TYPO3 CMS 8.0 – What's New

Summary of the new features, changes and improvements

Created by:
Patrick Lobacher and Michael Schams

29/March/2016

TYPO3

TYPO3 CMS 8.0 – What's New

Chapter Overview

Introduction

Backend User Interface

TSconfig & TypoScript

In-Depth Changes

Extbase & Fluid

Deprecated/Removed Functions

Sources and Authors

Introduction

The Facts

Introduction

TYPO3 CMS 8.0 – The Facts

- Release date: 22 March 2016
- Release type: Sprint Release
- Slogan: Start your engines

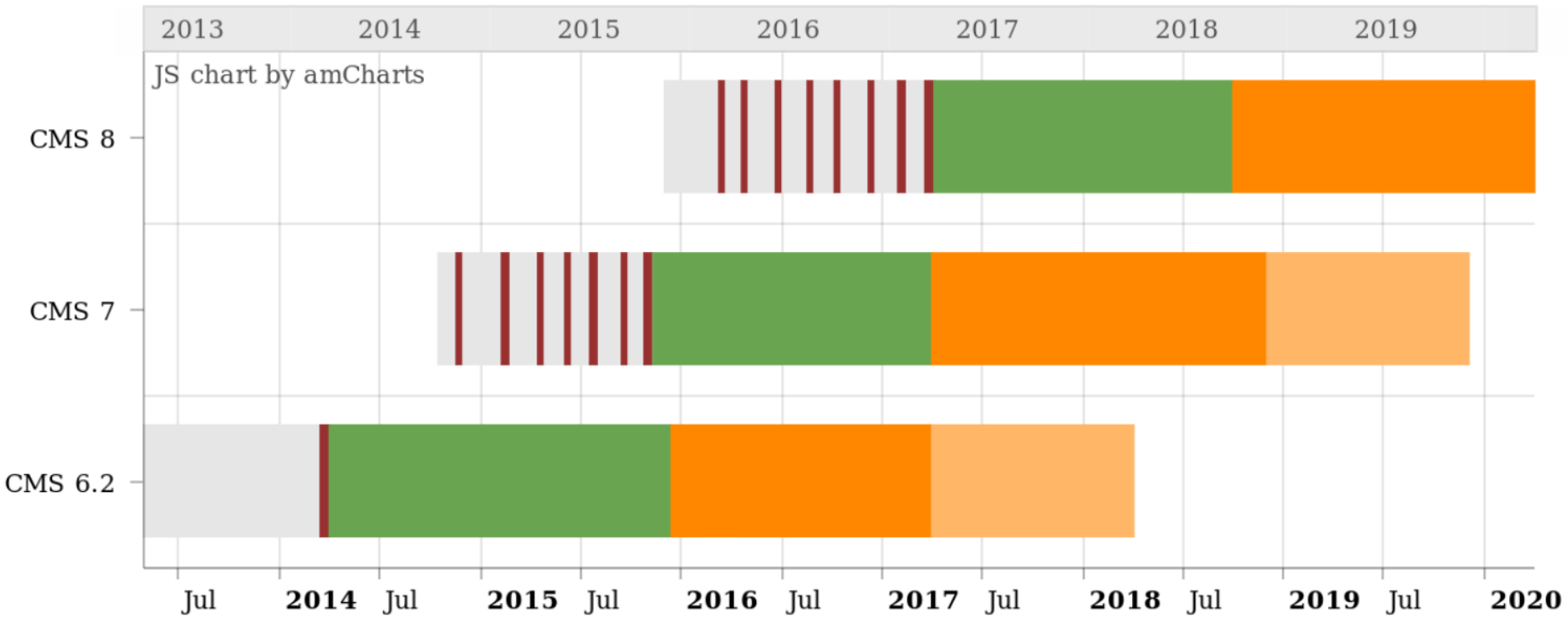
Introduction

System Requirements

- PHP: version 7
- MySQL: version 5.5 to 5.7
- Disk space: min 200 MB
- PHP settings:
  - `memory_limit` >= 128M
  - `max_execution_time` >= 240s
  - `max_input_vars` >= 1500
  - compilation option `--disable-ipv6` must not be used
- The backend requires Microsoft Internet Explorer 11 or later, Microsoft Edge, Google Chrome, Firefox, Safari or any other modern, compatible browser

