StackApps User & Reference Guide

Version 1.3.10

Table Of Content

1.1 What is StackApps?

Tired of using those pre-canned, static Stacks? No clue how to write your own web apps within RapidWeaver? It's time to rethink what Stacks have been so far and could be: here is StackApps - the stacks collection that moves Stacks to the next level …

StackApps is a completly new set of Stacks that is not just adding static web clips or HTML snippets to your page, but is actually generating dynamic code. Using StackApps you are now able to create dynamic web apps without writing one single line of code in less than 5 minutes.

Key features

- No coding required
- Follows the CRUD pattern
- Role-based security framework
- 40+ powerful stacks
- Table-style, List-style & Details view
- Automatic form handling
- Input validation
- Flexible CMS component
- Supports template areas & ExtraContent enabled templates
- File upload component with database support
- Richtext Editor component
- Markdown support
- Google Web Fonts support
- Powerful format & layout settings
- Customizable error messages and message strings
- Generates compact PHP 5 compliant code during site publishing
- IDE-style UI
- Develop & test on local server, deploy on your remote server
- Comprehensive runtime errors
- … and it's free

1.2 How does it work?

StackApps assumes a certain app model which is based on the so-called CRUD pattern which stands for Create, Read, Update and Delete. CRUD pattern is the basis of a standard we are all familiar with: the HTTP protocol. Another example is the REST API pattern, which is the most used API standard on the internet these days

Even though those examples are widely used, the most common usage of CRUD is in the area of databases. SQL, the Standard Query Language used in databases like MySQL supports the CRUD paradigm as well: INSERT is the SQL equivalent of the Create, SELECT Reads from the database, UPDATE does the Update and guess what, DELETE helps you to Delete records from the database. Using these basic database operations one can implement various use cases. But do you want to program those operations by hand?

This is where StackApps kicks in. Imagine you want to write a simple database driven app that manages your personal tasks. What do you need to achieve this? Typically you want to display a list of all tasks (Read) and an individual task (Read), a form that allows you to create and modify tasks (Create, Update) and a way to delete unwanted tasks from the list (Delete). StackApps supports all of those tasks by providing a proprietary framework wrapped by easy to use Stacks that can be configured and customized to adjust to a variety of use cases.

So, can you use StackApps to implement app XYZ? Well, if you can map your apps functionality to the CRUD paradigm it is very likely that you can actually use StackApps to implement your specific app. But to be very clear here, StackApps is not the silver bullet that solves all kinds of requirements. But the good news is (and we will see that later) you can actually extend the StackApps framework by your custom written Stacks that can even enhance StackApps beyond what's being provided out-of-the-box.

⇤ **TOC**

1.3 About Records & Fields

Ok, let's dive a little deeper. I guess it's fair to assume you know what a relational database is and you have some experience with databases like MySQL, the most commonly used relational database that can be found on the web. A relational database like MySQL consists of tables of records and records are comprised of one or many database fields.

StackApps use the same metaphor to create database-driven apps within RapidWeaver: to display one or many records on a page, you typically take one of the Records Stacks within StackApps. A Records Stack itself contains again one or many Field Stacks which determine what will be displayed within a Record Stack. In our CRUD paradigm, Records & Fields cover the Read part.

Both, Records and Fields have a lot of configuration parameters that allow you to customize and tweak the behavior and look of the web pages that gets generated. In general, Records can be rendered in a table- or list-style layout in case of many records being displayed. If you want to display just one specific record, StackApps provides a Details layout.

Record Stacks included in StackApps

- *TableGrid* - a table-style record view that includes a pager and flexible sorter
- *Lister* - a list-style record view that includes a pager
- *Details* - a single-record viewer

Field Stacks included in StackApps

- *TextField* - a general purpose database field that includes a label
- *RichTextField* - a richtext database field renderer with lots of formatting options
- *DateField* - same as TextField but knows about tome & date formats
- *ImageField* - a image database field renderer

⇤ **TOC**

1.4 About Forms & Inputs

Ok, so far we can display individual database records, but how can we actually create, update or delete one of those records? This is where Forms & Inputs kick in. Forms are the StackApps equivalent to HTML forms including input validation and automatic POST handling after pushing the submit button.

StackApps includes a lot of smart form handling logic in order to make your life as easy as possible. That even includes the generation and handling of the forms submit buttons that usually require a lot of coding.

Form Stacks included in StackApps

- *Form* - a form-based record editor that includes input validators and POST handling
- *Search* - a form-based search field that can be used to trigger search queries
- *Login* - a form-based login that is part of StackApps security framework

Input Stacks included in StackApps

- *Input* - a general purpose form input that includes a label
- *DateInput* - same as Input but knows about tome & date formats
- *CheckRadio* - a checkbox/radio button style form input
- *Select* - a drop-down/selection list style form input
- *TextArea* - a large text entry which includes an optional richtext editor
- *Upload* - an upload form input
- *Hidden* - a hidden input used to set defaults and hidden fields

⇤ TOC

1.5 Installation

Before you can actually use StackApps Stacks you have to make sure that everything is setup and installed appropriately. If StackApps isn't installed the correctly, apps are most likely going to break.

Preparing your Mac

There are a couple of mandatory and some optional components you need in order to successfully run and use StackApps on your Mac

- RapidWeaver ≥5.2 - for obvious reasons (required)
- YourHead Stacks ≥2.0 - to actually run the Stacks (required)
- PHP >5.x - to run the generated apps on your web server(required)
- MySQL >5.x - to store your app data (required)
- Sequel Pro ≥0.9.9 - to manage your databases (optional)
- MacGDBp 1.5.0 - to debug your apps in case of any issues (optional)

Download the latest StackApps version

As soon as you have all of those packages installed on your development machine, you should download the latest version of StackApps before you start developing your first project. As there is no dedicated installer available, either double-click the StackApps Stack or just copy the StackApps Stacks to the Stacks folder on your Mac.

Install StackApps using the Stacks installer

1. Simply double-click the StackApps Stacks

2. Restart RapidWeaver and the new Stacks should show up in the toolbar

In case the built-in Stack Installer doesn't do its job for whatever reason you can always install StackApps manually.

Manually copy StackApps to your Mac

1. Open the Finder and copy all StackApps Stacks into the clipboard with cmd-C
2. Navigate to ~/Library/Application Support/RapidWeaver/Stacks inside the Finder and paste the Stacks with ⌘-V into this folder
3. Restart RapidWeaver and the new Stacks should show up in the toolbar

Setting up RapidWeaver

There are a few setting within RapidWeaver you should keep in mind before publishing your project to the dev or production server:

- Use page-relative file links within the project setup
- if using navbar security within your project, don't use consolidated CSS files within the RW preferences
- Cruft-less links are a nice way of making URLs within your project look nicer

Preparing your hosting environment

Now that you are ready on the development side, you should check your hosting environment which will ultimately run the apps you have created with StackApps. It is highly recommended that you use the same or later releases of PHP and MySQL on your server compared to what's installed on your Mac. This avoids any possible incompatibilities between your development and your hosting environment. Nevertheless, there is no guarantee that a web app that has been generated on your Mac is running smoothly on your hosting environment.

2. Your First StackApps Project

Without further ado, let's step right into StackApps and create our first little project that demonstrates the basic capabilities of the StackApps framework. The project I have choosen for the start is neither too complex nor too simple. But there even more examples available that even show the more advanced features of StackApps. Ok, let's start …

2.1 My Tasks - A ToDo List Manager

As our first project I have decided to develop a simple todo list manager, that has the minimum set of features to create, read, update and delete tasks or CRUD as mentioned before :) My Tasks should have a list view that shows all active tasks or todos, a details view that shows all the task details and finally a task editor that is used to creat a new task, update existing tasks and finally delete unwanted tasks.

In order to get started, let's setup the database scheme for our little project.

2.2 The Database

Ok, here is a possible schema that contains all the fields that are needed to create our simple task manager

```
CREATE TABLE 'mytasks' (
  'task_id' int(11) NOT NULL AUTO_INCREMENT,     // this is our unique primary key to index our tas
ks
  'task_prio' int(11) NOT NULL,                  // every task needs a prio, right
  'task_desc' varchar(256) NOT NULL,             // and a description of course
  'task_done' int(11) DEFAULT '0',               // we need to track progress after all
  PRIMARY KEY ('task_id')
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=latin1;
```

Let's add some content to have something we can look at as we do the entry part in the end.

```
INSERT INTO 'mytasks' ('task_id', 'task_prio', 'task_desc', 'task_done')
VALUES
    (1,3,'Finish website',10),
    (2,2,'Finish StackApps',100),
    (3,1,'Create StackApps docs',20);
```

Using Sequel Pro or your preferred DB admin tool, you should be able to setup the database and enter the test content. Now we are all set to actually start using StackApps …

2.3 The Task List

TBD

2.4 The Task Details

TBD

2.5 The Task Entry

TBD

2.6 Other Features

TBD

⇤ **TOC**

3. Examples

In the future StackApps will come bundled with a couple of example apps that can be used to learn how to use StackApps and learn about the more sophisticated features of the product. Currently, these examples are not yet available.

MyTasks

MyTasks is a very basic StackApps project to manage simple tasks that contains all the ingredients of a minimal CRUD based app:

- an Overview Page on a TableGrid using different WHERE clauses, TextField Stacks and JOIN operations
- an Edit Page based on a Form and Input Stacks

Inventory

Inventory adds a couple of new features to what we've learned via the MyTask app:

- An image field that can display images from the server
- An Upload feature to store new images on the server
- A Details Page based on Details and Field Stacks

CMS

The CMS project covers the content management capabilities of StackApps. Using various CMS Stacks you can display database content on your site either in the regular content area or in a arbitrary ExtraContent area. The project also includes a content editor that can be linked to the actual content.

Blog

This is the most complex app example that creates a basic blog that uses the themes blog styles that are usually bundled with RapidWeaver themes. The blog features:

- A blog style overview page using the Lister Stack
- A custom script to calculate the number of comments for a blog post
- A blog post page that contains a comment thread

⇤ TOC

4. Tips & Tricks

Create Static Text Links with a Dynamic URL

Imagine you want to create an Edit button in a TableGrid that links to an editor for the specifc record. In order to achieve this, use a standard TextField Stack, set the Static Text to ‘Edit’, don’t enter a Label or Field and create a Static Action Page to point to the editor page. Use the record unique ID as a Action Param for the editor.

Creating a Logout Link

In case you want to provide a logout link for your users, create a normal textlink and point it to the login page adding the formAction=logout parameter, i.e. yourdomain.com/login/?formAction=logout

Clicking this link deletes the security context and send the user to the cancel page defined in your login Stack. You can of course combine this with the various button stacks that are available out there.

Creating a Login/Logout link with the user name being displayed

Imagine a link that either shows Login if you are not yet logged-in or your user name and a Logout link in case you are logged-in. In order to create such a element, use a Script Stack with the following code and Trigger being set to ‘Here’:

```
// retrieve $setup from global scope
$setup = $GLOBALS['setup'];

if ($setup->isLoggedIn())
{
    $link = new HTMLAnchor("Logout", array("href" => $setup->_config["secLogin"] . "?formAction=lo
gout"));
    out("Hi, " . $_SESSION["user"] . " | " . $link->toHTML());
}
else
{
    $link = new HTMLAnchor("Login", array("href" => $setup->_config["secLogin"]));
    out($link->toHTML());
}
```

Alternatively you can do something similar on the navbar by changing the Login menu depending on the login state.

```
<?php showLogin("Login", "Hi, %s"); ?>
```

Add the above code as the Login page name in the RW HUDand you will get a page named Login as long as you are not logged in and Hi, username in case you are logged in. The function takes two strings as input. If you leave them out, StackApps is using “Login/username” as a default.

A Security-enabled Navbar

One of the key feature of StackApps is its security framework that allows you to password protect pages or even smaller fragments of your website. So far it was not possible to actually extend the security to your website navbar, which so far showed menu items even though the user isn’t able to actually open the page due to its access rights.

Since StackApps 1.2.0 this has changed. You can now security enable your navbar by 3 simple modifications/prerequisits:

1. Modify your themes index.html
2. Change visibility of individual menu items by adding a security descriptor to the page name
3. Finally, make sure you use a Setup Stack on every single page no matter if it is dynamic or not

Those two modifications are relatively easy to accomplish. Let’s start with the template modification.

Modify the Templates index.html

Just go to the theme selector in RapidWeaver and search for your current theme. Right-click on the theme and either create a dublicate first or directly show the theme content inside the Finder. Either way, as soon as you see the content of the theme package just look for the index.html and open it with your preferred text editor.

Next thing you do is search for the %toolbar% section inside the HTML code. This is where we need to apply our modification. Simply replace %toolbar% with the following code section and don’t change a single piece.

```
<!-- StackApps Navbar Security Before Toolbar
==============================================
<?php ob_start(); ?>%pathto(styles.css)%<?php $path = pathinfo(ob_get_clean()); $path = $path['dir
name'];
if (substr($path, 0, 1) != '/' && substr($path, 0, 4) != 'http') {
include_once($path.'/../../plugins/stacks/libs_setup/simple_html_dom.php');
include_once($path.'/../../plugins/stacks/libs_setup/helper.php'); ob_start();?> -->

%toolbar%

<!-- StackApps Navbar Security After Toolbar
=============================================
<?php if (version_compare(PHP_VERSION, '5.3.1', '>='))
{checkNavbarSecurity(ob_get_clean());} else {ob_end_flush();}} ?> -->
```

As this code is wrapped in HTML comments, it doesn’t do anything until your page is PHP enabled, either by adding the StackApps Setup Stack or changing the page extension to .PHP manually.

But even in this case, the menu will not change as long as we do not modify the menu structure itself inside RapidWeaver. Ok then, let’s move to the second modification step. But first make sure you save the file. Another aspect you should be aware of, this feature requires PHP >= 5.3.1 in order to work. There is some code embedded that checks the used PHP version in order to handle older environments, but keep in mind that some of the menus may become visible in case you run on an environment that doesn’t have the required PHP version.

Visibility of Individual Menu Items

With the modification on the index.html we have all the plumming ready for us to control the visibility of individual menu items or whole menu branches. In order to do that go to RapidWeaver and select a page you want to protect in the navbar. Click on the page name in the page hierarchy and append the following security descriptor to the page name The used values in the descriptor reflect the ones used for page protection. What the security descriptor does is, it makes the menu item visible only for users that own the demo role. Simple.

The security descriptor supports the following security types:

- *authenticate* you have to be authenticated in order to see the item, e.g. <sec type=“authenticate”/>
- *logout* you have to be logged out in order to see the item, e.g. <sec type=“logout”/>
- *accesslevel* you have to have an access level >= the one define here to see the item, e.g. <sec type=“accesslevel” value=“10”/>
- *roles* you have to have one of the roles defined here to see the item, e.g. <sec type=“roles” value=“demo”/>

As the page title automatically goes to the browser title, you will see the security descriptor of the page in the browser title. In order to prevent this, choose a different browser title within RapidWeaver.

Assuming you have done the necessary markup and you have enabled the security framework inside StackApps, you will get a security-enabled navbar after you have done those changes. But please keep in mind that those changes only impact the menu visibility and not the page protection. You can sill open those pages if you know the URL. So make sure you have those pages protected as well using the Security Stack.

Even though this is a very elegant and non-disruptive way of getting this feature to work, the security-enabled navbar is still limited in the way StackApps handles the “remember login” feature. Due to the fact that Stacks can not add code in front of everything (in this case the navbar) we can not detect an already logged in user before the navbar is rendered. This is only the case when accessing the page after a session timeout and the security context need to be established again. This might be solved in the future as soon as Stacks is going to support arbitrary content to go in front of the page.

Creating Dynamic Content Pages

Using a CMS Stack you can not only draw content on page from the database, you can even render pages completely dynamic without the need of creating a RapidWeaver page for every page you need.

