

```
include 'conspirecomponents/conspire.table.php';
$component = new conspire_table();
$component->loadBuiltInTheme('onyx');
$component->setHeaders('string', 'Column 1, Column 2, Column 3');
$component->addDataRow('string', 'Born, Catch, Decides');
$component->addDataRow('string', 'Automatic, Besides, Community');
$component->addDataRow('string', 'Common, Declaration, Estatic');
$component->fullSort(0); //sorts by column 1 {Automatic, Born, Common}
$component->render();
```

Hint: fullSort() sorts the data in ascending fashion. To sort in a descending manner, call fullSort() twice successively.

3.3 – Customizing Table Appearance



Customizing the table component is generally done by calling the setThemeInformation() function. A list of elements which can be changed is included in the reference.

You can customize a table component by loading a built-in theme and then changing the parts you need to modify, or, by building a custom theme by successive calls to setThemeInformation. This example presumes the latter, but can be adapted to customizing a built-in theme.

Start by building a new conspire_table() object and fill it with the data.

The output without a theme is pretty basic, so we need to make a few adjustments.

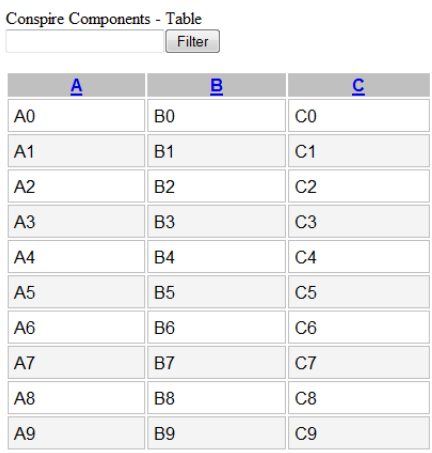
The setThemeInformation() function can apply a class name to the specified element, can apply an inline attribute, or a combination of the two.

If you HTML markup already has CSS defined for tables, you can set the $class parameter to match the stylesheet. If you CSS overrides whole element declarations, the, the table component will revert to these whenever a built-in theme hasn't bee loaded.

The class tag is appended before the inline style and anything declared in the $style parameter of setThemeInformation() will override the default styling for the page.

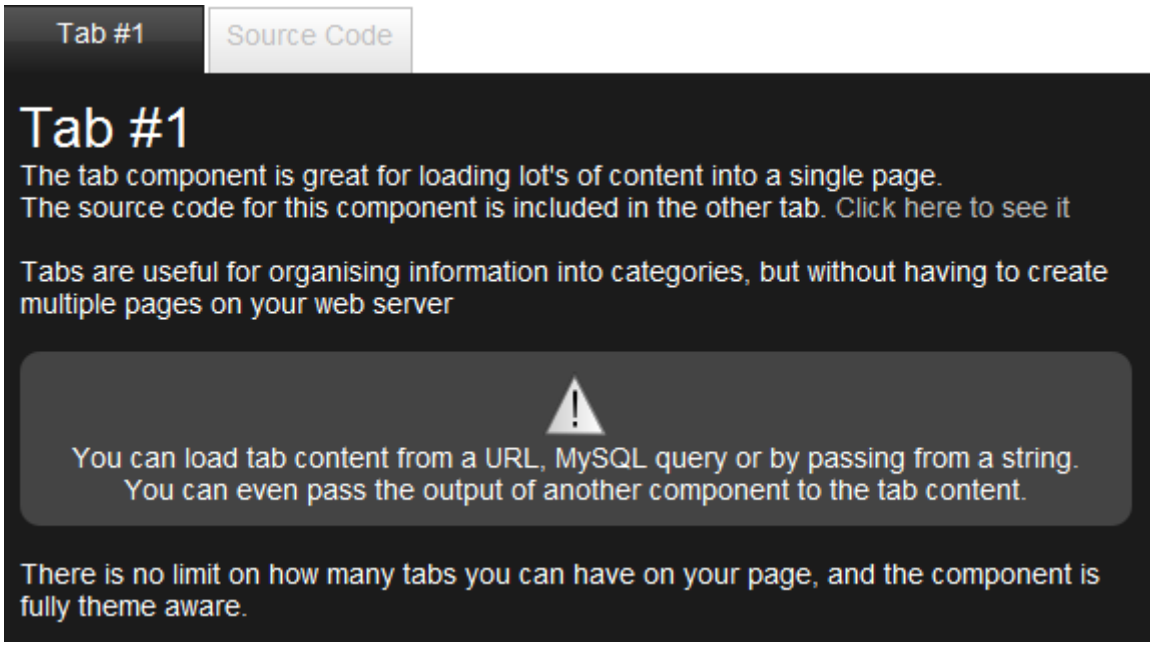
The following examples work by applying the inline style method, however, copying the style to a CSS style sheet referenced by the page and setting the $class parameter to match, will create the same result.

```
include 'conspirecomponents/conspire.table.php';
$component = new conspire_table();
$component->setThemeInformation('table', '', 'width:400px;font-family:arial;');
$component->setThemeInformation('table_frame', '', 'width:100%');
$component->setThemeInformation('tr_header', '', 'background:silver;font-weight:bold;width:100px');
$component->setThemeInformation('tr', '', 'width:100px');
$component->setThemeInformation('tr_alt', '', 'background:#f4f4f4;width:100px');
$component->setThemeInformation('td', '', 'border:1px solid silver;padding:5px');
$component->render();
```