Introduction

Development and Release Timeline



Introduction

TYPO3 CMS Roadmap

Release dates and their primary focus:

- v8.0 22/Mar/2016 Adding last minute things
- v8.1 03/May/2016 Cloud Integration
- v8.2 05/Jul/2016 Rich Text Editor
- v8.3 30/Aug/2016 Frontend Editing on Steroids
- v8.4 18/Oct/2016 *to be determined*
- v8.5 20/Dec/2016 Integrator Support
- v8.6 14/Feb/2017 *to be determined*
- v8.7 04/Apr/2017 LTS Preparation

https://typo3.org/typo3-cms/roadmap/

https://typo3.org/news/article/kicking-off-typo3-v8-development/

Introduction

Installation

- Official installation procedure under Linux/Mac OS X (DocumentRoot for example `/var/www/site/htdocs`):

```
$ cd /var/www/site
$ wget --content-disposition get.typo3.org/8.0
$ tar xzf typo3_src-8.0.0.tar.gz
$ cd htdocs
$ ln -s ../typo3_src-8.0.0 typo3_src
$ ln -s typo3_src/index.php
$ ln -s typo3_src/typo3
$ touch FIRST_INSTALL
```

- Symbolic links under Microsoft Windows:
  - Use `junction` under Windows XP/2000
  - Use `mklink` under Windows Vista and Windows 7

Introduction

Upgrade to TYPO3 CMS 8.x

- Upgrades only possible from TYPO3 CMS 7.6 LTS
- TYPO3 CMS < 7.6 LTS should be updated to TYPO3 CMS 7.6 LTS first
- Upgrade instructions:

  http://wiki.typo3.org/Upgrade#Upgrading_to_8.0
- Official TYPO3 guide "TYPO3 Installation and Upgrading":

  http://docs.typo3.org/typo3cms/InstallationGuide
- General approach:
  - Check minimum system requirements (PHP, MySQL, etc.)
  - Review **deprecation_*.log** in old TYPO3 instance
  - Update all extensions to the latest version
  - Deploy new sources and run Install Tool -> Upgrade Wizard
  - Review startup module for backend users (optionally)

Introduction

PHP Version 7

- PHP 7.0 is the minimum requirement for TYPO3 CMS 8.x
- TYPO3 will support subsequent PHP 7 releases as they come out
- This version raise gives a significant performance boost to the overall system
- Not only backend editors will notice a more fluent interface, but the new all-time record for a full cached page call in the frontend is below 7 milliseconds now, which is approximately 40% faster compared to running the very same website with PHP version 5.5
- We also started using new features from this PHP version, for instance the cryptographically secure pseudo-random generators are in active use already

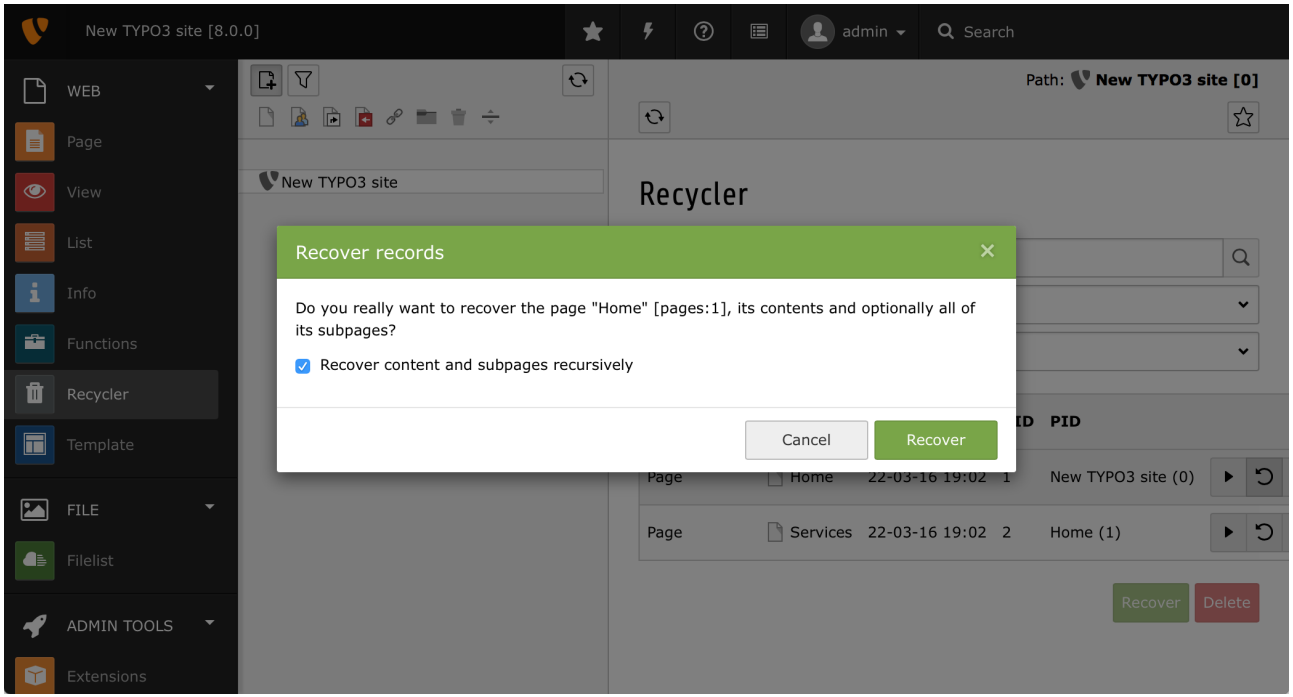
Chapter 1:

Backend User Interface

Backend User Interface

Recover pages recursively to top of rootline

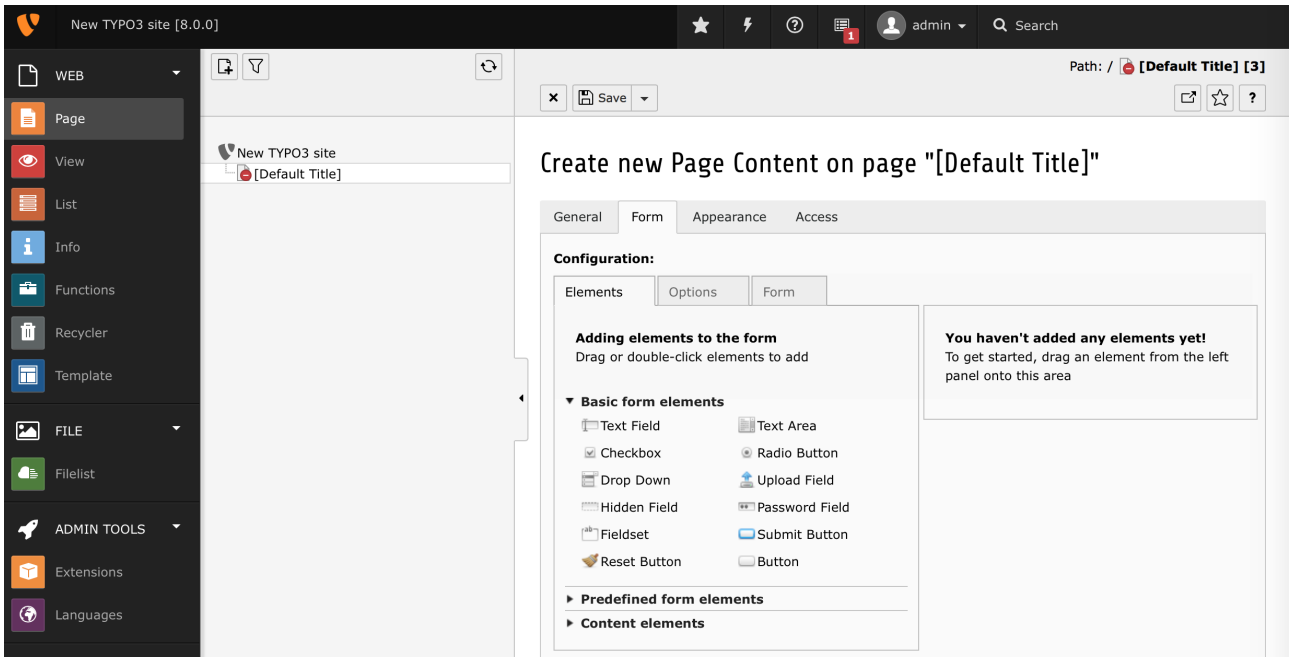
The Recycler supports the recursive recovery of deleted pages to the top of the rootline now. This feature is available for admin users only due to internal permission restrictions.