In order to achieve this, create a RapidWeaver page first and call this page ‘showContent’ for instance. Place a CMS Stack on the page and select a dynamic Content ID. The name you specify as a Content ID is now the URl parameter used to retrieve the actual content name, which can now be specified as part of the URL of the page. By calling the page with the URL parameter and the respective content ID you actually display the content with this ID on the page.

Example: yourdomain.com/showPage/?id=start will display the content with ID start on the page

Hiding any page content during form processing

The way StackApps process forms leads to a refresh of the page that actually includes the form. By pressing the submit button, the form page gets called once again and the input of the form gets processed. The StackApps takes care that the form itself will not be displayed the second time as the form knows in what state it is, but any additional text and stuff will be rerendered during the processing phase.

To avoid this behaviour, all non form content should be placed after the Setup Stack which should actually be on the top of the page. Doing so avoids any rerender of the content.

⇤ TOC

5. Support

In case of any question, please contact me either via Twitter or mail. Keep in mind that there is no guaranteed response time as this is after all a spare time project for me. Nevertheless, I am more than happy to help if possible.

5.1 Contacts

@gostackapps
support@stackapps.info

5.2 Release Notes

Release Notes:

- 1.3.10 Bug fix release (6/23/2013)

  Bug Fixes

  - Captcha was generating a strict code validation error (FIXED)
  - mime type validation for uploads not working (FIXED)
  - Support for max items in ase of no pager for Lister and TableGrid (CHANGED)
  - Fields that contained List() in one way or the other is considered to be a multi-value (FIXED)
  - Paging is still used in Lister Stack even if turned off (FIXED)
  - Return false instead of an error in CheckSecurity in case of an empty access list due to a deleted user (FIXED)
  - Handling of HTTPS has been improved in rel2abs (FIXED)
  - Code that handles Google Analytics string shorter than 2 to switch off code generation (CHANGED)

  Known Issues

  - StackApps does not support DB Views
  - Uploaded files are not shown after negative input validation
  - c5 Filemanager might not work in all environments / setups yet

- 1.3.9 Bug fix release (5/12/2013)

  Bug Fixes

  - Default setting for CheckRadio Stack wasn’t considering radio buttons (FIXED)
  - transferParams did not consider wildcard params correctly (FIXED)

- 1.3.8 Feature & Bug fix release (5/5/2013)

  New Features

  - Support all Status Board settings within the Report Stack (NEW)
  - &diagram=bar|line supported for Status Board Graphs. (NEW)
  - TabeGrid supports now pure table generation without other page content (NEW)
  - Generate table header only if there is a table header available (FIX)

  Bug Fixes

  - Select inputs with multiple selections didn’t work after refactoring (FIXED)

- 1.3.7 Feature & Bug fix release (4/28/2013)

  New Features

  - New Report format that supports Panic’s Status Board iPad app (NEW)
  - Extended event handling to Lister and Interator Stack. (CHANGED)

  Bug Fixes

  - Handle ports in URL scheme correctly in case rel2abs (FIXED)
  - Date fields generated –1 length text fields which IE can not handle (FIXED)

- 1.3.6 Bug fix release (3/4/2013)

  Bug Fixes

  - Introduced a regression within the Richtext Stack causing the template language to break (FIXED)

- 1.3.5 Bug fix release (3/3/2013)

  New Features

  - Added trim function to where clause parameters to better support fuzzy searches, etc. (CHANGED)
  - Tab Index added to input Stacks e.g. Input, Checkbox, Textarea, Select, Captcha, Button Set (NEW)

  Bug Fixes

  - Report Stack was generating errors within CSV (FIXED)
  - Paging in Lister Stack has been broken since 1.2 It is fixed now :) (FIXED)
  - transferParams didn’t work in Lister and TableGrid due to a refactoring I did in the past (FIXED)

- 1.3.4 Feature & Bug fix release (2/10/2013)

  New Features

  - Send Verification now support the same parameter structure as Send Mail (CHANGED)
  - Send Mail now supports emails to a fixed address and variable link param values (NEW)
  - Send Activation is now called Send Mail (CHANGED)
  - TextArea Stack now supports a default filemanager based on c5 Filemanager (NEW)
  - Thanks to Modernizer DatePicker support is now only usable for non-mobile devices as Mobile WebKit supports a native date picker (CHANGED)
  - Support for new HTML5 compliant input types. This will replace the DateInput which is now part of the Input Stack. Date Input will become deprecated and will be removed soon (NEW, CHANGED)
  - Added a new server-side image scaler with additional processing features (NEW)
  - Added back reference feature to actions which allows for a back jump to the page after form processing (NEW)
  - Floating control enabled for all Fields and Inputs (CHANGED)

  Bug Fixes

  - All jQuery libs used within StackApps are currently fixed to 1.8.3 to avoid 1.9 incompatibilities (FIXED)
  - Fixed logout issue that causes the page not to be redirected appropriately (FIXED)
  - RepeatedInput fields don’t require a unique field name. This is handled by StackApps (FIXED)
  - Target to a variable didn’t work because of a wrong sanity check (FIXED)
  - SendVerification Stack had some missing styles (FIXED)
  - Cleaned up handling of empty joins (FIXED)
  - Exception and error handling for connections was broken since 1.3.2 (FIXED)

- 1.3.3 Feature & Bug fix release (12/27/2012)

  New Features

  - Added Debug setting to scripts (NEW)
  - Added new Script events for Newsletter Send, Display Row, Logout (CHANGED)
  - New NL to
    convert option in Fields (NEW)
  - Added encoding for message strings in Setup Stack (CHANGED)
  - TextArea size has been changed from cols & rows to width & height in px (CHANGED)

  Bug Fixes

  - Various template stacks have been removed temporarily. Will be added later (FIXED)
  - Global Setup Stack was missing some properties (FIXED)
  - Reverted back from the new DOC_ROOT address scheme to page relative address scheme. Sorry about that! (FIXED)

- 1.3.2 Feature & Bug fix release (12/25/2012)

  New Features

  - Preliminary SSL support using the Security Stack (NEW)
  - StackApps now supports the different RW link themes (page relative, doc root relative & site address relative). Not all site themes do though :((NEW)
  - Upload supports image dimension validation (NEW)

  Bug Fixes

  - Code generation has been greatly improved. We are generating much nicer code now :) (FIXED)
  - Email setting removed from Activate, Reset and OptOut Stack. This is no longer needed since 1.3.0 (FIXED)
  - Email was still required in OptOut links, but not send by Send Verification Stack (FIXED)
  - Label handling during validation of Hidden fields fixed (FIXED)

  Known Issues

  - Mysterious image path issue
  - StackApps does not support DB Views
  - Uploaded files are not shown after negative input validation

- 1.3.1 Bug fix release (12/18/2012)

  New Features

  - Sorry, no new stuff :)

  Bug Fixes

  - Image path handling was broken. Not sure what I was thinking … (FIXED)
  - Report custom filename wasn’t working (FIXED)
  - Fixed DatePicker code (FIXED)
  - Adjusted settings for Colorbox (FIXED)

  Known Issues

  - StackApps does not support DB Views
  - Uploaded files are not shown after negative input validation

- 1.3.0 Feature release (12/16/2012)

  New Features

  - Colorbox lightbox support added to Setuo Stack. Just add Class parameter to links to launch the lightbox (NEW)
  - Slimbox2 lightbox support added to Setup Stack. Just add REL parameter to links to launch the lighbox. (NEW)
  - New dynamic TITLE Attr added for action links (NEW)
  - Concatination of fields thru transfer variable possibel to support multiple joins (NEW)
  - Extensive encoding of parameter inputs (NEW)
  - jQuery import has been removed from StackApps (NEW)
  - DatePicker date format has been added (NEW)
  - New Report Stack that can be used to export a table content (NEW)
  - Image Stack now supports a maximum number of rendered images (NEW)
  - Image Stack now supports multivalue fields generated by the Upload Stack (NEW)
  - Upload Stack now supports multiple uploads stored as multivalues (NEW)
  - Form Stacks now support a optional cancel action (NEW)
  - Extended pattern language available for Richtext Fields (NEW)
  - Transfer params setting added to Search Stack (NEW)
  - Extended UI for fields to display action parameters and output variables (NEW)
  - New UI to display actions in Forms and Richtext fields (NEW)
  - New output feature for fields to store a field value in a variable for later use in scripts or Field Stacks (NEW)
  - Visibility setting added to inputs as well to avoid If-Then-Else Stacks (NEW)
  - Border color added to Lister Stack (NEW)

  Bug Fixes

  - Image size detection now possible without GD installed (FIXED)
  - Hauler Stack ExtraContent target has been fixed (FIXED)
  - CMS bug fixed where runtime error is shown in case of the editor button not being used (FIXED)
  - Repeated Input Stack fixed: missing Stack ID input (FIXED)
  - Reworked HUD UI for Fields (FIXED)
  - Transfer params are now separated with & instead of , to make this consistent with URL syntax (FIXED)
  - Transfer parameters fixed for actions without a URL parameter (FIXED)
  - Unique hash generation was broken (FIXED)
  - The email is no longer part of verification links and therefor doesn’t need to be transfered anymore (FIXED)
  - Sanatize default parameter values (FIXED)

  Known Issues

  - StackApps does not support DB Views
  - Uploaded files are not shown after negative input validation

- 1.2.1 Bug fix release (11/07/2012)

  Bug Fixes

  - Handling of null values for Hidden fields had a side effect impacting Select Stacks

- 1.2.0 Feature Release (11/06/2012)

  New Features

  - The DatePicker finally works :) (NEW)
  - Security enabled navbar: enable/disable menu items based on users access rights (requires PHP >= 5.3.1) (NEW)
  - Global Less support within Setup Stack (NEW)
  - Fields can now easily be floated directly or using the Floater Stack for more complex scenarios (NEW)
  - New session variable ‘userid’ has been added which provides the index of the respective user record at runtime (NEW)
  - Custom input fields for Script Stack defined (NEW)
  - Added image manager interface to the TextArea Stack (NEW)
  - Support for Google Analytics in Setup Stack depending on your publishing mode (NEW)
  - Add new QRcode image to ImageField Stack (NEW)
  - Switched from TinyMCE to CKEditor (NEW)
  - New %linkstatic(any text)% pattern in RichText Stack which can be used to create a static link (NEW)
  - Forms support now labels above fields (NEW)
  - New OptOut Stack to delete users or subscriptions (NEW)
  - Due to the new OptOut Stack, various Stacks have been renamed: Send Verify => Send Activation, Verify => Activate, Send Reset => Send Verification (NEW)
  - The Markdown Stack now supports an optional nowdoc syntax which requires PHP >= 5.3 (NEW)
  - The Script Stack now supports an optional nowdoc syntax which requires PHP >= 5.3 (NEW)
  - The Script2 Stack in turn has been removed (NEW)
  - Search Stack now supports multiple input fields (NEW)
  - Search Stack now uses the ButtonSet (NEW)
  - ButtonSet Stack now support Form, Login and Search Stacks (NEW)
  - Client-side paging option has been removed from TableGrid and Lister (NEW)
  - Custom URL scheme added to action type ‘dynamic’ (NEW)

  Bug Fixes

  - Repeated inputs have now a required marker as well (FIXED)
  - Try to restore the security context whenever possible (FIXED)
  - New error message that handles repeated input fields (FIXED)
  - Floater Stack now actually works :) (FIXED)
  - Custom port handling fixed (FIXED)
  - No absolute links required anymore - anywhere :) (FIXED)
  - Handling of login page call fixed (FIXED)
  - Send Activation & Send Verification Stacks don’t need absolute links anymore (FIXED)
  - Field validation fixed in case of empty field labels (FIXED)
  - Security fix, no details though :) (FIXED)
  - Required handling for Captcha has been implemented (FIXED)
  - Lister Stack table generation has been fixed (FIXED)
  - ADODB updated to version 5.18 (FIXED)
  - DB error handling has been improved (FIXED)
  - Apply value formater after size limiter for Fields (FIXED)
  - Fixed size limit and Markdown processing for TextFields (FIXED)
  - Paging has been fixed in TableGrids in case of ALL being selected (FIXED)
  - Some error handling added in redirectToLogin and all areas where an absolute path is calculated from a relative path (FIXED)
  - Various null accesses fixed (FIXED)
  - Lister iteration was broken (FIXED)

- Generated CSS uses now conditionals to generate cleaner code (FIXED)
- Layout issues of buttons in Login Stack fixed (FIXED)

Known Issues

- The security-enabled navbar is still limited in the way StackApps handles the “remember login” feature. Due to the fact that Stacks can not add code in front of everything (in this case the navbar) we can not detect an already logged in user before the navbar is rendered. This is only the case when accessing the page after a session timeout and the security context need to be established again.

- 1.1.0 Feature Release (TBD)

  Bug Fixes

  - Publishing error hopefully fixed once and forever that caused critical assets not to be published (FIXED)
  - ADODB_ASSOC_CASE = 2 in order to preserve the case of field names (FIXED)
  - ADODB updated to version 5.17 (FIXED)
  - Error handling for TableGrid/Lister with empty result sets fixed (FIXED)
  - Redirect workaround using now GET instead of POST to avoid browser resend confirmation dialogs (FIXED)
  - Logging to file is now supported (FIXED)
  - Login behaviour fixed: return session was not reset (FIXED)
  - Default setting in CMS Stack was broken (FIXED)
  - MSG_NO_DETAILS was not used in Details Stack (FIXED)
  - Updated the Global Setup to the current revison (FIXED)
  - Unique fields could not be validated (FIXED)
  - Support other ports than 80 (FIXED)
  - MSG_NO_ITEMS & MSG_NO_DETAILS handling has been changed. Field in TableGrid & Lister is empty (single space) now using the default message (FIXED)
  - CheckRadio default setting changed to checkbox widget instead of input widget (FIXED)
  - CheckRadio behaviour was not correct. Only active checkboxes could be written into the database (FIXED)
  - Handling of notice error messages (FIXED)

  New Features

  - Sparkle support added. Note: Before installation of the new version, the old version has to be removed (NEW)
  - Captcha Stack now supports font size settings (NEW)
  - New Iterator Stack to iterate over a record set and execute a custom script (NEW)
  - Image list in rich text editor (NEW)
  - Salted passwords, i.e. add a secret to the password before hashing (NEW)
  - New Newsletter Stack (NEW)
  - New Send Verify & Verify Stack to verify a email address and activate a new user account (self registration) (NEW)
  - New Send Reset & Reset Stack to reset forgoten passwords (self registration) (NEW)

- 1.0.4 Feature release (7/31/2012)

  New Features

  - New CAPTCHA component

  Known Issues

  - Sparkle update not yet implemented
  - The error log functionality is currently not implemented
  - Client-side paging is currently not activated
  - Due to a Stacks 2 issue two open projects can influence each other on a global Setup Stack level. Avoid to run two project open in parallel.

- 1.0.3 Bug fix release (7/29/2012)

  Fixes

  - improved form handling
  - fixed error reporting
  - fixed padding in Lister

  Known Issues

  - Sparkle update not yet implemented
  - The error log functionality is currently not implemented
  - Client-side paging is currently not activated
  - Due to a Stacks 2 issue two open projects can influence each other on a global Setup Stack level. Avoid to run two project open in parallel.

- 1.0.2 Bug fix release (7/24/2012)

  Fixes

  - transfer parameter handling

  Known Issues

  - Sparkle update not yet implemented
  - The error log functionality is currently not implemented
  - Client-side paging is currently not activated
  - Due to a Stacks 2 issue two open projects can influence each other on a global Setup Stack level. Avoid to run two project open in parallel.

- 1.0.1 Bug fix release (7/23/2012) First bug fix release of StackApps

  Fixes

  - ‘return’ session var has not been deleted properly
  - Added graceful PHP version handling even for version < 5.2

  Known Issues

  - Sparkle update not yet implemented
  - The error log functionality is currently not implemented
  - Client-side paging is currently not activated
  - Due to a Stacks 2 issue two open projects can influence each other on a global Setup Stack level. Avoid to run two project open in parallel.