4.0 – Tab Component

The tab component allows you to easily separate information to make it easier for your users to read.



In a typical usage scenario, the tabs component is constructed as an object and then tabs are added to the component. The output above is generated by a few basic commands.

```
include 'conspirecomponents/conspire.tabs.php';
$component = new conspire_tab();
$component->loadBuiltInTheme('onyx');
$component->setTitle('Theme Demo');
$component->addTab('Tab #1', 'URL', 'tab_contents_1.html');
$component->addTab('Tab #2', 'URL', 'tab_contents_2.html');
$component->render();
```

In this example the tabs object is constructed, the default 'Onyx' theme is then applied and two tabs are added to the output. The tabs in this example are created from URL sources. The component retrieves the output for the URL and displays it inside the tab.

The output is rendered (pre-styled) and is read for output to the user's browser. The user can then select which tab they want to view and the page is reloaded with the ?t_tab={tabnumber} value representing the selected tab.

Each tab can be filled with a string value and any valid HTML can be rendered provided all tags opened are closed before the end of the tab.

The tab component supports all standard built-in theme styles and fully supports custom styling.

4.1 – Tabs – Code Reference

class_conspire_tab();

function __construct();
Constructs the tabs component.

function setWorkingDir(string $dir);
Sets the working directory for the current component.
Note: the directory must exist and must be a valid ConspireComponents installation folder.

No directory checking is performed and incorrect usage of the setWorkingDir() function may result in the improper execution of the component.

function addTab(string $title, string $content, [link] $dblink);
Adds a tab to the component.

> ***$title*** sets the display title of the tab
>
> ***$type*** sets the content type of the tab
>
> Can be set to 'html', 'url', or 'database'.
>
> ***$content*** sets the tab content
>
> > **If $type is set to 'html'** the string can be a valid HTML string as long as any tags opened within the HTML content is closed before the end of the content string.
> >
> > **If $type is 'url'** the content string can be set to a relative or absolute URL.
> > The content is returned in the same manner as the web server displays it, so the content must be able to be viewed from a web browser.
> >
> > **If $type is 'database'** and the optional $dblink parameter is set the $content string should be a valid SQL query which will be executed on the $dblink provided.
> > The contents of the query will be used as the tab content

function addTabAsArray(array $input);
The same as addTab() but the parameters are set via an array which is then passed as $input.

function registerDatabase(string$type, string $host, string $user, string $pass, string $db);

Registers a database with the table component.

This is required before calling the addDataSource() function with MySQL as the type.

Note: $type can only be set to 'MySQL'.

function fullSort();

Sorts the internal arrays.

Useful for sorting the tab list before displaying the output to the user. This should be called after all data has been added to the table and before rendering the output.

function getThemes();

Returns an array of all the supported themes for the component.

The returned array is not indexed.

function loadBuiltInTheme(string $theme);

Loads and sets a theme from one of the built-in theme styles.

> **$theme names one of the built in themes**
> Call getThemes() to see a list of valid theme names

function setThemeInformation(string $element, string $class, string $style, boolean $inlineCSS);

Overrides or sets the theme style for the $element.