Backend User Interface

Directly load form wizard as inline wizard

The wizard of EXT:form is loaded directly as inline wizard. There is no need to save and reload the newly created content element anymore in order to be able to open the wizard. This is a huge usability improvement.



Backend User Interface

Set an alternative backend logo via Extension Manager

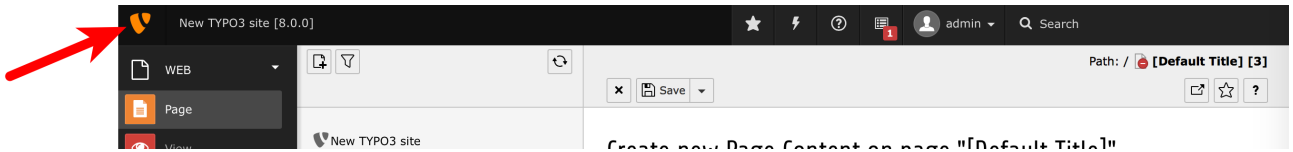
The backend logo in the upper left corner can now be configured in the extension configuration of EXT:backend in the Extension Manager. Configuration options are:

- resource as a relative path of the TYPO3 installation

  e.g. "`fileadmin/images/my-background.jpg`"
- path to an extension

  e.g. "`EXT:my_theme/Resources/Public/Images/my-background.jpg`"
- an external resource

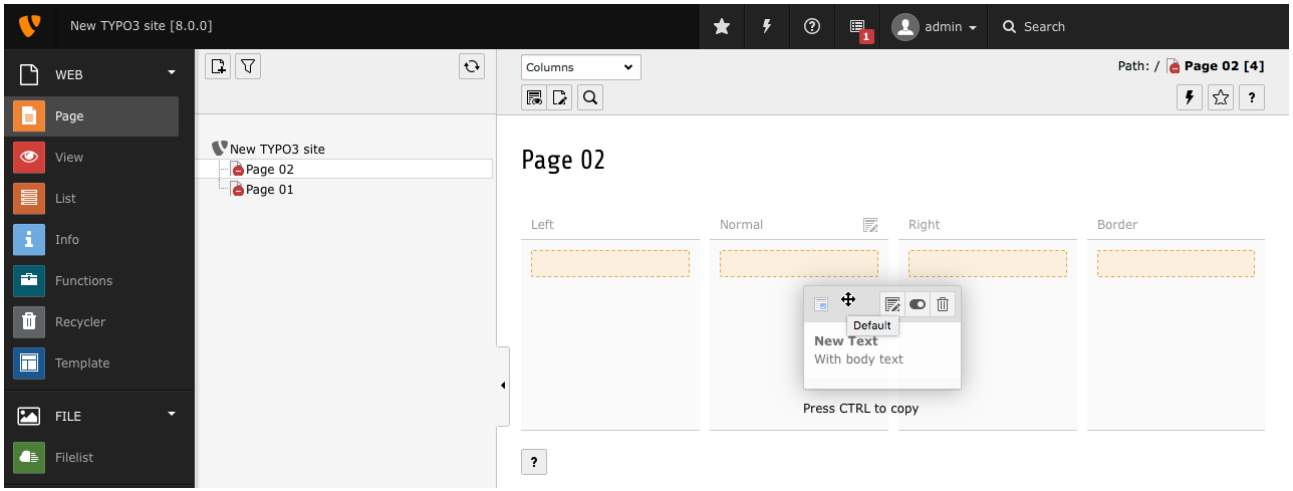
  e.g. "`//example.com/my-background.png`"



Backend User Interface

Copy pages in drag & drop mode

Additionally to the usual drag and drop feature in the page module (that *moved* content elements), it is now possible to create copies: press the CTRL key while dropping to create a copy of the dragged element. After dropping is complete, the page module will reload to make sure the new element will be generated with all necessary information.