- 1.0 Initial release (7/20/2012) This is a functional complete, but not fully tested release, which should not be used for production

  Known Issues

  - Sparkle update not yet implemented
  - The error log functionality is currently not implemented
  - Client-side paging is currently not activated
  - Due to a Stacks 2 issue two open projects can influence each other on a global Setup Stack level. Avoid to run two project open in parallel.
  - Everything else is working :)

⇤ TOC

6. Component Reference

Component Index

- CSS Stack
- Tag Stack
- Markdown Stack
- Floater Stack
- FieldSet Stack
- Hauler Stack

6.0 Feature Overview

Consistency

One of the main objectives during the component design was consistency: similar stacks have similar if not the same configurations & setups. Format options are always designed the same way and should be self explaining in most cases. One of the things you'll find with every stack is the so-called stacks ID which is always shown in the top/right corner similar to this example: stacks_in_183_page_0 - You can use this id to address this specific stack and its associated DIV within some custom CSS code. Use it wisely and know what you are doing.

Global Setup

As the Setup Stack is required on all pages that use StackApps Stacks, StackApps provides a special Stack that is unlike all other StackApps Stacks. The Setup Stack is available in two different versions: a Local Setup and a Global Setup Stack. The Global Setup Stack can be found within the Stack Templates section and basically shares all its settings across all of its instances. As you basically put the Setup Stack on all pages, the Global Setup Stack shares those settings on all pages and allows for a consistent setup even though there are multiple instances on the website.

Note: the Setup Stack is the first StackApp Stack on each of your pages that contain other StackApp Stacks.

But why do we need a normal Setup Stack anyway? First of all, the Global Setup is based on the Local Setup Stack anyway, but it is also used to overcome a limitation of the Global Setup Stack which is also its strength: you can only use one setup per Global Setup Stack. Local Setup Stacks can for instance access different databases on every page which could solve some rare application requirements.

In 99% of all cases you don't have to mess around with Local Setup Stacks and Global Setup Stacks will work for you with no limitations.

Security

StackApps comes with an easy to use though very powerful security framework that can handle most of your security requirements you'll find in your typical apps. In order to use the security framework you need a database table that holds at least a user ID, password and in case of a more finegrained security a security field to hold the access rights. You also need a page that holds the Login Stack. Activate the security feature in the Setup Stack and enter all the required data mentioned above.

Every page that needs to be protected and requires an authenticated user requires a Security Stack on top of the page, right after the Setup Stack. If you want to use finegrained security features, enter the list of security roles or a numeric access level, depending on the security type you have specified in the Setup Stack.

StackApps supports three different security types that can be used to tweak StackApps security model to your needs:

- *Authenticated:* requires the user to be authenticated to view a protected page
- *Access Level:* requires the user to have an numeric access level that is equal or higher than the page access level
- *Role-based:* requires the user to have a role that is part of the page roles to view the page

Depending on those types, your user database need to reflect those values in the security field.

Passwords used and stored by StackApps can be hashed in order to be protected from hackers and accidental access to the user database. Use this feature to avoid any security holes and attacks to your generated app.

In order to support the possibility to remember the user login for following sessions, StackApps stores certain infos (no passwords or such) between sessions. Depending on the remember login setting, those infos are either stored in the current session or in a persistent cookie with a given expiration date. In order to avoid any conflicts with other generated apps, the cookie name can be customized. Use a reverse domain naming scheme for this purpose.

Visibility

Based on StackApps security framework, StackApps also provides a visibility concept which allows for fine-grained visibility of database fields within a TableGrid, Lister or Details Stack. You can decide for every Field Stack if the field as such is visible or not visible based on your access rights or if you just want to make the field visible to everybody, but not the link that is associated with it. This allows for list that can be rendered for everybody, but the details link or edit link is only visible is you have the right to see it.

In case of TabGrids you can also make Columns only visible for certain user groups. This allows for instance for an edit button in a column that is shown only if the user has access to edit records.

Extended SQL

StackApps heavily relies on SQL when it comes to create queries within its components. But SQL alone doesn’t fly here as you will have to reference value from the StackApps runtime within the SQL very often. In order to make this possible StackApps extends SQL via a couple of functions that allow you to access runtime information:

source(name) where source ∊ {field, param, paramint, paramlookup, session, cookie, var, globals, server}

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **paramint:** access a URL int parameter (use this for int only params to avoid code injection)
- **paramlookup:** use a URL string parameter to lookup a value in another table
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

Those functions can also be used as default values for Input Stacks as well as variables within the RichTextField unsing the respective template syntax, e.g. %field(id)%.

Input & Action Params

StackApps uses URL parameters to transfer parameters between views and forms. To configure those parameters correctly, StackApps uses a simple URL parameter scheme:

- Input parameters within Form Stacks: field1=param1&field2=param2& …
- Action parameters within Fields Stacks: param1=field1&param2=field2& …

As you can see the destination is always on the left of the =, whereas the source is on the right.

Tansfer Params

One of the key features of StackApps to create more complex apps are transfer params, i.e. the possibility to transfer URL parameters to the next page to keep the state that has already been provided. Example: in a TableGrid Stack you show records based on a search query which should stay the same by paging through the TableGrid. In order to make this possible you have to declare the search parameter as a transfer param. Transfer params are defined as a & separated list. If you want to keep pageing or sorting parameters from a TableGrid or Lister, use wildcards to match specifc parameters, e.g. system parameter page_791 can be transferred via page*

Back References

One extremely helpful concept are Back References used when linking to an edit form. When using a back reference in a lin to a form, StackApps stores the page you are linking from in an special URL parameter. After the form has been processed and there is a bak reference available, the form redirects back to that page instead of using the form action. This is very useful in scenarios where you have an edit form for a record that can be reached from various other pages. In this case there is no single action page that can be used. Instead you can use the back reference to link back to each of the various originating pages. If you have a conflict in how the back referencing parameter is names, Setup provides you the flexibility to change that parameter name.

Joins

A JOIN in database lingo is a means for combining fields from two tables by using values common to each. StackApps provides something similar to JOINS, but not necessarily implemented as a JOIN. StackApps offers 2 types of JOINS, static JOINS and dynamic JOINS. Static JOINS are basically static mappings to map a value from one table to a different value. This is typically used to transform technical representations into readable representations. Let's imagine you would like to store priorities in a database as int value 1, 2, 3 to represent High, Medium and Low prios. You can easily translate the int value into the string representation via this static JOIN definition: '1,High|2,Medium|3,Low'.

The second JOIN type is the dynamic JOIN, which basically does this kind mapping using a second table, key and value field. Given the example above let's assume that the priorities are stored in a table names 'prios' with 'prio_id', 'prio_name' fields. The dynamic JOIN uses the prio valaue of the source table to index the 'prios' table via the 'prio_id' and use 'prio_name' as the display value.

So, which one is better, static or dynamic JOINS? Well, both have pros and cons. The static JOIN is lightweight and low maintenance, but static, whereas the dynamic JOIN has an overhead attached to it, but can grow at runtime.

Multivalues

Typically StackApps has to deal with single value fields like strings, integers, etc. But there are situations where you want to store multivalues within a database field. StackApps does that does that by storing multivalues in a list, e.g. list(1, 2, 3, 4) which is in turn stored as a string.

But how do you create multivalues? This can be done using a normal Input, entering the list definition, but is normally done via a CheckRadio and Select inputs, which support multivalues.

But how are multivalues displayed? All fields support a so called 'Format Output' setting, which is used to format Field outputs, be it a single or multivalue. The format string is setup like this 'list-element-format|last-element-format', where formats could be any string with %s replacing the actual field value. Here is an example:

Format String: '%s, |%s'
List: list(one,two,three)
Output: one, two, three

Use a single %s with no separator | to format single values.

Checkboxes vs Radio Buttons vs. Selects

The CheckRadio & Selct Inputs is very powerful when it comes to representing different input types and values. The following table lists all the different configurations that can be used with the Input

WIDGET	TYPE	MX	JOIN	Comment
Radio	Value(Join)	Yes	Yes	This is a list of mutually exclusive radio buttons with a value

				depending on the join definition
CheckBox	Multivalue(Join)	No	Yes	This is a list of checkboxes with a list of values depending on the join definition
CheckBox	Value(0 or 1)	No	No	This is a single checkbox with a value of 0 and 1 depending on the check state
Select	Value(Join)	Yes	Yes	This is a single-select list with a list of values depending on the join definition
Select	Multivalue(Join)	No	Yes	This is a multi-select list with a list of values depending on the join definition

CSS & Formatting

StackApps Stacks support a multitude of formatting options which basically translate into related CSS definition. In general, if there are formatting options in a component like a TableGrid, containing Stacks like Column or a TextField Stack can overwrite setting of the parent component and be more specific. Let's assume you have a font size of 16px defined for the table body, you can easily overwrite the setting for an individual table column content on the TextField level. A value of *inherit* basically inherits the CSS value from the parent, the more general setting.

If you feel that the given possibilities of a StackApps Stack are too limited, you can specify a custom class name which you can use in your own CSS file or within the CSS Stack that comes bundled with StackApps. Alternatively you can target specific stacks using the stacks ID that is shown on every stack in the right top-corner. There are no limits and whatever CSS can do for you is possible.

Beautiful Urls & Cruftless Links

StackApps fully supports URL rewriting and especially the cruftless links option that RapidWeaver provides. Enable this option to easily beautify URLs inside RapidWeaver. To make sure that you get nice descriptive Urls rename the path to the page in the respective page inspector.

Scripts

One of the powerful features of StackApps are the so-called Scripts. Scripts allow you to easily extend the functionality of the StackApps framework by adding custom code that can be triggered by certain events. What type of events are available depend on where you put the Script Stack. There are basically 4 types of locations

- *On top level*
  If you put the Script Stack on top level you can only use the *Here* trigger, which basically executes the script right here.
- *Inside a Record*
  i.e. TableGrid, Lister, Details, Login & Form you can use the two *Record Display* triggers, which can execute before or after the Record is displayed
- *Inside a Form*
  you can use the various Insert, Update, Delete and Cancel triggers, which can execute before or after the respective action takes place
- *Inside a Login*
  you can use the Do Login & Cancel Login trigger, which can execute before or after the login takes place

Please note: If you want to output something within you custom code, you should use the out() function. You can not use the standard print commands.

Markdown

StackApps comes with built-in Markdown support in all components that generate text output, i.e. TextField, RichTextField and CMS. When your field contains Markdown formated content, you can enable the Markdown formatter to convert the output into HTML. Use a standard TextArea to actually create Markdown content.

3rd Party Stacks Compatibility

StackApps is not guaranteed to work with other 3rd party stacks, but there is a high probability that StackApps is compatible. You should not place though other stacks into a TableGrid or Column Stack. Just give it a try and don’t be surprised if you run into issues. You’ll be warned :)

6.1 Global Setup Stack (Setup)

Overview

The Global Setup Stack is normally used on every StackApps page to give access to all global settings. In case you have to use more than one Setup Stack (you want to use a second database for instance) you have to use the Local Setup Stack in this case.

Container: Top Level

Config & Settings

Stack - Global Custom Messages

A set of messages that can be changed on a global basis as part of the Setup stack. Some of these messages can be changed on a stack level as well. You can use the built-in richtext editing features of Stacks to change the text format.

Setup - Publish

- **Publish Mode: (select)**
  - *Local Publish*: publish your app to a local web server. Use this mode during development.
  - *Remote Publish*: publish your app to a remote web server. Use this mode for production.

Both modes select a set of related settings that can be different for both publish modes

- **DSN: (input)**
  A database source that connects to the respective database, i.e. mysql://user:password@127.0.0.1:3306/database connects to db test on server 127.0.0.1 using user *root* and password *root*
- **Error Level: (select)**
  Sets the error reporting level for the respective publish mode. Make sure you use the right level during production. Please refer to the PHP reference for more details on the respective error levels
- **Log Errors?: (checkbox)**
  Do you want to log errors in a file insteaf of showing them on screen?
- **Error Log File: (input)**
  This is the file where errors will be logged to in case of file logging being enabled. Make sure that PHP has all the access rights to access the log file. Make sure to log errors to a file in a production system with log level set to E_ALL.

Examples: * *UNIX* /var/log/httpd/php_error_log * *OSX* /Users/yourname/Library/Logs/php_error.log

Use ‘syslog’ as a special value to use the system standard logger, i.e. * *UNIX* /var/log/messages * *OSX* /private/var/log/system.log

- **Google Analytics: (input)**
  An alternative way to integrate Google Analytics that respects the different publish profile, i.e. you can omit analytics for a local profile. Use the property ID provided by Google Analytics here.
- **Use Local jQuery: (checkbox)**
  Use a local jQuery instance instead of calling a CDN

- **Database Debug?: (checkbox)**
  This setting should only be used during development to trace all database related activities that take place within the ADODB framework. This is very helpful during problem resolution.

Setup - Security

- **Enable Security: (checkbox)**
  Enable the following security settings. You don’t have to enable these settings if you don’t need protected pages
- **Security Type: (select)**
  StackApps supports three different security types that can be used to adapt StackApps security model to your needs:
    - *Authenticated:* requires the user to be authenticated to view a page
    - *Access Level:* requires the user to have an numeric access level that is equal or higher than the page access level
    - *Role-based:* requires the user to have a role that is part of the page roles to view the page
- **User Table: (input)**
  This is the database table that holds all user data used by the security framework
- **User ID: (input)**
  This is the field that stores the user id in the database, i.e. the respective index in ther user table
- **Login: (input)**
  This is the field that stores the user login in the database
- **Password: (input)**
  This is the field that stores the user password in the database
- **Roles: (input)**
  The roles field stores the security level of the current user used for the security check described above.
- **Secret: (input)**
  StackApps uses a secret to hash login information during runtime. Make sure that this is large enough and keept secret!
- **Salt Password: (checkbox)**
  Salt passwords with the hash secret to make them even more secure
- **Cookie: (input)**
  In case a user remembers his login, the cookie name will be used to create the respective cookie. This way you can avaoid conflicts with other apps.
- **Expiration: (number)**
  Specify the number of days when the app cookie and therefor the login expires for a user.
- **Login Page: (link)**
  Select the page that holds the Login Stack. Put the the login page into the parent of any protected page or into the doc root of your site.

Setup - Extensions

- **BackRef Param: (input)**
  Provide the name of the systems back referencing parameter used in Forms and other places to store the page you are coming from
- **Font Awesome: (checkbox)**
  Enable Font Awesome Fonts
- **Google Web Fonts: (checkbox)**
  Enable Google Web Fonts
- **Font List: (input)**
  Provide a list of imported Google Web Fonts that can be used in font specifiers
- **Font Hack: (checkbox)**
  Set the new font for the BODY of the document to be used within the document
- **Less: (checkbox)**
  Enable global Less support
- **Stylesheet: (link)**
  Link to the less stylesheet typically in the resource section of the site
- **Lightbox: (select)**

Select a lightbox type, Slimbox2 or Colorbox

In case you select the Colorbox you have additional settings available

- **Theme: (select)**
  Select out of 5 different lightbox themes
- **Animation: (select)**
  Select between 3 different animations used by the lighbox
- **Scale Photos: (checkbox)**
  Are photos going to be scaled to fit on the screen?
- **Slideshow: (checkbox)**
  Photos are displayed in a slideshow that automatically starts and has a delay of 5 secs.
- **Localize: (input)**
  Use the string “close|previous|next|{current} of {total}|start show|stop show” to translate all Colorbox strings. Make sure you don’t modify the format.

Setup - Localization

- **Locale: (input)**
  Define the locale of your website. This is used during date conversion and other locale specific functionality. Use here the standard locale identifier
- **Timezone: (input)**
  The Timezone is needed for all date/time related functions and conversions. Just use the standard time zone here. Please refer to the PHP reference material for more details.
- **Date Format: (input)**
  Use a standard PHP date format string to format a date field output. This only applys for areas where no specific data format string is available.