> ***$element can be one of the following:***
>
> tab: Applied to the tab selectors
> tab_active: Applied to the active tab
> tab_list: The container for the tab selectors.
> tab_frame: The outer container for the tab content
> pages: The block container that holds the page buttons and label
> page_label: The display text for the current page
> page_button: The style for the next page / previous page buttons.
>
> ***If $class*** is defined every instance of $element will be tagged with a class attribute in the TML markup with $class as it's value.
>
> ***If $style is defined and $inlineCSS is true*** every instance of $element will have the $style applied to it in a style attribute in the HTML markup.
>
> $tabPos sets which tab position to apply the theme for. Can be set to 'top', 'left' or 'right'

function render(boolean $echo, [string] $tabPosition, [boolean] $paging);

Prepares and sends the output to the browser.

The render function will output a comment explaining what the component is and does, copyright information and licensee information in the form of a 'Licensed by: {Company Name}' string. The company name is set in the licensing control panel. These comments cannot be removed or altered as per the license agreement.

Following these comments, the component outputs the HTML markup for the table.

If you have not supplied theme information or loaded a built-in theme, the component will simply output plain HTML without any style information included.

> ***$echo*** if set to false, the output will be returned instead of echoed directly to the browser
> This defaults to true.
>
> ***$tabPosition*** defines where the tabs should be rendered. This can be 'top', 'left', or 'right'
> The setThemeInformation will take the same parameter to apply custom themes.
> This defaults to true.
>
> ***$paging*** sets whether or not the component should render 'pages' of tabs when the amount of tabs rendered exceeds the width available to the component. This defaults to true.

function __destruct();

Destroys the table and removes any reference to itself in memory and the user session.

This function is normally called at the end of scope and should not need to be called otherwise.

4.2 - Tab Examples

This section contains examples on how specific functionality is handled using the tabs component.

Create tabs from URLs

```
include 'conspirecomponents/conspire.tabs.php';
$component = new conspire_tabs();
$component->addTab('Internal URL', 'URL', 'tabcontent.html');
$component->addTab('External URL', 'URL', 'http://www.conspireweb.com/');
/*
 External URLs may not work depending on your php.ini settings.
 Take care when using this feature, security may be a concern when loading pages from external servers.
*/
$component->render();
```

Create tabs from data retrieved from a MySQL database query

```
include 'conspirecomponents/conspire.tabs.php';
$component = new conspire_tabs();
/*
 Register the database for use with the table component.
 *MUST* be called before addDataSource();
*/
$dblink = $component->registerDatabase('mysql', 'server', 'username', 'password', 'database');
$component->addTab('DB 1', 'database', 'SELECT * FROM tabs WHERE id = "1"', $dblink);
$component->addTab('DB 2', 'database', 'SELECT * FROM tabs WHERE id = "2"', $dblink);
$component->render();
```

Create tabs from string or HTML content

```
include 'conspirecomponents/conspire.table.php';
$component = new conspire_table();
$component->addTab('Tab 1', 'string', 'Plaintext string');
$component->addTab('Tab 2', 'string', '<h1>HTML Content</h1>');
$component->render();
```

Change tab location

```
include 'conspirecomponents/conspire.table.php';
$component = new conspire_table();
$component->addTab('Tab 1', 'string', 'Plaintext string');
$component->addTab('Tab 2', 'string', '<h1>HTML Content</h1>');
$component->render(true, 'left');
```

Allow tab paging

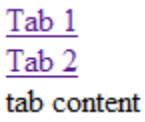
```
include 'conspirecomponents/conspire.table.php';
$component = new conspire_table();
$component->addTab('Tab 1', 'string', 'Plaintext string');
$component->addTab('Tab 2', 'string', '<h1>HTML Content</h1>');
$component->render(true, 'top', true);
```

4.3 - Customizing Tab Appearance

Customizing the tabs component is generally done by calling the setThemeInformation() function. A list of elements which can be changed is included in the reference.

You can customize a tab component by loading a built-in theme and then changing the parts you need to modify or by building a custom theme by successive calls to setThemeInformation. This example presumes the latter, but can be adapted to customizing a built-in theme. Unlike other components, the setThemeInformation() function on the tab component requires a fourth parameter ($tabPosition) to be set.