⇤ **Component Index**

6.2 Login Stack (Security)

Overview

The Login Stack is a major part of StackApps security framework. There is typically one Login page within each StackApps project and the Login Stack basically does all what's required to create this Login page.

Container: Top Level

Config & Settings

Login - Labels & Fields

- **Labels: (input)**
  The main customization for the Login Stack takes place here where you define both the user login and password label. Remember, the actual fields are specified in the Setup Stack Security section. Labels can be formated using the built-in rich text editing of Stacks.
- **Desc: (input)**
  Enter a placeholder that will be shown as long as there is no input yet available

Login - ButtonSet Drop Zone

- In order to make the Login Stack complete you need to add a ButtonSet Stack within the drop zone in order to generate the required Cancel and OK buttons.

Login - Format

- **Login Font, Label Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Transform: (select)**
  Select a text transformation
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Error?: (color)**
  Change the validation error color by using the standard color swatch

Login - Layout

- **Label Width: (number)**
  Define the label width of the two input fields in px.
- **Field Width: (number)**
  Define the field width of the two input fields in px.
- **Field Width: (number)**
  Define the field height of the two input fields in px.
- **Field Gap?: (slider)**
  Define the gap between the user ID and password field in px.

- **Add Label Colon: (checkbox)**
  Automatically add a colon after each label name?

Login - Setup

- **Password Hashed?: (checkbox)**
  Is the password hashed in the database?
- **Hash Algorithm: (select)**
  Select the has method you want to use for hasing the password. Make sure that the hash method used here is the same as the one used in the admin form.
- **Remember Login?: (checkbox)**
  Do you want a 'Remember Login' Checkbox?
- **Display Logged-In State?: (checkbox)**
  Do you want to display the user and logout button on the login page if the user is logged in?

Login - Action

- **Login Page: (link)**
  Select the login page being called after the user has successfully signed in. This is not the case if the login is triggered by accessing a protected page. In this case the protected page will be called after signing in.
- **Cancel Page: (link)**
  Select the cancel page being called after the user hits the cancel or logout button of the login dialog.

⇤ Component Index

6.3 Security Stack (Security)

Overview

The Security Stack is the part of StackApps security framework that actually protects a certain page, i.e. the page requires a user to enter his credentials and has the appropriate access right according to the security type of the app defined in the Setup Stack.

Container: Top Level

Config & Settings

Security

- **Page Access: (input)**
  Specify the page accesss either as a numeric or a set of roles, depending on the security type of the app. For security type “Authenticated” you can leave this input empty. This is the same as with the Visibility Stack.
- **Login Page: (link)**
  Select the page that holds the Login Stack. This is an optional setting in case you want to use multiple logins. There are no limitations here where to put the login page.
- **Use SSL?: (checkbox)**
  Enforce the use of SSL for this page. Make sure that SSL is supported on your runtime environment.

⇤ **Component Index**

6.4 Visibility Stack (Security)

Overview

The Visibility Stack allows for a more fine-grained page security. Depending on the users access right you can show different content on the page or tell the user that he has to login in in order to see certain content.

Container: Top Level

Config & Settings

Visibility - Content Drop Zone

The two drop zones hold the respective stacks/content that will be shown if access is granted or not.

Visibility

- **Visibility: (input)**
  Specify the visibility level either as a numeric or a set of roles, depending on the security type of the app. For security type “Authenticated” you can leave this input empty. This is the same as with the Security Stack.

⇤ **Component Index**

6.5 Send Mail Stack (Security)

Overview

The Send Mail Stack is part of a stack duo together with the Activate Stack. The Send Mail Stack is used to send out an email within a Form Stack for instance to newly registered user or the administrator. When using a link within the generated mail, the user finally clicks on the link send to his email, which can in turn call the Activate Stack to actually confirm the email address and activate the account. Both stacks communicate via a common hash field that’s been used to identify the newly created record.

Container: Form

Config & Settings

Send Mail - Mail Template

Specify the verification mail that gets send out to the newly created user.

- **Subject: (input)** Specify the mail subject of the verification mail

source(name) where source ∊ {field, param, link, session, cookie, var, globals, server} joinparam(tableName, indexField, valueField)

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable
- **Body: (dropzone)** Specify the body part of the mail. You can use any kind of content stack to create the mail body.

source(name) where source ∊ {field, param, link, session, cookie, var, globals, server} joinparam(tableName, indexField, valueField)

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **link:** access to the link value and the link parameter value via the respective fieldname e.g. %link(hash_field)%
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

Send Mail - Setup

- **Email Field: (input)**
  The database field that holds the users email address.
- **Link Param: (input)**
  The parameter name used within the email embedded link. Typically used for the hash value that is used to check if the actual link send is called.
- **Send From: (input)**
  This is the send from address used for the activation email. This doesn’t have to be a real address.
- **HTML Mail?: (checkbox)** Do you want to send out an HTML mail or a plain text mail?

Send Mail - Action

- **Link Page: (link)**
  The link to the page provided in the email.
- **Success Page: (link)**
  The link to a success page. The related email address will be send via an URL parameter &email=john@doe.com
- **Error Page: (link)**
  The link to an error page. The related email address will be send via an URL parameter &email=john@doe.com

⇤ **Component Index**

6.6 Activate Stack (Security)

Overview

The Activate Stack is part of a stack duo together with the Send Activatation Stack. The Activate Stack is used to verify if the user has actually received the mail send by the Send Activation Stack via a unique hash code. If the email is actually verified, the security level of the user will be set to a specific value which gives the user full access to for instance his complete user account.

Container: Top Level

Config & Settings

Activate - Success Drop Zone

Add a content stack that will be displayed if the activation is successfull.

source(name) where source ∊ {field, param, session, cookie, var, globals, server} joinparam(tableName, indexField, valueField)

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

Activate - Error Drop Zone

Add a content stack that will be displayed if the activation is not successfull.

source(name) where source ∊ {param, session, cookie, var, globals, server} joinparam(tableName, indexField, valueField)

- **param:** access a URL string parameter
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

Activate - Setup

- **Table Name: (input)**
  Enter a table name if this is not the user database from the Setup Stack.
- **Hash Field: (input)**
  The database field that holds the hash generated for the email verification. The hash value is used to check if the actual link send is called.
- **Access Rights: (input)**
  Specify the access right the user gets assigned after a successful registration

⇤ Component Index

6.7 Send Verification Stack (Security)

Overview

The Send Verification Stack is part of a stack duo together with the Reset or the OptOut Stack. The Send Verification Stack is used to send out a verification email to a user . When the user finally clicks on the link send to him by email, he will verify the action and start the action depending on the used verification Stack.

Container: Top Level

Config & Settings

Send Verification - Mail Template

Specify the verification mail that gets send out to the newly created user.

- **Subject: (input)** Specify the mail subject of the verification mail

source(name) where source ∊ {field, param, link, session, cookie, var, globals, server} joinparam(tableName, indexField, valueField)

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable
- **Body: (dropzone)** Specify the body part of the mail. You can use any kind of content stack to create the mail body.

source(name) where source ∊ {field, param, link, session, cookie, var, globals, server} joinparam(tableName, indexField, valueField)

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **link:** access to the link value and the link parameter value via the respective fieldname e.g. %link(hash_field)%
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

Send Verification - Button Label & Preview

- **Button: (input)**
  Specify the search button label
- **Desc: (input)**
  Enter a placeholder that will be shown as long as there is no input yet available

Send Verification - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.

- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Transform: (select)**
  Select a text transformation
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Focus?: (color)**
  Change the focus color by using the standard color swatch

Send Verification - Layout

- **Field Width (number)**
  Define the search field width in px.
- **Field Height (number)**
  Define the search field width in px.
- **Float: (select)** Select the floating behaviour of the search field and the button: *left* or *right*

Send Verification - Button

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Transform: (select)**
  Select a text transformation
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Style: (select)**
  Select a text transformation
  - *Default*, i.e. the default button style.
  - *Class*, i.e. define a custom class
  - *Custom*, i.e. define the button style
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Background: (color)**
  Change the text color by using the standard color swatch
- **Border: (color)**
  Change the border color by using the standard color swatch
- **Style: (select)** Change the border style using the standard CSS values
- **Width: (slider)**
  Define button border width in px
- **Radius: (slider)**
  Define button border radius in px
- **Padding LR, Padding TB: (slider)**
  Specify the padding of the search button left, right and top, botton in px
- **Gap: (slider)**

Specify the gap between the input field and the button in px

Send Verification - Setup

- **Email Field: (input)**
  The database field that holds the users email address.
- **Link Param: (input)**
  The link parameter name used within the email embedded link.
- **Send From: (input)**
  This is the send from address used for the verification email. This doesn’t have to be a real address.
- **HTML Mail?: (checkbox)** Do you want to send out an HTML mail or a plain text mail?

Send Verification - Action

- **Link Page: (link)**
  The link to the verification page.
- **Success Page: (link)**
  The link to a success page. The related email address will be send via an URL parameter &email=john@doe.com
- **Error Page: (link)**
  The link to an error page. The related email address will be send via an URL parameter &email=john@doe.com

⇤ **Component Index**

6.8 Reset Stack (Security)

Overview

The Reset Stack is part of a stack duo together with the Send Verification Stack. The Reset Stack is used to actually reset a user password after the user has actually received the mail send by the Send Reset Verification via a unique hash code. The Reset Stack uses a normal Form Stack to actually set the new password or any additional profile information at this point.

Container: Top Level

Config & Settings

Reset - Form Drop Zone

Add a Form stack that will actually set the new password

Reset - Error Drop Zone

Add a content stack that will be displayed if the password reset is not successfull.

source(name) where source ∊ {param, session, cookie, var, globals, server} joinparam(tableName, indexField, valueField)

- **param:** access a URL string parameter
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

Reset - Setup

- **Hash Field: (input)**
  The database field that holds the hash generated for the email verification. The hash value is used to check if the actual link send is called.

⇤ **Component Index**

6.9 OptOut Stack (Security)

Overview

The OptOut Stack is part of a stack duo together with the Send Verification Stack. The OptOut Stack is used to actually delete a user account after the user has actually received the mail send by the Send Verification via a unique hash code.

Container: Top Level

Config & Settings

OptOut - Success Drop Zone

Add a content stack that will be displayed if the password reset is not successfull.

source(name) where source ∊ {param, session, cookie, var, globals, server} joinparam(tableName, indexField, valueField)

- **param:** access a URL string parameter
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

OptOut - Error Drop Zone

Add a content stack that will be displayed if the password reset is not successfull.

source(name) where source ∊ {param, session, cookie, var, globals, server} joinparam(tableName, indexField, valueField)

- **param:** access a URL string parameter
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

OptOut - Setup

- **Table Name: (input)**
  Enter a table name if this is not the user database from the Setup Stack.
- **Hash Field: (input)**
  The database field that holds the hash generated for the email verification. The hash value is used to check if the actual link send is called.

⇤ Component Index

6.10 TableGrid Stack (Read)

Overview

The TableGrid Stack is one of the most used of all StackApps Stacks. In the CRUD nomenclature, the TableGrid Stack is taking care of the Read part as you can display a set of records of a database table based on a custom query. The TableGrid Stack supports features like paging and column sorters and is highly customizable when it comes to the generated output. *Please note:* the TableGrid/Column Stack do not support other 3rd party Stacks for layout purposes. This includes even that built-in Stacks that come with Stacks out of the box. All formating and layout need to be inside the fields that get displayed. In case you need more advanced layouting, you might wanna consider to use the Lister Stack that has all that.

Container: Top Level

Config & Settings

TableGrid - Database Query

The query tries to resemble a simple SQL SELECT statement containing a FROM and a WHERE part. The FROM parts specifies the table we are looking at whereas the WHERE part defines the where clause that filters the table records. As we are always accessing ALL record fields, you don't have to specify the fields you are interested in. In addition there is no need to declare a sort order using ORDER BY, as this will be handled within the Column Stack.

- **FROM: (input)** Specify the table you are querying by entering the actual table name
- **WHERE: (input)** Specify the WHERE part of the SQL query. This is using regular SQL syntax. Keep in mind that strings within a WHERE clause are delimited with "string".

There are a couple of StackApps proprietary extensions available in order use certain custom value within your WHERE clause. You can insert these values by using the following syntax:

source(name) where source ∈ {field, param, paramint, paramlookup, session, cookie, var, globals, server}

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **paramint:** access a URL int parameter (use this for int only params to avoid code injection)
- **paramlockup:** lookup a URL string parameter via a table, index and value i.e. paramlookup(param, table, indexField, valueField)
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

  Examples:

  - field(task_done) <> 100
  - blog_active = 1 AND (blog_text LIKE "%param(search)%" or blog_title LIKE "%param(search)%")
  - inventory_id=paramint(id)

TableGrid - Global Custom Messages

- **First:, Prev:, Next:, Last:, Of:, Show: (input)**
  A set of messages that can be used to customize the TableGrid pager. You can use the built-in richtext editing features of Stacks to change the text format.

- **MSG_NO_ITEMS: (input)**
  You can customize the message that will be shown when viewing an empty result set on a TableGrid basis. This message can be changed on a global basis as well within the Setup Stack.

TableGrid - Columns Drop Zone

This is where you define how many columns your table will ultimately display. Add a Column Stack for each column you will display. Each of those columns will finally hold one Field Stack that will actually display the respective field value. Check the Column Stack and the various Field Stacks for more details.

TableGrid - Format

- **Header Font, Table Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Transform: (select)**
  Select a text transformation
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch

TableGrid - Layout

- **Table Width (input)**
  Define the table width in % or px
- **Padding X, Padding Y: (slider)**
  Define the used cell padding in px
- **Arrow Size: (slider)**
  Define the sorters arrow size in px
- **Remove Page HTML?: (checkbox)**
  Generate a table only without the surrounding page HTML. This can be used to feed apps like Panic’s Status Board. In this case you need to add <?php ob_start(); ?> to the Page Header>Prefix

TableGrid - Colors

- **Color: (color)**
  Change the border color by using the standard color swatch
- **Border Width: (slider)**
  Define table border width in px
- **Header Border: (checkbox)**
  Do you want to draw a header broder as well?
- **Header Row, Odd Row, Even Row: (color)**
  Change the header row, odd row and even row background color by using the standard color swatch
- **Opacity: (slider)**
  Change the background color opacity in %

TableGrid - Pager

- **Pager?: (checkbox)**
  Do you want to show/use a pager?

- **Pager Position: (select)**
  Where do you want to put the pager, top or bottom?
- **Pager Size: (number)**
  Define the number of list elements shown on a page

TableGrid - Params

- **Transfer: (input)**
  List of & separated URL parameters that will be transferred across links within the TableGrid, i.e. pager and sorter links. If you use for example a certain URL parameter to generate the TableGrid this takes care that the parameter will be saved while moving to the next page.

  Example: active,status

- **Exclude Param (checkbox)**
  Should the transfer parameters be excluded from the original URL parameters

⇤ Component Index

6.11 Column Stack (Read)

Overview

The Column Stack is used in combination with the TableGrid Stack. Within the TableGrid Stack you define the number of columns the table should have by adding the respective number of Column Stacks. Each Column Stack then holds one or more Field Stack that actually represents the value that will be displayed in the very table column. Beside that you typically define the sorting for the TableGrid within the Column Stack by marking the column as being sortable and if a column is initially sorted. Those infos will determine the ORDER BY field ASC/DESC part of the TableGrid SELECT that's been used.

In case you use more than one field per column, only the last one will be used to determine the column label and sort field. But you can use multiple fields to combine multiple joins. You do this by retrieving one field value, join it and store it in a variable. Use the variable as the field name for the next field and do the same thing again, but now using a new join.

Container: TableGrid

Config & Settings

Column - Field Drop Zone

Add one or many Field Stack to each column that will be used to retrieve the actual value for the table column. See above for a detailed explaination on how multiple fields will be treated.