The output without a theme is pretty basic, so we need to make a few adjustments.



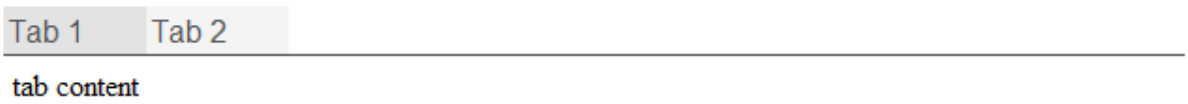
You can customize a table component by loading a built-in theme and then changing the parts you need to modify, or, by building a custom theme by successive calls to setThemeInformation. This example presumes the latter, but can be adapted to customizing a built-in theme.

The setThemeInformation() function can apply a class name to the specified element, can apply an inline attribute - or a combination of the two.

The class tag is appended before the inline style and anything declare in the $style parameter of setThemeInformation() will override the default styling for the page.

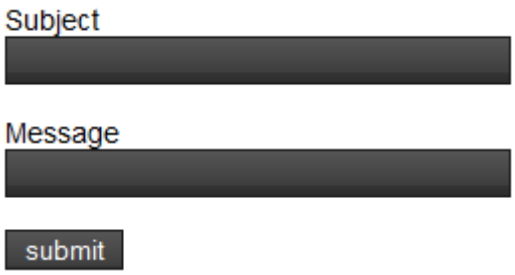
The following examples work by applying the inline style method - copying the style to a CSS stylesheet referenced by the page and setting the $class parameter to match will create the same result.

```
include 'conspirecomponents/conspire.tabs.php';
$component = new conspire_tabs();
$component->setThemeInformation('tab', '', 'width:70px;background-color:#f4f4f4;text-decoration:none;padding:3px;float:left;', 'top');
$component->setThemeInformation('tab_active', '', 'width:70px;background-color:#e3e3e3;text-decoration:none;padding:3px;float:left');
$component->setThemeInformation('tab_list', '', 'font-family:arial;text-decoration:none;color:#555;');
$component->setThemeInformation('tab_frame', '', 'clear:both;width:100%;border-top:1px solid #444;padding:5px');
$component->render();
```



5.0 - Form Component

The form component makes it easy to collect data from users by rendering HTML forms and handling the messy validation and processing side of things.



In a typical usage scenario, the form component is constructed as an object, to which fields are added and a data processing function is declared. The output above is generated by a few basic commands.

```
include 'conspirecomponents/conspire.form.php';
$component = new conspire_form();
$component->loadBuiltInTheme('onyx');
$component->addField('text', 'Subject');
$component->addField('text', 'Message');
$component->addSubmissionMethod('echo');
$component->render();
```

In this example, the form object is constructed, the default 'Onyx' theme is then applied and two fields are added to the output. The fields in this example are text type without validation parameters. The component renders the form and waits for $_POST variables that a passed to the same script that the object is run from.

When submission is detected, the form processes the command, in this case, by echo-ing the data back to the user.

The output is rendered (pre-styled) and ready to output directly to the user's browser.

A variety of field types and validation is possible and the submission methods can be anything from storing data in a database or sending an email to an administrator.

The form component supports all the standard built in theme styles and fully supports custom styling.

5.1 - Form Code Reference

class conspire_form();

function __construct();
Constructs the form component.

function setWorkingDir(string $dir);
Sets the working directory for the current component.
Note: the directory must exist and must be a valid ConspireComponents installation folder.
No directory checking is performed and incorrect usage of the setWorkingDir() function may result in the improper execution of the component.

function addField(string $type, string $name, string $default, string$validation, [variant] $params);
Adds a field to the form.

$type sets the type of data to collect

'text' displays a standard text input box
(most browsers will default to this when invalid types are passed).

'password' a masked input box suitable for passwords or sensitive information.

'checkbox' provides the user with a tick box.

'select' produces a drop-down box with preset items.
When using 'select' make sure the $params parameter is an associative array ('key'=>'value') filled with the items you would like your user to select from. The returned value will be 'key' whereas the user would see 'value'.

'file' a file upload.
NOTE: the component has no internal capabilities of handling file uploads, a custom submission method will need to be used if you need to add file upload fields.

$name sets the name of the field for use with $_POST data.
NOTE: the component replaces spaces (' ') with an underscore ('_') as spaces can be unpredictable when used in $_POST names.

$default

If $type is set to 'text' or 'textarea' the $default will be made visible inside the text box when the form is rendered to the user.

If $type is 'checkbox' the form will return the value you set here if the checkbox is ticked when submitted.

**If $type is 'select'** the component will look through the available items ($params) and look for a match. If a match is found, the value will be automatically selected for the user.

$validation determines the data validation method to use
If validation fails, an error will be returned to the user and an explanation on how to resolve the issue.

'none' means no data validation will be performed (default).

'number' tells the component that only numbers should be accepted on the field.

'alpha' tells the component to allow numbers and letters - but not symbols.

'email' checks the input has a domain and an @ symbol, however it does not check for user accuracy in anyway.

'notnull' accepts any input as long as it isn't empty.

'match' checks to to see whether the input of two fields match - pass the name of another field to $params to check for a match (not available on 'select' fields).

'custom' allows for custom validation of the data.
$params should be set to a string of the function name to call for checking.
The function should take one parameter containing the string value of the returned data. If the data is valid, the function should return a boolean 'true'.

$params A string or array depending on the other settings used.

function addFieldList(string $listType, string /or/ array $input);
Adds multiple fields to the component.

$listType can either be 'string' or 'array'

$input can be either a string or an array, depending on the $listType

If 'string' the component will treat every new line as a new field and each line should have 3 comma separated values. *NOTE: only $type, $name and $default can be set when using a string.*

If 'array' the component will treat $input as a multi-dimensional array containing field definitions.

function addSubmissionMethod(string $listType, string /or/ array $parameter);

Adds multiple fields to the component.

$type sets the type of submission method

'file' writes a CSV to the path specified in $parameter.

'database' adds the data to a database table. See generateDatabaseSettings.

'email' sends an email to the address supplied in $parameter.

'custom' will call a function named as a string in $parameter. The function should take no parameters and handle traversing the $_POST data itself.
The submission will fail if the function returns false and will continue to other submission methods or completion if the function returns true.

'echo' outputs the input as a pre-formatted print_r() to the browser (should only be used for debugging).

$parameter can be either a string or an array depending on the $type

If $type is 'file' then the $parameter should be the path to write a CSV file to.
If the path is not writable, the submission process will fail.

If $type is 'database' then the $parameter should be a database settings array returned by generateDatabaseSettings().

If $type is 'email' the $parameter should be a valid email address.

If $type is 'custom' the $parameter should be a string containing the name of a function to call.

function generateDatabaseSettings(string $type, string $host, string $user, string $pass, string $db, string $sql);

Similar to registerDatabase() on other components, but takes an additional $sql parameter.
The function returns an array containing the settings which can be passed to addSubmissionMethod.
The $sql parameter should be an 'INSERT' query which contains '' and '' which the component will replace with the appropriate data before running. mapFIeldToDatabase() should also be called.
Note: $type can only be set to 'MySQL'.

function mapFieldToDatabase(string $field, string $dbcolname);

Maps a field to the database when the database field name is different to the field name given to the component.

> ***$field*** should be equal to the name given to the field in the form component.
>
> ***$dbcolname*** is the name of the database column to place the data in.

function loadBuiltInTheme(string $theme);

Loads and sets a theme from one of the built-in theme styles.

> ***$theme*** names one of the built in themes
> Call getThemes() to see a list of valid theme names

function setThemeInformation(string $element, string $class, string $style);

Overrides or sets the theme style for the $element.

> ***$element*** can be one of the following:
>
> > **input:** Applied to each input element
>
> ***If $class is defined*** every instance of $element will be tagged with a class attribute in the TML markup with $class as it's value.
>
> ***If $style is defined*** and $inlineCSS is true every instance of $element will have the $style applied to it in a style attribute in the HTML markup.

function render(boolean $echo, [boolean] $reRenderOnFail, [boolean] $forceRedirect);

Prepares and sends the output to the browser.

The render function will output a comment explaining what the component is and does, copyright information and licensee information in the form of a 'Licensed by: {Company Name}' string. The company name is set in the licensing control panel. These comments cannot be removed or altered as per the license agreement.