Column - Format

- **Header Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Transform: (select)**
  Select a text transformation
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (color)**
  Change the text color by using the standard color swatch
- **Text: (color)**
  Change the text color by using the standard color swatch

Column - Layout

- **Header Align (select)**
  Define the horizontal alignment for the column/table header: *left, center, right, justify*
- **Cell Align (select)**
  Define the horizontal alignment for the column/table cell: *left, center, right, justify*
- **Column Width: (input)**
  Define column width in % or px
- **Visibility: (input)**
  Show the column only if the current user has the appropriate access right.

Column - Sorter

- **Enable Column Sorter? (checkbox)**
  Do you want to sort this column?
- **Initial Sort? (checkbox)**
  Do you want to sort the table by this column initially? *NOTE: There should only be one Column Stack with this checkbox set.*
- **Sort Descending? (checkbox)**
  Do you want to sort initially with descending order?

⇤ **Component Index**

6.12 Lister Stack (Read)

Overview

The Lister Stack is very similar to the TableGrid Stack in terms of being able to display a set of records of a database table based on a custom query. But other than the TableGrid Stack, the Lister Stack layouts the result set not in a table-stylel fashion but renders in a list-style fashion. Other than TableGrid Stack, the Lister Stack supports other 3rd party Stacks to layout the list according to your needs, which is not possible in a TableGrid. Like the TableGrid Stack the The Lister Stack supports paging.

Container: Top Level

Config & Settings

Lister - Database Query

The query tries to resemble a simple SQL SELECT statement containing a FROM, a WHERE and a ORDER BY part. The FROM parts specifies the table we are looking at whereas the WHERE part defines the where clause that filters the table records. As we are always accessing ALL record fields, you don't have to specify the fields you are interested in. In addition you can specify an ORDER BY part listing the fields that you want to use to sort the result set.

- **FROM: (input)** Specify the table you are querying by entering the actual table name
- **WHERE: (input)** Specify the WHERE part of the SQL query. This is using regular SQL syntax. Keep in mind that strings within a WHERE clause are delimited with "string".
- **ORDER BY: (input)** Specify the order of the result set by specifying the field name that's being used to order the set and add DESC or ASC for the sort order.

There are a couple of StackApps proprietary extensions available in order use certain custom value within your WHERE clause. You can insert these values by using the following syntax:

source(name) where source ∊ {field, param, session, cookie, var, globals, server}

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **paramint:** access a URL int parameter (use this for int only params to avoid code injection)
- **paramlockup:** join a URL string parameter via a table, index and value
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

  Examples:

  - field(task_done) <> 100
  - blog_active = 1 AND (blog_text LIKE "%param(search)%" or blog_title LIKE "%param(search)%")
  - inventory_id=paramint(id)

Lister - Global Custom Messages

- **First:, Prev:, Next:, Last:, Of:, Show: (input)**
  A set of messages that can be used to customize the TableGrid pager. You can use the built-in richtext editing features of Stacks to change the text format.
- **MSG_NO_ITEMS: (input)**

You can customize the message that will be shown when viewing an empty result set on a Lister basis. This message can be changed on a global basis as well within the Setup Stack.

Lister - Fields Drop Zone

This is where you define what kind of record fields you will ultimately display. Add as many Field Stacks as you want to display and use other layout stacks to layout and format your list according to your needs.

Lister - Layout

- **Float Items (checkbox)**
  Do you want to left float each of the fields, i.e. don’t create a seperate line for each Field Stack?
- **Add Label Colon: (checkbox)**
  Automatically add a colon after the label name?

Lister - Colors

- **Odd Row, Even Row: (color)**
  Change the odd row and even row background color by using the standard color swatch
- **Opacity: (slider)**
  Change the background color opacity in %

Lister - Pager

- **Pager?: (checkbox)**
  Do you want to show/use a pager?
- **Pager Position: (select)**
  Where do you want to put the pager, top or bottom?
- **Pager Size: (number)**
  Define the number of list elements shown on a page

Lister - Params

- **Transfer: (input)**
  List of & separated URL parameters that will be transferred across links within the Lister, i.e. pager links. If you use for example a certain URL parameter to generate the Lister this takes care that the parameter will be saved while moving to the next page.

  Example: active,status

⇤ **Component Index**

6.13 Iterator Stack (Read)

Overview

The Iterator Stack is very similar to the Lister Stack in terms of being able to iterate over a set of records of a database table based on a custom query. But other than the TableGrid Stack and the Lister Stack the Iterator Stack doesn't actually render a list or a table, but is usaly used to execute custom code for each record. This is done by using the Script Stack inside the Iterator Stack. You can access the current record element by using the varianle $record inside your custom code. Access a specific record field via $record->afield.

Container: Top Level

Config & Settings

Iterator - Database Query

The query tries to resemble a simple SQL SELECT statement containing a FROM, a WHERE and a ORDER BY part. The FROM parts specifies the table we are looking at whereas the WHERE part defines the where clause that filters the table records. As we are always accessing ALL record fields, you don't have to specify the fields you are interested in. In addition you can specify an ORDER BY part listing the fields that you want to use to sort the result set.

- **FROM: (input)** Specify the table you are querying by entering the actual table name
- **WHERE: (input)** Specify the WHERE part of the SQL query. This is using regular SQL syntax. Keep in mind that strings within a WHERE clause are delimited with “string”.
- **ORDER BY: (input)** Specify the order of the result set by specifying the field name that's being used to order the set and add DESC or ASC for the sort order.

There are a couple of StackApps proprietary extensions available in order use certain custom value within your WHERE clause. You can insert these values by using the following syntax:

source(name) where source ∊ {field, param, session, cookie, var, globals, server}

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **paramint:** access a URL int parameter (use this for int only params to avoid code injection)
- **paramlockup:** join a URL string parameter via a table, index and value
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

  Examples:

  - field(task_done) <> 100
  - blog_active = 1 AND (blog_text LIKE “%param(search)%” or blog_title LIKE “%param(search)%”)
  - inventory_id=paramint(id)

Iterator - Global Custom Messages

- **MSG_NO_ITEMS: (input)** You can customize the message that will be shown when viewing an empty result set on a Lister basis. This message can be changed on a global basis as well within the Setup Stack.

Iterator - Script Drop Zone

This is where your custom code goes using a Script Stack. You can access the current record element by using the varianle $record inside your custom code. Access a specific record field via $record->afield.

⇤ Component Index

6.14 Report Stack (Read)

Overview

The Report Stack can be used to export a DB table content based on a query similar to the TableGrid Stack. Column Stacks are used to define sort oder and concrete table fields that are used during the export. Currently the Report Stack generate CVS type exports as well as Status Board compatible JSON feeds, but additional export formats can be added in the future.

NOTE: in order to use the Report Stack on a page, you have to add <?php ob_start(); ?> before any code using RapidWeavers Header Prefix feature. If this code is not added accordingly, Report Stack does simply not work. In order to call the Report Stack, create a link to the repective page that hosts the Report Stack.

Container: Top Level

Config & Settings

Report - Database Query

The query tries to resemble a simple SQL SELECT statement containing a FROM and a WHERE part. The FROM parts specifies the table we are looking at whereas the WHERE part defines the where clause that filters the table records. As we are always accessing ALL record fields, you don't have to specify the fields you are interested in. In addition there is no need to declare a sort order using ORDER BY, as this will be handled within the Column Stack.

- **FROM: (input)** Specify the table you are querying by entering the actual table name
- **WHERE: (input)** Specify the WHERE part of the SQL query. This is using regular SQL syntax. Keep in mind that strings within a WHERE clause are delimited with "string".

There are a couple of StackApps proprietary extensions available in order use certain custom value within your WHERE clause. You can insert these values by using the following syntax:

source(name) where source ∊ {field, param, paramint, paramlookup, session, cookie, var, globals, server}

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **paramint:** access a URL int parameter (use this for int only params to avoid code injection)
- **paramlockup:** lookup a URL string parameter via a table, index and value i.e. paramlookup(param, table, indexField, valueField)
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

  Examples:

  - field(task_done) <> 100
  - blog_active = 1 AND (blog_text LIKE "%param(search)%" or blog_title LIKE "%param(search)%")
  - inventory_id=paramint(id)

Report - Columns Drop Zone

This is where you define how many columns your table will ultimately display. Add a Column Stack for each column you will display. Each of those columns will finally hold one Field Stack that will actually display the respective field value. Check the Column Stack and the various Field Stacks for more details.

Report - Setup

- **Format: (select)**
  Select the export format of this report, CSV, Status Board compatible JSON format or Table format. s.a. Status Board
- **Output: (select)**
  Select the output type i.e. stream or download. Stream generates a normal data stream in a given format, download initiates the download of a generated file. If you want to feed an app like Panic's Status Board, use stream.
- **Filename: (input)**
  Set a custom file name for the export file in case of download type. If empty, the table name and .csv as an file extension is used.
- **Header: (select)**
  Determine if the column header is based on the field name or the column label
- **Separator: (input)**
  The value separator used during CSV export.
- **Diagram: (select)**
  Choose between Bar and Line graph type. There is another feature that allows you to select the graph type via a URL parameter provided when calling the respective page URL. Use the parameter diagram=bar|line to select the respective graph type. This way you don't have to generate two different reports for each graph type.
- **Refresh: (number)**
  Set the refresh rate in seconds for the Status Board widget.
- **Colors: (input)**
  A comma separated list of color values that associate with the respective bar or graph item. Possible values are yellow, green, red, purple, blue, mediumGray, pink, aqua, orange, or lightGray
- **Show every X-axis label?: (bool)**
  Show every X-axis label.
- **Show Total?: (bool)**
  Add a total element to the graph.
- **Show every X-axis label?: (bool)**
  Show every X-axis label.
- **Y Min & Max Value: (number)**
  Set the displayed Y min and ax values.
- **Y Unit Pre & Suffix: (input)**
  Set the displayed Y min and max values.

⇤ Component Index

6.15 Details Stack (Read)

Overview

Other than the TableGrid and Lister Stacks, the Details Stack just displays a single table record of a database. Typically this is a details page that shows all fields of the record instead of just a few like in a TableGrid or Lister. The typical use case here is that by clicking on a record in a TableGrid or Lister Stack you will link to the Details Stack to actually show the details of that specific record.

Container: Top Level

Config & Settings

Details - Database Query

The query tries to resemble a simple SQL SELECT statement containing a FROM, and WHERE. The FROM parts specifies the table we are looking at whereas the WHERE part defines the where clause that filters the table records. As we are always accessing ALL record fields, you don't have to specify the fields you are interested in. 99% of all where clauses are referring the the tables primary key here.

- **FROM: (input)** Specify the table you are querying by entering the actual table name
- **WHERE: (input)** Specify the WHERE part of the SQL query. This is using regular SQL syntax. Keep in mind that strings within a WHERE clause are delimited with "string".

There are a couple of StackApps proprietary extensions available in order use certain custom value within your WHERE clause. You can insert these values by using the following syntax:

source(name) where source ∊ {field, param, session, cookie, var, globals, server}

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **paramint:** access a URL int parameter (use this for int only params to avoid code injection)
- **paramlookup:** join a URL string parameter via a table, index and value
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

  Examples:

  - inventory_id=param(id)

Details - Global Custom Messages

- **MSG_NO_ITEMS: (input)**
  You can customize the message that will be shown when viewing an empty result set on a Details basis. This message can be changed on a global basis as well within the Setup Stack.

Details - Fields Drop Zone

This is where you define what kind of record fields you will ultimately display. Add as many Field Stacks as you want to display and use other layout stacks to layout and format your output according to your needs.

Details - Format

- **Details Font, Label Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Transform: (select)**
  Select a text transformation
- **Transform: (select)**
  Select a text transformation
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch

Details - Layout

- **Label Width (number)**
  Define the label width of the fields in px on a Details Stack level. You can always overwrite the setting for each individual Field Stack.
- **Add Label Colon: (checkbox)**
  Automatically add a colon after the label name?

⇤ Component Index

6.16 Newsletter Stack (Read)

Overview

The Newsletter Stack is used to send out a newsletter to a group of subscribers, usually your customers or users. You typically have a newsletter table which holds the newsletter body text. This newsletter table can be managed with your own Form and can be displayed with a TableGrid Stack and you can link from this page to the actual send newsletter page which holds the Newsletter Stack using an URL parameter specifying the newsletter that should be send out to your users. On the Newsletter page you include the Newsletter Stack and inside the Newsletter Stack you place a Details Stack that retrieves the newsletter content for the specific newsletter being send out to your subscribers based on the URL parameter. Use the appropriate query in the Details Stack.

Container: Top Level

Config & Settings

Newsletter - Database Query

The query tries to resemble a simple SQL SELECT statement containing a FROM, and WHERE. The FROM parts specifies the table we are looking at whereas the WHERE part defines the where clause that filters the table records. As we are always accessing ALL record fields, you don’t have to specify the fields you are interested in. 99% of all where clauses are referring the the tables primary key here.

- **FROM: (input)** Specify the table you are querying by entering the actual table name
- **WHERE: (input)** Specify the WHERE part of the SQL query. This is using regular SQL syntax. Keep in mind that strings within a WHERE clause are delimited with “string”.

There are a couple of StackApps proprietary extensions available in order use certain custom value within your WHERE clause. You can insert these values by using the following syntax:

source(name) where source ∊ {field, param, session, cookie, var, globals, server}

- **field:** access a field of the record set
- **param:** access a URL string parameter
- **paramint:** access a URL int parameter (use this for int only params to avoid code injection)
- **paramlookup:** join a URL string parameter via a table, index and value
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

  Examples:

  - inventory_id=param(id)

Newsletter - Button Label & Preview

- **Button: (input)** Specify the search button label

Newsletter - Details Drop Zone

This is where you define what kind of record fields you will ultimately display. Add as many Field Stacks as you want to display and use other layout stacks to layout and format your output according to your needs.

Newsletter - Button

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Transform: (select)**
  Select a text transformation
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Style: (select)**
  Select a text transformation
  - *Default*, i.e. the default button style.
  - *Class*, i.e. define a custom class
  - *Custom*, i.e. define the button style
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Background: (color)**
  Change the text color by using the standard color swatch
- **Border: (color)**
  Change the border color by using the standard color swatch
- **Style: (select)**
  Change the border style using the standard CSS values
- **Width: (slider)**
  Define button border width in px
- **Radius: (slider)**
  Define button border radius in px
- **Padding LR, Padding TB: (slider)**
  Specify the padding of the search button left, right and top, botton in px

Newsletter - Setup

- **Email Field: (input)**
  The database field that holds the users email address.
- **Send From: (input)**
  This is the send from address used for the verification email. This doesn’t have to be a real address.
- **HTML Mail?: (checkbox)**
  Do you want to send out an HTML mail or a plain text mail?

⇤ Component Index

6.17 Search Stack (Search)

Overview

The Search Stack is typically used in combination with a TableGrid or Lister Stack. It generates a search field and a search button that will trigger another page that will be called with a search parameter included in the URL that can be used on the new page to filter a TableGrid or Lister Stack.

Container: Top Level

Config & Settings

Search - Input & Button Drop Zone

Add Input Stacks and a ButtonSet Stack to build the search form

Search - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Transform: (select)**
  Select a text transformation
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Focus?: (color)**
  Change the focus color by using the standard color swatch

Search - Layout

- **Field Width (number)**
  Define the search field width in px.
- **Field Height (number)**
  Define the search field width in px.

Search - Button

- **Font: (select)** Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)** Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Transform: (select)**
  Select a text transformation
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic

- **Style: (select)**
  Select a text transformation
  - *Default*, i.e. the default button style.
  - *Class*, i.e. define a custom class
  - *Custom*, i.e. define the button style
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Background: (color)**
  Change the text color by using the standard color swatch
- **Border: (color)**
  Change the border color by using the standard color swatch
- **Style: (select)** Change the border style using the standard CSS values
- **Width: (slider)**
  Define button border width in px
- **Radius: (slider)**
  Define button border radius in px
- **Padding LR, Padding TB: (slider)**
  Specify the padding of the search button left, right and top, botton in px

Search - Action

- **Action: (link)** Select the action page of the search form that will be called after pressing the search button or hitting the return key.
- **Transfer: (input)**
  A list of & separated URL parameters that will be transfered from the original URL *Example: order&comment_id*
- **Exclude Param (checkbox)**
  Should the transfer parameters be excluded from the original URL parameters

⇤ Component Index

6.18 TextField Stack (Read)

Overview

The TextField Stack is most probably one of the most used stacks in combination with the TableGrid and Lister Stacks. It’s main purpose is to retrieve a field value from the current result set, format the output and display it on the page. The output can be adjusted via various options and modified with an powerful join mechanism. Finally the output can be wrapped with a link to another page including multiple URL parameters.

Container: TableGrid, Lister, Details

Config & Settings

TextField - Main

- **Label: (input)**
  Define the label name of the field
- **Field: (input)**
  Define the field name as used in the database or reference an existing variable, e.g. $result

TextField - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Class: (input)**
  Add one or a list of css classes to the field DIV

TextField - Layout

- **Show Label: (checkbox)**
  Do you want to show the field label?
- **Label Width (input)**
  Define the label width in % or px
- **Align (select)**
  Define the horizontal alignment field value: *left, center, right, justify*
- **Float (checkbox)**
  Do you want to float the field?
- **Direction (select)**
  Define the direction of the float: *left or right*

TextField - Output

- **Use Markdown?: (checkbox)**
  Do you use the Markdown markup inside the content?

- **Strip HTML?: (checkbox)**
  Do you want to strip all HTML markup before rendering the content?
- **Limit: (number)**
  Limit the output to a specified number of characters. Leave this empty if you don't want a limit at all. This limit applies for each element of multi-value fields. Format output is applied after the size has been limited.
- **Format: (input)**
  This is a format string that defines how the field output will be rendered. Use %s as a variable for the field content within the format string. For multivalues the string contains two parts separated by a | character. The first part defines how list elements will be rendered, whereas the second part is used for the last element of a list. *Example: %s, | %s renders a list of 1,2,3,4 as 1, 2, 3, 4*
- **Static Text: (input)**
  This text is rendered no matter of the value of the field. This is an option to create static text links with a dynamic URL. You don't use a database field in this case, especially not the field you wanna use in the action. Simply leave the field name empty.
- **Join: (select)**
  Joins are used to translate an actual field value into another value by using the actual field value as an index to find the new value. Joins can be either
  - *none*, i.e. the original value will be displayed.
  - *static*, i.e. based on a static mapping list containing index and value pairs.
    - **Key/Value List: (input)**
      A list of key,value pairs seperated by a | pipe. *Example: 1,foo|2,bar|3,foobar*
  - *dynamic*, i.e. based on another database table, an index field and a value field or
    - **Table: (input)**
      The name of the join table
    - **Index Field: (input)**
      The field used as an index into the join table
    - **Value Field: (input)**
      The field used as the mapping value from the join table
    - **Default Value: (input)**
      The value that should be mapped in case of a no match

  Depending on your concrete situation or requirement choose either one or the other. The dynamic join can dynamically extended during run time while the static join is fixed during compile time.
- **Target: (select)**
  A field normally generates output that gets rendered on the page directly. In order to work with the output of the field in another later step, e.g. in a Richtext Field, you can store the value in a variable that can be accessed either in custom code or via %global(var)% resp. global(var) later.
  - *Page*, i.e. output goes to the HTML page.
  - *Variable*, i.e. output goes to a custom variable
    - **Variable Name: (input)**
      The name of the variable, e.g. $result, to store the output
- **Visibility: (input)**
  Control the visibility of the field by specifying a access level.

TextField - Action

- **Type: (select)**
  Select the action type used for the link that's being generated
  - *none*, i.e. no link will be generated.
  - *static*, i.e. the field links to a static page with a set of action parameters encoded in the URL.
    - **Page: (link)**
      The page that will be linked to
    - **Params: (input)**
      Empty or a URL parameter list of param=field pairs *Exmaple: id=blog_id&comment=comment_id*
    - **Transfer: (input)**

        A list of & separated URL parameters that will be transfered from the original URL *Example: order&comment_id*
        - **Exclude Param (checkbox)**
          Should the transfer parameters be excluded from the original URL parameters
        - **Add Back Reference? (checkbox)**
          Should we add an automatic back reference to the current page to link back from a form?
    - *dynamic*, i.e. based on another field that contains the link URL
        - **Page: (link)**
          Base URL used to append the field value to generate a fully qualified link. Leave this empty if the complete link is stored in the database
        - **Field: (link)**
          A field that contains the actual link URL
        - **Schema: (select)**
          The schema used for the dynamic action, i.e. http://, mailto:, etc.
    - **Title: (input)**
      Specify a fieldname used to retrieve the links title attribute or leave empty.
    - **Class: (input)**
      Specify a Class attribute for the link used in lightboxes for instance.
    - **REL: (input)**
      Specify a REL attribute for the link used in lightboxes for instance. Use %s to insert the unique Stack ID within the string.
    - **Visibility: (input)**
      Control the visibility of the link by specifying a access level.

⇤ **Component Index**

6.19 RichTextField Stack (Read)

Overview

The RichTextField Stack is similar to the TextField Stack but allows for multiple field values to be displayed at once using a pattern mechanism and the built-in richtext formating of Stacks.

Container: TableGrid, Lister, Details

Config & Settings

RichTextField - Main

Use the main editing area of the RichTextField to layout the output. To actually retrieve values to be displayed use as many %[mod]source(name)% expressions as needed, where mod ∈ {link,join,linkjoin}, source ∈ {field,static,param,session,cookie,var,globals,server} link will create a action link for this expression, whereas join will join the expression value with the Join definition of the field. Please keep in mind that there is only one action and join definition per Stack. If you need more complex settings, you have to split those things up.

- **field:** access a field of the record set
- **static:** use a static text
- **param:** access a URL parameter
- **session:** access a session parameter
- **cookie:** access a cookie
- **var:** access a variable
- **globals:** access a global variable
- **server:** access a server variable

RichTextField - Format

- **Class: (input)**
  Add one or a list of css classes to the field DIV

RichTextField - Layout

- **Margin Left: (number)**
  Set the left margin for the field in situations where you want to move the field content to the left on a Details Stack. In this case use negative values for the margin that reflects the label width, e.g. –100px
- **Float (checkbox)**
  Do you want to float the field?
- **Direction (select)**
  Define the direction of the float: *left or right*

RichTextField - Output

- **Use Markdown?: (checkbox)**
  Do you use the Markdown markup inside the content?
- **Strip HTML?: (checkbox)**
  Do you want to strip all HTML markup before rendering the content?
- **Limit: (number)**
  Limit the output to a specified number of characters. Leave this empty if you don’t want a limit at all. This limit applies for each element of multi-value fields. Format output is applied after the size has been limited.
- **Format: (input)**
  This is a format string that defines how the field output will be rendered. Use %s as a variable for the field content within the format string. For multivalues the string contains two parts separated by a I character. The first part

defines how list elements will be rendered, whereas the second part is used for the last element of a list. *Example: %s, | %s renders a list of 1,2,3,4 as 1, 2, 3, 4*

- **Date Format: (input)**
  Use a standard PHP date format string to format a date field output. This obviously only applys for fields of data type time, date, etc.
- **Static Text: (input)**
  This text is rendered no matter of the value of the field. This is an option to create static text links with a dynamic URL.
- **Join: (select)**
  Joins are used to translate an actual field value into another value by using the actual field value as an index to find the new value. Joins can be either
  - *none*, i.e. the original value will be displayed.
  - *static*, i.e. based on a static mapping list containing index and value pairs.
    - **Key/Value List: (input)**

      A list of key,value pairs seperated by a | pipe. *Example: 1,foo|2,bar|3,foobar*
  - *dynamic*, i.e. based on another database table, an index field and a value field or
    - **Table: (input)**

      The name of the join table
    - **Index Field: (input)**

      The field used as an index into the join table
    - **Value Field: (input)**

      The field used as the mapping value from the join table
    - **Null Value: (input)**

      The value that should be mapped in case of a no match

  Depending on your concrete situation or requirement choose either one or the other. The dynamic join can dynamically extended during run time while the static join is fixed during compile time.
- **Target: (select)**
  A field normally generates output that gets rendered on the page directly. In order to work with the output of the field in another later step, e.g. in a Richtext Field, you can store the value in a variable that can be accessed either in custom code or via %global(var)% resp. global(var) later.
  - *Page*, i.e. output goes to the HTML page.
  - *Variable*, i.e. output goes to a custom variable
    - **Variable Name: (input)**

      The name of the variable, e.g. $result, to store the output
- **Visibility: (input)**
  Control the visibility of the field by specifying a access level.

RichTextField - Action

- **Type: (select)**
  Select the action type used for the link that's being generated
  - *none*, i.e. no link will be generated.
  - *static*, i.e. the field links to a static page with a set of action parameters encoded in the URL.
    - **Page: (link)**

      The page that will be linked to
    - **Params: (input)**

      Empty or a URL parameter list of param=field pairs *Exmaple: id=blog_id&comment=comment_id*
    - **Transfer: (input)**

      A list of & separated URL parameters that will be transfered from the original URL *Example: order&comment_id*
    - **Exclude Param (checkbox)**

      Should the transfer parameters be excluded from the original URL parameters
    - **Add Back Reference? (checkbox)**

        Should we add an automatic back reference to the current page to link back from a form?
- *dynamic*, i.e. based on another field that contains the link URL
    - **Page: (link)**
      Base URL used to append the field value to generate a fully qualified link. Leave this empty if the complete link is stored in the database
    - **Field: (link)**
      A field that contains the actual link URL
    - **Schema: (select)**
      The schema used for the dynamic action, i.e. http://, mailto:, etc.
- **Title: (input)**
  Specify a fieldname used to retrieve the links title attribute or leave empty.
- **Class: (input)**
  Specify a Class attribute for the link used in lightboxes for instance.
- **REL: (input)**
  Specify a REL attribute for the link used in lightboxes for instance. Use %s to insert the unique Stack ID within the string.
- **Visibility: (input)**
  Control the visibility of the link by specifying a access level.

⇤ Component Index

6.20 DateField Stack (Read)

Overview

The DateField Stack is very similar to the TextField Stack, but is used for date/time format fields.

Container: TableGrid, Lister, Details

Config & Settings

DateField - Main

- **Label: (input)**
  Define the label name of the field
- **Field: (input)**
  Define the field name as used in the database

DateField - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Class: (input)**
  Add one or a list of css classes to the field DIV

DateField - Layout

- **Show Label: (checkbox)**
  Do you want to show the field label?
- **Label Width (input)**
  Define the label width in % or px
- **Align (select)**
  Define the horizontal alignment field value: *left, center, right, justify*
- **Float (checkbox)**
  Do you want to float the field?
- **Direction (select)**
  Define the direction of the float: *left or right*

DateField - Output

- **Date Format: (input)**
  use a standard PHP date format string to format a date field output. This obviously only applys for fields of data type time, date, etc.
- **Target: (select)**
  A field normally generates output that gets rendered on the page directly. In order to work with the output of the field

in another later step, e.g. in a Richtext Field, you can store the value in a variable that can be accessed either in custom code or via %global(var)% resp. global(var) later.
    - *Page*, i.e. output goes to the HTML page.
    - *Variable*, i.e. output goes to a custom variable
        - **Variable Name: (input)**
          The name of the variable, e.g. $result, to store the output
- **Visibility: (input)**
  Control the visibility of the field by specifying a access level.

DateField - Action

- **Type: (select)**
  Select the action type used for the link that's being generated
    - *none*, i.e. no link will be generated.
    - *static*, i.e. the field links to a static page with a set of action parameters encoded in the URL.
        - **Page: (link)**
          The page that will be linked to
        - **Params: (input)**
          Empty or a URL parameter list of param=field pairs *Exmaple: id=blog_id&comment=comment_id*
        - **Transfer: (input)**
          A list of & separated URL parameters that will be transfered from the original URL *Example: order&comment_id*
        - **Exclude Param (checkbox)**
          Should the transfer parameters be excluded from the original URL parameters
        - **Add Back Reference? (checkbox)**
          Should we add an automatic back reference to the current page to link back from a form?
    - *dynamic*, i.e. based on another field that contains the link URL
        - **Page: (link)**
          Base URL used to append the field value to generate a fully qualified link. Leave this empty if the complete link is stored in the database
        - **Field: (link)**
          A field that contains the actual link URL
        - **Schema: (select)**
          The schema used for the dynamic action, i.e. http://, mailto:, etc.
    - **Title: (input)**
      Specify a fieldname used to retrieve the links title attribute or leave empty.
    - **Class: (input)**
      Specify a Class attribute for the link used in lightboxes for instance.
    - **REL: (input)**
      Specify a REL attribute for the link used in lightboxes for instance. Use %s to insert the unique Stack ID within the string.
    - **Visibility: (input)**
      Control the visibility of the link by specifying a access level.

⇤ Component Index

6.21 ImageField Stack (Read)

Overview

The ImageField Stack can be used to display images (single or multivalue) either by retrieving their filename or URL from the database. This Stack supports fields with multiple values.

Container: TableGrid, Lister, Details

Config & Settings

ImageField - Main

- **Label: (input)**
  Define the label name of the field
- **Field: (input)**
  Define the field name as used in the database
- **Default: (image)**
  The default images that gets displayed in case there is no image available in the database

ImageField - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Class: (input)**
  Add one or a list of css classes to the field DIV

ImageField - Layout

- **Show Label: (checkbox)**
  Do you want to show the field label?
- **Label Width (input)**
  Define the label width in % or px
- **Align (select)**
  Define the horizontal alignment field value: *left, center, right, justify*
- **Float (checkbox)**
  Do you want to float the field?
- **Direction (select)**
  Define the direction of the float: *left or right*

ImageField - Output

- **Type: (select)**
  Choose the image type you want to use. This could be a normal image or a QRcode that is generated based on a

field value or an action URL
  - *Image*, i.e. a normal image
  - **Image Dir: (link)**
    Link to the directory where the images are stored. Leave this empty if you are using external URLs.
    - **Max Width: (number)**
      Limit the maximum image width to x px. You can leave the setting empty.
    - **Max Height: (number)**
      Limit the maximum image height to x px. You can leave the setting empty.
    - **Use TimThumb Extension: (checkbox)**
      Use a server-side image processor to size and process images on the server.
    - **TimThumb Params: (input)**
      Add additional parameters to enable even more sophisticated TimThumb features. Check the TimThumb page for a good overview and examples.
    - **ALT Attr: (input)**
      This is ALT attribute that's used for this image.
  - *ORcode*, i.e. a QRcode generated based on the field value
    - **Data Source: (select)**
      Either Field or Action. Field uses the field value wheras Action uses the URL that's generated by the Stacks Action setting.
    - **Foreground: (color)**
      The QRcode foreground color
    - **Background Color?: (checkbox)**
      Do you wanna use a background color or a transparent background?
    - **Background: (color)**
      The QRcode bckground color
    - **Render Method: (select)**
      Either canvas or div
    - **Width: (number)**
      Image width to x px.
    - **Height: (number)**
      Image height to x px.
- **Join: (select)**
  Joins are used to translate an actual field value into another value by using the actual field value as an index to find the new value. Joins can be either
  - *none*, i.e. the original value will be displayed.
  - *static*, i.e. based on a static mapping list containing index and value pairs.
    - **Key/Value List: (input)**
      A list of key,value pairs seperated by a I pipe. *Example: 1,fooI2,barI3,foobar*
  - *dynamic*, i.e. based on another database table, an index field and a value field or
    - **Table: (input)**
      The name of the join table
    - **Index Field: (input)**
      The field used as an index into the join table
    - **Value Field: (input)**
      The field used as the mapping value from the join table
    - **Null Value: (input)**
      The value that should be mapped in case of a no match

  Depending on your concrete situation or requirement choose either one or the other. The dynamic join can dynamically extended during run time while the static join is fixed during compile time.
- **Target: (select)**
  A field normally generates output that gets rendered on the page directly. In order to work with the output of the field

in another later step, e.g. in a Richtext Field, you can store the value in a variable that can be accessed either in custom code or via %global(var)% resp. global(var) later.
  - *Page*, i.e. output goes to the HTML page.
  - *Variable*, i.e. output goes to a custom variable
    - **Variable Name: (input)**
      The name of the variable, e.g. $result, to store the output
- **Visibility: (input)**
  Control the visibility of the field by specifying a access level.