Following these comments, the component outputs the HTML markup for the table.

If you have not supplied theme information or loaded a built-in theme, the component will simply output plain HTML without any style information included.

> ***$echo*** if set to false, the output will be returned instead of echoed directly to the browser. This defaults to true.
>
> ***$reRenderOnFail*** defines whether the component should re-render it's output should the data submission process fail.
>
> ***$forceRedirect*** sets whether or not the component should attempt to redirect the page to the URL specified in setContinueLink(). This will not work if the browser has been supplied with content before the component is rendered.

function setContinueLink(string $link);

Sets the URL of a page to continue to automatically on successful submission of the data, if $forceRedirect is set to true in the render() call.
This should be set BEFORE calling render()

function setSubmitLabel(string $label);

Sets the text to be displayed on the submit button.

function __destruct();

Destroys the component and removes any reference to itself in memory and the user session.
This function is normally called at the end of scope and should not need to be called otherwise.

5.2 - Form Examples

This section contains examples on how specific functionality is handled using the form component.

Add Fields

```
include 'conspirecomponents/conspire.form.php';
$component = new conspire_form();
$component->setIdentifier('form1');
$component->addField('text', 'field1');
$component->addField('password', 'field2');
$component->addField('checkbox', 'field3');
$component->addField('select', 'field4' array( 'value1'=>'Display 1', 'value2'=>'Display 2' ));
$component->addSubmissionMethod('echo');
$component->render();
```

5.2 - Customizing Form Appearance

Customizing the form component is generally done by calling the setThemeInformation() function. A list of elements which can be changed is included in the reference. You can customize a form component by loading a built-in theme and then changing the parts you need to modify - or by building a custom theme by successive calls to setThemeInformation(). This example presumes the latter, but can be adapted to customizing a built in theme.

The output without a theme is a plain HTML form, which in most projects is fine since the stylesheet will generally contain rules for forms. However, in some circumstances, you may wish to use the inbuilt theme functions.

The setThemeInformation() function can apply a class name to the specified element, can apply an inline attribute, or a combination of the two.

If your HTML markup already has CSS defined for forms, you can set the $class parameter to match the stylesheet. If your CSS overrides whole element declarations, the table component will revert to these whenever a built-in theme hasn't been loaded.

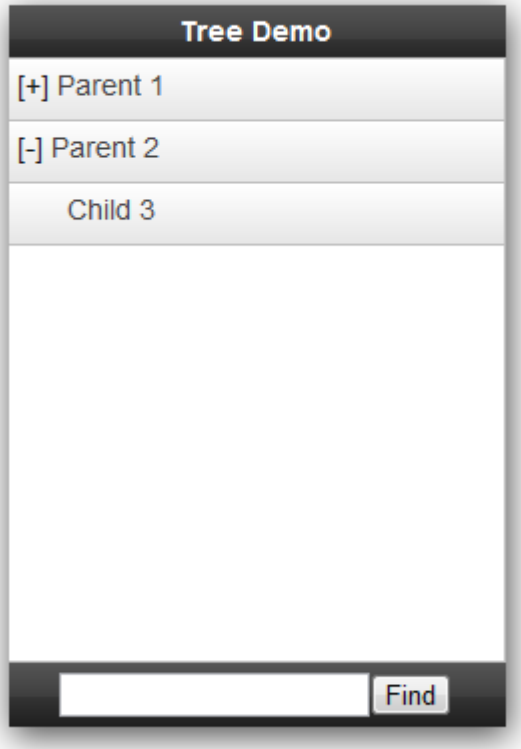
The class tag is appended before the inline style and anything declared in the $style parameter of setThemeInformation() will override the default styling for the page.

The following examples work by applying the inline style method, copying the style to a CSS stylesheet referenced by the page and setting the $class parameter to match will create the same result.

```
include 'conspirecomponents/conspire.form.php';
$component = new conspire_form();
$component->setThemeInformation('form', '', 'font-family:arial;width:187px;');
$component->setThemeInformation('field_label', '', 'border:1px solid silver;padding:3px;');
$component->setThemeInformation('input_text', '', 'border:1px solid silver;padding:3px;background-color:#f4f4f4%');
$component->setThemeInformation('textarea_label', '', 'display:inline;border:1px solid silver;padding:3px');
$component->setThemeInformation('textarea', '', 'border:1px solid silver;padding:3px;background-color:#f4f4f4');
$component->setThemeInformation('input_submit', '', 'float:right;border:1px solid silver;background-color:#f4f4f4');
$component->render();
```

6.0 - Tree Component

The tree component helps organize data into parent/child nodes that can be expanded and collapsed on demand.