ImageField - Action

- **Type: (select)**
  Select the action type used for the link that’s being generated
  - *none*, i.e. no link will be generated.
  - *static*, i.e. the field links to a static page with a set of action parameters encoded in the URL.
    - **Page: (link)**
      The page that will be linked to
    - **Params: (input)**
      Empty or a URL parameter list of param=field pairs *Exmaple: id=blog_id&comment=comment_id*
    - **Transfer: (input)**
      A list of & separated URL parameters that will be transfered from the original URL *Example: order&comment_id*
    - **Exclude Param (checkbox)**
      Should the transfer parameters be excluded from the original URL parameters
    - **Add Back Reference? (checkbox)**
      Should we add an automatic back reference to the current page to link back from a form?
  - *dynamic*, i.e. based on another field that contains the link URL
    - **Page: (link)**
      Base URL used to append the field value to generate a fully qualified link. Leave this empty if the complete link is stored in the database
    - **Field: (link)**
      A field that contains the actual link URL
    - **Schema: (select)**
      The schema used for the dynamic action, i.e. http://, mailto:, etc.
  - **Title: (input)**
    Specify a fieldname used to retrieve the links title attribute or leave empty.
  - **Class: (input)**
    Specify a Class attribute for the link used in lightboxes for instance.
  - **REL: (input)**
    Specify a REL attribute for the link used in lightboxes for instance. Use %s to insert the unique Stack ID within the string.
  - **Visibility: (input)**
    Control the visibility of the link by specifying a access level.

⇤ **Component Index**

6.22 CMS Stack (Read)

Overview

The CMS Stack is a very powerful stack that can be used on top level (i.e. no TableGrid, Lister or Details Stack required) to retrieve content from a database and display it either within the page main content area or in an ExtraContent area. If your template supports ExtraContent, you can even target those areas by selecting the specific EC area.

Container: Top Level

Config & Settings

CMS - Content ID & Area

- **Content ID: (input)**
  Specify the ID of the content that should be retrieved

CMS - Global Custom Messages

- **MSG_NO_CONTENT: (input)**
  You can customize the message that will be shown when viewing an empty content box on a CMS basis. This message can be changed on a global basis as well within the Setup Stack.

CMS - Database

- **Mode: (select)**
  Select static or dynamic content ID mode. The static mode uses a static content ID to retrieve the content from the DB whereas the dynamic mode uses an URL parameter to send the content ID to the Stack.
- **Table: (input)**
  Specify the content table name
- **ID Field: (input)**
  Specify the content ID field that will be used to retrieve the content based on the specifc content ID mentioned above
- **Content Field: (input)**
  Specify the content field that actually holds the content

CMS - Output

- **Use Markdown?: (checkbox)**
  Do you use the Markdown markup inside the content?
- **Strip HTML?: (checkbox)**
  Do you want to strip all HTML markup before rendering the content?
- **Limit: (number)**
  Limit the output to a specified number of characters. Leave this empty if you don’t want a limit at all.
- **Target: (select)**
  Select the page area you want to target with the content?
  - *Header*, i.e. the header area
    - **Position: (select)**
      Specify position relative to the target, i.e. Before or After
  - *Sidebar*, i.e. the sidebar area
    - **Position: (select)**
      Specify position relative to the target, i.e. Before or After
  - *Content*, i.e. the content area

    - *Footer*, i.e. the footer area
        - **Position: (select)**
          Specify position relative to the target, i.e. Before or After
    - *ExtraContent (static)*, i.e. the content is rendered in a static ExtraContent area
        - **EC Area: (number)**
          Specify the ExtraContent area
    - *ExtraContent (dynamic)*, i.e. the content is rendered in a dynamic ExtraContent area
        - **EC Area Field: (input)**
          Specify the database field to retrieve the ExtraContent area

CMS - Action

- **Edit Button?: (checkbox)**
  Do you want to show an edit button that directly links to the respective content editor?
- **Text link: (input)**
  This is the text link that will be generated in case of the edit button is enabled
- **Visibility: (input)**
  Specify which access level is able to see the edit button
- **Editor Page: (link)**
  Select the page that will be called if you press the edit button. This page should host the editor for this content.
- **Param: (input)**
  Specify the parameter and the primary key field that will be used as an index to the edit page. Example: id=content_id

⇤ **Component Index**

6.23 Form Stack (Create, Update, Delete)

Overview

The Form Stack is the heard of any update operation of the database. Through the Form Stack you can create, update and finally delete database records. In order to do its work, the Form Stack uses a set of Input Stacks which are representing the various database fields within the form.

Container: Top Level

Config & Settings

Form - Database Query

The query tries to resemble a simple SQL UPDATE statement with a table name and a WHERE part. The WHERE part defines the where clause that determines the record that gets updated by the form. This is typically done using the primary key of the table and an external parameter.

- **UPDATE: (input)** Specify the table you are updating by entering the actual table name
- **WHERE: (input)** Specify the WHERE part of the SQL query. This is using regular SQL syntax plus the special variables param(name) and paramint(name) that can be used to access an external URL parameter. Keep in mind that strings within a WHERE clause are delimited with “string”.
- **param:** access a URL string parameter
- **paramint:** access a URL int parameter (use this for int only params to avoid code injection)

  Examples:

  - inventory_id=param(id)

Form - Custom Messages

- **MSG_DELETE_RECORD: (input)**
  You can customize the message that will be shown when trying to delete a recortd on a Form basis. This message can be changed on a global basis as well within the Setup Stack.

Form - Input Drop Zone

This is where you add the various Input Stacks of the form. make sure that there is a Input Stack for all record fields available, even for those which are hidden e.g. creation date, etc.

Form - Format

- **Form Font, Label Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Transform: (select)**
  Select a text transformation
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?

- **Text: (color)**
  Change the text color by using the standard color swatch

Form - Layout

- **Label Width (input)**
  Define the label width in px
- **Field Width (input)**
  Define the field width in px
- **Field Height (input)**
  Define the field height in px
- **Field Gap (slider)**
  Define the gap between input rows in px
- **Label Above: (checkbox)**
  Move the label above the field?
- **Add Label Colon: (checkbox)**
  Automatically add a colon after each label name?
- **Add ‘Required’ Marker: (checkbox)**
  Automatically add an asterix after each label of a required field?

Form - Setup

- **Method (select)**
  Select the HTTP method used for that form. This is typically POST.
- **Encoding (select)**
  Select the HTTP method used for that form. This is typically POST.
  - *URL Encode*, i.e. the default encoding for normal form handling.
  - *Multipart*, i.e. the encoding you have to use while doing uploads handling.
  - *Text*, i.e. the encoding used for mail forms.
- **Locale (input)**
  The locale used during form validation, e.g. en, de, it, …
- **Input (input)**
  This is a list of field/parameter pairs that will be used to initialize certain fields with URL parameter values, e.g. comment_id=comment&blog_id=blog

Form - Action

- **Action Page: (link)**
  Set the action page that will be called after the form has been successfully processed, i.e. no validation errors occured
- **Cancel Page: (link)**
  Set the cancel page that will be called (if set) after the cancel button has been processed, i.e. form has been canceled
- **Transfer: (input)**
  A list of & separated URL parameters that will be transfered from the original URL *Example: order&comment_id*
- **Exclude Param (checkbox)**
  Should the transfer parameters be excluded from the original URL parameters

⇤ Component Index

6.24 Input Stack (Create, Update, Delete)

Overview

The Input Stack is the most generic form input you can use inside Form Stacks

Container: Form

Config & Settings

Input - Main

- **Label: (input)**
  Define the label name of the input field
- **Field: (input)**
  Define the input field name as used in the database.
- **Desc: (input)**
  Enter a placeholder that will be shown as long as there is no input yet available

Input - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Focus?: (color)**
  Change the focus color by using the standard color swatch

Input - Layout

- **Show Label: (checkbox)**
  Do you want to show the field label?
- **Label Width (number)**
  Define the label width in px
- **Field Width (number)**
  Define the field width in px
- **Field Height (number)**
  Define the field height in px

Input - Setup

- **Default (input)**
  Define the default value. You can use a constant as well as a set of predefined functions
    - **param:(name)** access a URL parameter
    - **session(name):** access a session parameter
    - **cookie(name):** access a cookie

    - **var(name):** access a variable
    - **globals(name):** access a global variable
    - **server(name):** access a server variable
    - **time():** retrieve the current date and time
    - **rand(max):** create a random number between 0 and max
    - **randhash(seed):** create a random hash code based on a seed
- **Tab Index: (number)**
  Index that defines the tabbing order of the element within the form or leave this empty. Set this to 0 if you want to exclude a form element.
- **Type?: (select)**
  Set the type of the input field accrding to the HTML5 spec. The behavior varies depending on the used browser. This makes the Date Input superflous as the functionalty is now fully included here.
- **Hash Input?: (checkbox)**
  Do you want hash input before storing it in the database?
- **Is Repeated Input (checkbox)**
  Is this a repeated input of another input?
- **Stack ID (input)**
  Specify the Stack ID as a back reference to the primary Input that should have the same value as this Input, e.g. stacks_in_101_page4.

⇤ **Component Index**

6.24 DateInput Stack (Create, Update, Delete)

Overview

The DateInput Stack is very similar to the standard Input Stack but knows about date formatting. Date Input is now deprecated as the Input Stack now supports date fields as well.

Container: Form

Config & Settings

DateInput - Main

- **Label: (input)**
  Define the label name of the input field
- **Field: (input)**
  Define the input field name as used in the database
- **Desc: (input)**
  Enter a placeholder that will be shown as long as there is no input yet available

DateInput - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Focus?: (color)**
  Change the focus color by using the standard color swatch

DateInput - Layout

- **Show Label: (checkbox)**
  Do you want to show the field label?
- **Label Width (number)**
  Define the label width in px
- **Field Width (number)**
  Define the field width in px
- **Field Height (number)**
  Define the field height in px

DateInput - Setup

- **Default (input)**
  Define the default value. You can use a constant as well as a set of predefined functions
  - **param:(name)** access a URL parameter
  - **session(name):** access a session parameter

    - **cookie(name):** access a cookie
    - **var(name):** access a variable
    - **globals(name):** access a global variable
    - **server(name):** access a server variable
    - **time():** retrieve the current date and time
- **Date Format: (input)**
  Use a standard PHP date format string to format a date input.
- **Use DatePicker?: (checkbox)**
  Do you want to use a jQuery UI based Datepicker? Let’s do it …
- **Picker Format: (input)**
  Use a standard jQuery UI Datepicker date format string to format a date input.
- **Theme: (input)**
  You can use your own theme for the Datepicker. Using the jQuery ThemeRoller you can create your own theme, upload it in ther RW resource section and enter the theme name here.

⇤ **Component Index**

6.26 CheckRadio Stack (Create, Update, Delete)

Overview

The CheckRadio Stack is used to input boolean values either as checkboxes or radios buttons. CheckRadio can handle multivalues as well.

Container: Form

Config & Settings

CheckRadio - Main

- **Label: (input)**
  Define the label name of the input field
- **Field: (input)**
  Define the input field name as used in the database

CheckRadio - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch

CheckRadio - Layout

- **Show Label: (checkbox)**
  Do you want to show the field label?
- **Label Width (number)**
  Define the label width in px

CheckRadio - Setup

- **Default (input)**
  Define the default value. You can use a constant as well as a set of predefined functions
    - **param:(name)** access a URL parameter
    - **session(name):** access a session parameter
    - **cookie(name):** access a cookie
    - **var(name):** access a variable
    - **globals(name):** access a global variable
    - **server(name):** access a server variable
    - **time():** retrieve the current date and time
- **Tab Index: (number)**
  Index that defines the tabbing order of the element within the form or leave this empty. Set this to 0 if you want to

exclude a form element.

- **Use Checkbox?: (checkbox)**
  Do you want to use checkboxes instead of radio buttons?
- **Multivalue?: (checkbox)**
  Allow multi values? This makes sense only with checkboxes.
- **Format Output: (input)**
  This is a format string that defines how the input will be rendered. The string contains two parts seperated by a | character. The first part defines how list elements will be rendered, whereas the second part is used for single values or the last element of a list. Use %s as a variable for the field content within the format string. *Example: %s, | %s renders a list of 1,2,3,4 as 1, 2, 3, 4*
- **Join: (select)**
  Joins are used to translate an actual field value into another value by using the actual field value as an index to find the new value. Joins can be either
  - *none*, i.e. the original value will be displayed.
  - *static*, i.e. based on a static mapping list containing index and value pairs.
    - **Key/Value List: (input)**
      A list of key,value pairs seperated by a | pipe. *Example: 1,foo|2,bar|3,foobar*
  - *dynamic*, i.e. based on another database table, an index field and a value field or
    - **Table: (input)**
      The name of the join table
    - **Index Field: (input)**
      The field used as an index into the join table
    - **Value Field: (input)**
      The field used as the mapping value from the join table

Note: you have to use a JOIN with CheckRadio inputs in order to create the various options even though you might not use the multivalue feature.

⇤ Component Index

6.27 Select Stack (Create, Update, Delete)

Overview

The CheckRadio Stack is used to input boolean values either as checkboxes or radios buttons. CheckRadio can handle multivalues as well.

Container: Form

Config & Settings

Select - Main

- **Label: (input)**
  Define the label name of the input field
- **Field: (input)**
  Define the input field name as used in the database

Select - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch

Select - Layout

- **Show Label: (checkbox)**
  Do you want to show the field label?
- **Label Width (number)**
  Define the label width in px
- **Field Width (number)**
  Define the input field width in px
- **Select Size: (number)**
  Define the number of items that are shown in the select

Select - Setup

- **Default (input)**
  Define the default value. You can use a constant as well as a set of predefined functions
    - **param:(name)** access a URL parameter
    - **session(name):** access a session parameter
    - **cookie(name):** access a cookie
    - **var(name):** access a variable
    - **globals(name):** access a global variable

    - **server(name):** access a server variable
    - **time():** retrieve the current date and time

- **Tab Index: (number)**
  Index that defines the tabbing order of the element within the form or leave this empty. Set this to 0 if you want to exclude a form element.
- **Multivalue?: (checkbox)**
  Allow multi values?
- **Format Output: (input)**
  This is a format string that defines how the input will be rendered. The string contains two parts seperated by a | character. The first part defines how list elements will be rendered, whereas the second part is used for single values or the last element of a list. Use %s as a variable for the field content within the format string. *Example: %s, | %s renders a list of 1,2,3,4 as 1, 2, 3, 4*
- **Join: (select)**
  Joins are used to translate an actual field value into another value by using the actual field value as an index to find the new value. Joins can be either
    - *none*, i.e. the original value will be displayed.
    - *static*, i.e. based on a static mapping list containing index and value pairs.
        - **Key/Value List: (input)**
          A list of key,value pairs seperated by a | pipe. *Example: 1,foo|2,bar|3,foobar*
    - *dynamic*, i.e. based on another database table, an index field and a value field or
        - **Table: (input)**
          The name of the join table
        - **Index Field: (input)**
          The field used as an index into the join table
        - **Value Field: (input)**
          The field used as the mapping value from the join table

Note: you have to use a JOIN with Select inputs in order to create the various options even though you might not use the multivalue feature.

⇤ Component Index

6.28 TextArea Stack (Create, Update, Delete)

Overview

The CheckRadio Stack is used to input boolean values either as checkboxes or radios buttons. CheckRadio can handle multivalues as well.