In a typical usage scenario, the tree component is constructed as an object, nodes (called branches) are added to the object and the display is rendered. The output is generated by a few basic commands.

```
include 'conspirecomponents/conspire.tree.php';
$component = new conspire_tree();
$component->loadBuiltInTheme('onyx');
$component->addBranch('PNode1', 'Parent 1');
 $component->addBranch('CNode1', 'Child 1', 'PNode1');
  $component->addBranch('SNode1', 'Sub Child 1', 'CNode1');
  $component->addBranch('SNode2', 'Sub Child 2', 'CNode1');
 $component->addBranch('CNode2', 'Child 2', 'PNode1');
$component->addBranch('PNode2', 'Parent 2');
 $component->addBranch('CNode3', 'Child 3', 'PNode2');
$component->render();
```

In this example, the tree object is constructed, the default 'Onyx' theme is then applied and two parent nodes are added to the output. Child nodes are then appended to the parent nodes.

6.1 - Tree Code Reference

class conspire_tree();

function __construct();
Constructs the tree component.

function setWorkingDir(string $dir);
Sets the working directory for the current component.
Note: the directory must exist and must be a valid ConspireComponents installation folder.
No directory checking is performed and incorrect usage of the setWorkingDir() function may result in the improper execution of the component.

function setTitle(string $title);
Sets the title of the tree component.
This is normally set to a short string title such as 'My Title' but can also be any valid HTML markup provided any tags opened in the markup are closed before the end of the string.

function addBranch([variant] $id, string $text, string $parent);
Adds a node to the tree or a parent item.

> ***$id*** can be any unique identifier
>
> ***$text*** is the text display in the tree node.
> Text can be any valid HTML as long as any opened tags are closed before the end of the string, however size constraints may be a limiting factor when using the built-in themes.
>
> ***$parent*** is the identifier of the parent node this node should be added to.
>
> > **If blank**, a new top-level node will be added to the tree.
> >
> > **If set to the ID of another node** the new node will be added to the $parent and users can expand the parent item to reveal the new child node.

function loadBuiltInTheme(string $theme);
Loads and sets a theme from one of the built-in theme styles.

> ***$theme*** names one of the built in themes
> Call getThemes() to see a list of valid theme names

function setThemeInformation(string $element, string $class, string $style);

Overrides or sets the theme style for the $element.

$element can be one of the following:

tree_container: The outer wrapper.
title: The top title bar
search_container: The lower search box
node: Each node

If $class is defined every instance of $element will be tagged with a class attribute in the HTML markup with $class as it's value.

function render([boolean] $echo, [boolean] $disableSearch);

Prepares and sends the output to the browser.

The render function will output a comment explaining what the component is and does, copyright information and licensee information in the form of a 'Licensed by: {Company Name}' string. The company name is set in the licensing control panel. These comments cannot be removed or altered as per the license agreement. Following these comments, the component outputs the HTML markup for the table.

If you have not supplied theme information or loaded a built-in theme, the component will simply output plain HTML without any style information included.

$echo if set to false, the output will be returned instead of echoed directly to the browser This defaults to true.

$disableSearch excludes the lower search bar from the rendered output.

function __destruct();

Destroys the component and removes any reference to itself in memory and the user session.
This function is normally called at the end of scope and should not need to be called otherwise.

6.2 - Tree Examples

This section contains examples on how specific functionality is handled using the tree component.