Container: Form

Config & Settings

TextArea - Main

- **Label: (input)**
  Define the label name of the input field
- **Field: (input)**
  Define the input field name as used in the database

TextArea - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch

TextArea - Layout

- **Show Label: (checkbox)**
  Do you want to show the field label?
- **Label Width (number)**
  Define the label width in px
- **Width (number)**
  Define the text area width in chars
- **Height (number)**
  Define the text area height in chars

TextArea - Setup

- **Default (input)**
  Define the default value. You can use a constant as well as a set of predefined functions
    - **param:(name)** access a URL parameter
    - **session(name):** access a session parameter
    - **cookie(name):** access a cookie
    - **var(name):** access a variable
    - **globals(name):** access a global variable

    - **server(name):** access a server variable
    - **time():** retrieve the current date and time

- **Tab Index: (number)**
  Index that defines the tabbing order of the element within the form or leave this empty. Set this to 0 if you want to exclude a form element.
- **Use Richtext Editor?: (checkbox)**
  Do you want to use a richtext editor?
- **Editor Type?: (checkbox)**
  Select the used editor type, i.e. *simple, advanced, full*?
- **Image Browser: (link)**
  Provide the link to the page that is the image browser for the TextArea Stack.

⇤ **Component Index**

6.29 Upload Stack (Create, Update, Delete)

Overview

The Upload Stack is used to upload files to the server. Make sure you set the form encoding methode to Multipart in this case.

Container: Form

Config & Settings

Upload - Main

- **Label: (input)**
  Define the label name of the input field
- **Field: (input)**
  Define the input field name as used in the database

Upload - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch

Upload - Layout

- **Show Label: (checkbox)**
  Do you want to show the field label?
- **Label Width (number)**
  Define the label width in px
- **Field Width (number)**
  Define the field width in px

Upload - Setup

- **Filetype: (checkbox)**
  Set the allowed filetypes using comma separated list of mime types e.g. image/jpeg, image/png. Leave this empty if there is no restriction.
- **Max Size (number)**
  Define the max upload size in KB. Leave this empty if there is no maximum defined.
- **Max Width (number)**
  Define the max image width in px. Leave this empty if there is no maximum defined.
- **Max Height (number)**
  Define the max image height in px. Leave this empty if there is no maximum defined.
- **Upload Dir: (link)**

Select the image dir to where the files will be uploaded to? Don’t use remote servers here, just local directories.

- **Multiple Uploads: (checkbox)**
  Enable multiple file uploads? Files will be stored as a multivalue list inside the field. Male sure that there is enough space available.
- **Max Uploads (number)**
  Limit the number of uploads in case of multiple uploads. Leave this empty if there is no maximum defined.
- **Unique Name: (checkbox)**
  Generate unique file names?
- **Store Path?: (checkbox)**
  Store the full path instead of the filename only?

⇤ **Component Index**

6.30 Hidden Stack (Create, Update, Delete)

Overview

The Hidden Stack is used to input form value without having a visible input field.

Container: Form

Config & Settings

Hidden - Main

- **Field: (input)**
  Define the input field name as used in the database

Hidden - Setup

- **Default (input)**
  Define the default value. You can use a constant as well as a set of predefined functions
  - **param:(name)** access a URL parameter
  - **session(name):** access a session parameter
  - **cookie(name):** access a cookie
  - **var(name):** access a variable
  - **globals(name):** access a global variable
  - **server(name):** access a server variable
  - **time():** retrieve the current date and time
- **Force Default?: (checkbox)**
  Force the default value even for updated records? This could be used to overwrite a field in a form.
- **Date Format: (input)**
  Use a standard PHP date format string to format a date input.

⇤ **Component Index**

6.31 Captcha Stack (Create, Update, Delete)

Overview

The Captcha Stack creates a CATCHA that you can use inside Form Stacks

Container: Form

Config & Settings

Captcha - Main

- **Label: (input)**
  Define the label name of the input field
- **Field: (input)**
  Define the input field which is actually not in the database. This is used internally only.
- **Desc: (input)**
  Enter a placeholder that will be shown as long as there is no input yet available

Captcha - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Color?: (checkbox)**
  Do you want to change the colors?
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Focus?: (color)**
  Change the focus color by using the standard color swatch

Captcha - Layout

- **Show Label: (checkbox)**
  Do you want to show the field label?
- **Label Width (number)**
  Define the label width in px
- **Field Width (number)**
  Define the field width in px
- **Field Height (number)**
  Define the field height in px
- **Float? (checkbox)**
  Float the image and the input field

Captcha - Setup

- **Tab Index: (number)**
  Index that defines the tabbing order of the element within the form or leave this empty. Set this to 0 if you want to exclude a form element.

- **Width (number)**
  Define the CAPTCHA width in px
- **Height (number)**
  Define the CAPTCHA height in px
- **# Chars (number)**
  Define the number of used chars
- **# Lines (number)**
  Define the number of background lines
- **Use Color?: (checkbox)**
  Use colors for the background lines?

⇤ **Component Index**

6.32 ButtonSet Stack (Create, Update, Delete)

Overview

The ButtonSet Stack generates a variety of buttons for the Form and the Login Stack. Just add the stack to the respective Form and Layout Stack and you are done.

Container: Form

Config & Settings

ButtonSet - Main

- **Insert, Update, Delete, Cancel, Login, Logout, Search: (input)**
  Define the label name of the button. Remove buttons by leaving a label empty. This way you can remove the 'Delete' button for instance.

ButtonSet - Format

- **Font: (select)**
  Select the render font from the list of predefined fonts, choose Inherit to use the themes font or select Custom to use a custom font. Make sure that whatever font is used here is available on your target system.
- **Size: (select)**
  Choose the font size from a set of predefined sizes
- **Custom Size: (input)**
  Define the required text size using the usual CSS size units eg. 15px/150%. Leave this input empty to inherit the font size from the surrounding container or the theme.
- **Transform: (select)**
  Select a text transformation
- **Bold?, Italic?: (checkbox)**
  Change the text format by checking bold and/or italic
- **Style: (select)**
  Select a text transformation
  - *Default*, i.e. the default button style.
  - *Class*, i.e. define a custom class
  - *Custom*, i.e. define the button style
- **Text: (color)**
  Change the text color by using the standard color swatch
- **Background: (color)**
  Change the text color by using the standard color swatch
- **Border: (color)**
  Change the border color by using the standard color swatch
- **Style: (select)** Change the border style using the standard CSS values
- **Width: (slider)**
  Define button border width in px
- **Radius: (slider)**
  Define button border radius in px
- **Padding LR, Padding TB: (slider)**
  Specify the padding of the search button left, right and top, botton in px

ButtonSet - Layout

- **Width: (input)**
  Define button width in px
- **Gap: (slider)**

Specify the gap between the search field and the button in px

- **Align: (select)**
  Select a button alignment
- **Float (checkbox)**
  Do you want to float the bottons?

ButtonSet - Setup

- **Type: (select)**
  Select the required button types
- **Tab Index: (number)**
  Index that defines the tabbing order of the element within the form or leave this empty. Set this to 0 if you want to exclude a form element.

⇤ Component Index

6.33 Script Stack (Code)

Overview

The Script Stack allows you to add custom code to your app. Place it in layout where you want the code to be executed. Within a script you can access the context in which the script is executed via the $context vaiable. This is either a Record or a Form. Scripts provide a couple of custom settings that can be used to customize a script. See also Script - Custom. To access or store values on a global scope access variables via $GLOBALS['varname']. Scripts are triggered by certain events that relate to Record and Forms operations.

Container: Top Level

Config & Settings

Script - Main

- **Text Edit: (input)**
  Add the custom script code

Script - Setup

- **Code Trigger: (select)**
  Select the event that should trigger the respective piece of code. Those events are related to record and form operations.
- **Code Use 'nowdoc' PHP >= 5.3: (checkbox)**
  If you are using PHP >= 5.3 on your environment, you can leverage a feature that is only available on those environments and gives you more freedom in regards to what's being allowed in side a Script Stack. Using the nowdoc systax for strings, you can use all characters including single and double quotes. If you on a PHP environment which doesn't support nowdoc syntax, you can not use single quotes inside scripts.
- **Debug: (checkbox)**
  Scripts output some debug info during invocation.

Script - Custom

- **Custom Values?: (checkbox)**
  Enable custom value settings that can be used inside your custom code
- **Input1–3: (input)**
  Use 3 different custom inputs inside your code. Those fields can be referenced by $customInput1, …, $customInput3
- **Color1–2: (color)**
  Use 2 different custom colors inside your code. Those fields can be referenced by $customColor1, $customColor2
- **Link: (link)**
  Use a custom link inside your code. This field can be referenced by $customLink1

⇤ Component Index

6.34 If Then Else Stack (Code)

Overview

The IfThenElse Stack allows you to display/execute stacks depending on some custom code expression.

Container: Top Level

Config & Settings

IfThenElse - Content Drop Zone

The two drop zones hold the respective stacks/content that will be shown if your expression evaluates to true or false.

⇤ Component Index

6.35 CSS Stack (Layout)

Overview

The CSS Stack generates a inline CSS section that can be used to customize the page layout.

Container: Top Level

Config & Settings

CSS - Main

- **Text Edit: (input)**
  Add the custom CSS code

⇤ Component Index

6.36 Tag Stack (Layout)

Overview

The Tag Stack generates can be used to generate arbitrary HTML code with begin and end tags. Use this stack to create DIVs with custom class names or anything you like.

Container: Any

Config & Settings

CSS - Main

Add the custom code in front or after the drop zone. Male sure you balance tags right.

⇤ Component Index

6.37 Markdown Stack (Layout)

Overview

The Markdown Stack converts a Markdown text into normal HTML section.

Container: Top Level

Config & Settings

Markdown - Main

- **Text Edit: (input)**
  Add the custom Markdown code

⇤ Component Index

6.38 Floater Stack (Layout)

Overview

The Floater Stack allows you float its containing stack left or right in order to create fluid layouts

Container: Top Level

Config & Settings

Floater

- **Float: (select)**
  Choose to float the containing stack either *left or right*.

⇤ Component Index

6.39 FieldSet Stack (Layout)

Overview

The FieldSet Stack generates a highly customizable HTML fieldset

Container: Top Level

Config & Settings

FieldSet - Content Drop Zone

Add the FieldSets containing content and change the FieldSet label

Fieldset

- **Highlight Legend: (checkbox)**
  Highlight the fieldset legend
- **Padding LR, Padding TB: (slider)**
  Specify the LR and TB padding of the fieldset
- **Background Color?: (checkbox)**
  Do you want to change the fieldsets background color?
- **Background: (color)**
  Change the fieldsets background color
- **Border: (color)**
  Change the fieldsets border color
- **Width: (slider)**
  Change the fieldsets border width in px
- **Radius: (slider)**
  Change the fieldsets border radius in px
- **Shadow: (slider)**
  Change the fieldsets shadow size in px
- **Blur: (slider)**
  Change the fieldsets blur size in px

⇤ Component Index

6.40 Hauler Stack (Layout)

Overview

The Hauler Stack allows other Stacks to target page areas like the header, sidebar, footer, ExtraContent areas or any custom area that can be addressed via a CSS selector. You can even decide to put content before or after the respective area. If your template supports ExtraContent, you can target those areas by selecting the specific EC area. In addition to assign a static ExtraContent area, the CMS Stack supports dynamic assignment, i.e. you can specify in your content database which ExtraContent area is targeted by a specific content. Check the CMS Stack for more details.

Container: Top Level

Config & Settings

Hauler - Content Drop Zone

Add the Stack that should be moved to a different area

Hauler

- **Target: (select)**
  Select the page area you want to target with the content?
  - *Header*, i.e. the header area
    - **Position: (select)**
      Specify position relative to the target, i.e. Before or After
  - *Sidebar*, i.e. the sidebar area
    - **Position: (select)**
      Specify position relative to the target, i.e. Before or After
  - *Content*, i.e. the content area
  - *Footer*, i.e. the footer area
    - **Position: (select)**
      Specify position relative to the target, i.e. Before or After
  - *ExtraContent*, i.e. the content is rendered in an ExtraContent area
    - **EC Area: (number)**
      Specify the ExtraContent area
  - *Selector*, i.e. address a custom area via a CSS selector
    - **Position: (select)**
      Specify position relative to the target, i.e. Before or After
    - **Selector: (input)**
      Specify the target via a CSS selector, i.e. #stacks_in_26_page9

⇤ **Component Index**

7. SDK Reference

7.1 Record

7.2 Field

7.3 Form

7.4 Input

⇤ TOC

8. Licenses

8.1 StackApps

As of right now, StackApps is free to use but not yet open source. Even though I intend to move StackApps under one of the major open source licenses I consider StackApps to be under my full control for the time being.

Please keep in mind that i am not responsible for any damages and liabilities caused by apps generated by the StackApps framework.

StackApps uses a couple of 3rd party libraries and frameworks that are protected by their own license agreements. StackApps is referring to those license agreements in case of any questions. Please make sure that you comply with those licenses in case of commercially using any code generated by StackApps.

8.2 ADODB

ADODB is used as the underlying abstraction layer to access the used database. Check the respective license.

8.3 CKFinder

CKEditor can be used as a richtext editor as a replacement for a textarea input. Check the respective license.

8.4 PHP Markdown Extra

PHP Markdown is a PHP library used to format content based on the Markdown markup language, introduced by John Gruber. Check the respective license.

8.5 php-captcha

php-captcha is a PHP library used to generate and verify simple captchas, written by Edward Eliot. Check the respective license.

8.6 Font Awesome

The iconic font designed for use with Twitter Bootstrap. Check the respective license.

8.9 Google Web Fonts

Free web fonts provided by Google. Check the respective license.

8.10 jQuery.qrcode

A jQuery plugin to generate a QR code. Check the respective license.

8.11 jQuery UI

Used to provide a UI DatePicker. Check the respective license.

8.12 Simple HTML DOM

A PHP library to parse, search and modify an HTML5 DOM. Used to manipulate the site navbar. Check the respective license.

8.13 FastImage.php

A PHP library to determine the size of an bitmap image. Doesn't require any image library like GD. Check the respective license.

8.14 Slimbox 2

An ultraslim Lightbox clone. Check the respective license.

8.15 ColorBox

A lightweight customizable lightbox plugin for jQuery. Check the respective license.

8.16 TimThumb

A powerful server-side image scaler. Check the respective license.

8.17 c5 Filemanager

A flexible, powerful filemanager used by CKEditor. Check the respective license.

⇤ TOC

9. History

StackApps has quiet a long history: inspired by RAD tools like **CodeCharge**, I had this app development framework already developed in 2006/2007 as a spare time project using **NetObjects Fusion** as a host environment, but never released this project to the public. At time it was called ConFusion :)

Being written in Java as a Fusion plug-in initially, I decided to move this project to **RapidWeaver** in 2011, my new web development platform on the Mac.

At the beginning I tried to implement **StackApps** as a native RapidWeaver plug-in using Objective-C, but soon I realized that this was too much of an effort, as there was no good component layout engine available in the SDK.

As **YourHead Stacks** for RapidWeaver became more and more popular as the unofficial plug-in API for RapidWeaver, I decided to start the project using Stacks as the development environment. Stacks is not only much simpler to use compared to the official RapidWeaver plug-in API, it also offers a flexible and powerful layout engine & component model which perfectly suited my requirements.

December 2010 I started with the development and finally finished the first internal release around August 2011. At the time it became obvious that Stacks was about to get its next major version update which fixed a lot of the initial UI limitations of Stacks. Participating in the early beta I soon decided to migrate **StackApps** to **Stacks 2.0** first before releasing it to the public.

With Stacks 2.0 now being released to the public it's about time to release **StackApps** as well, being the first full-fledged application development framework being available for Stacks.

StackApps is currently available at (http://stackapps.designdisorder.com/download).

⇤ TOC

10. About Me

⇤ TOC