Create a basic tree

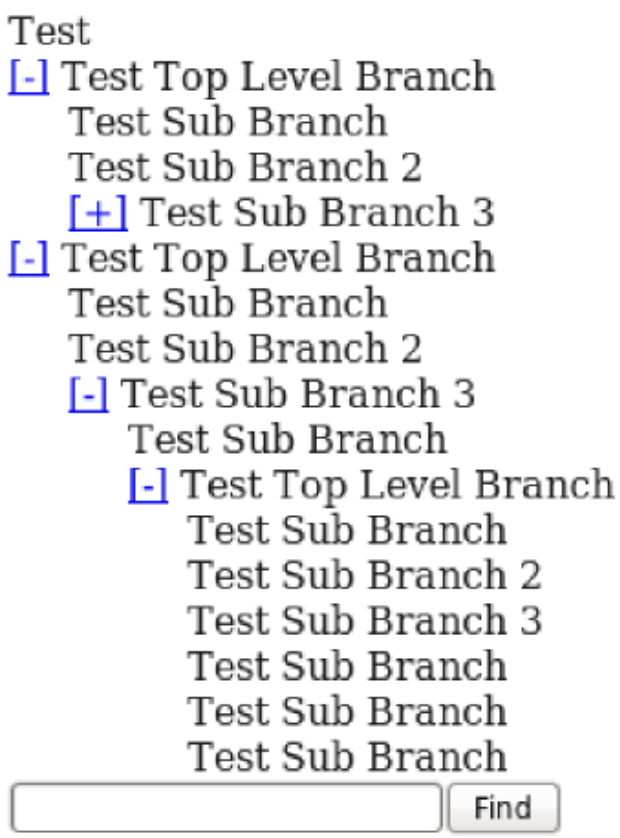
```
include 'conspirecomponents/conspire.form.php';
$component = new conspire_form();
$component->setThemeInformation('form', '', 'font-family:arial;width:187px;');
$component->setThemeInformation('field_label', '', 'border:1px solid silver;padding:3px;');
$component->setThemeInformation('input_text', '', 'border:1px solid silver;padding:3px;background-color:#f4f4f4%');
$component->setThemeInformation('textarea_label', '', 'display:inline;border:1px solid silver;padding:3px');
$component->setThemeInformation('textarea', '', 'border:1px solid silver;padding:3px;background-color:#f4f4f4');
$component->setThemeInformation('input_submit', '', 'float:right;border:1px solid silver;background-color:#f4f4f4');
$component->render();
```

Disable the search container

```
include 'conspirecomponents/conspire.form.php';
$component = new conspire_form();
$component->setThemeInformation('form', '', 'font-family:arial;width:187px;');
$component->setThemeInformation('field_label', '', 'border:1px solid silver;padding:3px;');
$component->setThemeInformation('input_text', '', 'border:1px solid silver;padding:3px;background-color:#f4f4f4%');
$component->setThemeInformation('textarea_label', '', 'display:inline;border:1px solid silver;padding:3px');
$component->setThemeInformation('textarea', '', 'border:1px solid silver;padding:3px;background-color:#f4f4f4');
$component->setThemeInformation('input_submit', '', 'float:right;border:1px solid silver;background-color:#f4f4f4');
$component->render(true, false);
```

6.3 - Customizing Tree Appearance

Customizing the tree component is generally done by calling the setThemeInformation() function. A list of elements which can be changed is included in the reference.



You can customize a tree component by loading a built-in theme and then changing the parts you need to modify - or by building a custom theme by successive calls to setThemeInformation. This example presumes the latter, but can be adapted to customizing a built-in theme.

Start by building a new conspire_tree() object and fill it with nodes. The output without a theme is pretty basic so we need to make a few adjustments.

The setThemeInformation() function can apply a class name to the specified element, can apply an inline attribute, or a combination of the two.

If your HTML markup already has CSS defined, you can set the $Class parameter to match the stylesheet. If your CSS overrides whole element declarations, the component will revert to these whenever a built-in theme hasn't been loaded.

The class tag is appended before the inline style, and anything declared in the $style parameter of setThemeInformation() will override the default styling for the page.

The following examples work by applying the inline style method - copying the style to a CSS stylesheet referenced by the page and setting the $Class parameter to match will create the same result.

```
include 'conspirecomponents/conspire.tree.php';
$component = new conspire_tree();
$component->setThemeInformation('tree_container', '', 'border:1px solid silver;height:250px;overflow:auto;');
$component->setThemeInformation('field_label', '', 'background-color:#e3e3e3;text-align:center');
$component->setThemeInformation('input_text', '', 'background-color:#e3e3e3;text-align:center');
$component->render();
```