Chapter 2:

TSconfig & TypoScript

TSconfig & TypoScript

Sort order of new content element wizard tab

- It is possible to configure the order of the tabs in the new content element wizard by setting `before` and `after` values in Page TSconfig:

```
mod.wizards.newContentElement.wizardItems.special.before = common
mod.wizards.newContentElement.wizardItems.forms.after = common,special
```

TSconfig & TypoScript

HTMLparser.stripEmptyTags.keepTags

- New option for the HTMLparser.stripEmptyTags configuration has been added, which allows for keeping configured tags
- Before this change, only a list of tags could be provided that should be removed
- The following example strips all empty tags **except** tr and td tags:

```
HTMLparser.stripEmptyTags = 1
HTMLparser.stripEmptyTags.keepTags = tr,td
```

Important: if this setting is used, configuration stripEmptyTags.tags has no effect anymore. You can only use one option at a time.

TSconfig & TypoScript

EXT:form - integration of predefined forms (1)

- The content element of EXT:form now allows the integration of predefined forms.
- An integrator can define forms (e.g. within a site package) using plugin.tx_form.predefinedForms
- An editor can add a new mailform content element to a page and choose a form from a list of predefined elements
- Integrators can build their forms with TypoScript, which provide much more options than doing it within the form wizard (e.g. integrators can use stdWrap functionality, which is not available when using the form wizard (for security reasons)

TSconfig & TypoScript

EXT:form - integration of predefined forms (2)

- There is no need for editors to use the form wizard anymore. Editors can choose the predefined forms which are optimized layout-wise.
- Forms can be re-used throughout the whole installation
- Forms can be stored outside the DB and versioned
- In order to be able to select the pre-defined form in the backend, the form has to be registered using PageTS:

```
TCEFORM.tt_content.tx_form_predefinedform.addItems.contactForm =
  LLL:EXT:my_theme/Resources/Private/Language/locallang.xlf:contactForm
```

TSconfig & TypoScript

EXT:form: integration of predefined forms (3)

- Example form:

```
plugin.tx_form.predefinedForms.contactForm = FORM
plugin.tx_form.predefinedForms.contactForm {
  enctype = multipart/form-data
  method = post
  prefix = contact
  confirmation = 1
  postProcessor {
    1 = mail
    1 {
      recipientEmail = test@example.com
      senderEmail = test@example.com
      subject {
        value = Contact form
        lang.de = Kontakt Formular
      }
    }
  }
  10 = TEXTLINE
  10 {
    name = name
...
```

Chapter 3:

In-Depth Changes

In-Depth Changes

Support PECL-memcached in MemcachedBackend

- Support for the PECL module "memcached" has been added to the MemcachedBackend of the Caching Framework
- If both, "memcache" and "memcached" are installed, "memcache" is used to avoid being a breaking change.
- An integrator may set the option `peclModule` to use the preferred PECL module:

```
$GLOBALS['TYPO3_CONF_VARS']['SYS']['caching']['cacheConfigurations']['my_memcached'] = [
  'frontend' => \TYPO3\CMS\Core\Cache\Frontend\VariableFrontend::class
  'backend' => \TYPO3\CMS\Core\Cache\Backend\MemcachedBackend::class,
  'options' => [
    'peclModule' => 'memcached',
    'servers' => [
      'localhost',
      'server2:port'
    ]
  ]
];
```

In-Depth Changes

Native support for Symfony Console (1)

- TYPO3 supports the Symfony Console component out-of-the-box now by providing a new command line script located in `typo3/sysext/core/bin/typo3`. On TYPO3 instances installed via Composer, the binary is linked into the `bin`-directory, e.g. `bin/typo3`.
- The new binary still supports the existing command line arguments when no proper Symfony Console command was found as a fallback.
- Registering a command to be available via the `typo3` command line tool works by putting a `Configuration/Commands.php` file into any installed extension. This lists the Symfony/Console/Command classes to be executed by `typo3` is an associative array. The key is the name of the command to be called as the first argument to `typo3`.

In-Depth Changes

Native support for Symfony Console (2)

- A required parameter when registering a command is the `class` property. Optionally the `user` parameter can be set so a backend user is logged in when calling the command.
- A `Configuration/Commands.php` could look like this:

```
return [
  'backend:lock' => [
    'class' => \TYPO3\CMS\Backend\Command\LockBackendCommand::class
  ],
  'referenceindex:update' => [
    'class' => \TYPO3\CMS\Backend\Command\ReferenceIndexUpdateCommand::class,
    'user' => '_cli_lowlevel'
  ]
];
```

In-Depth Changes

Native support for Symfony Console (3)

- An example call could look like:

```
bin/typo3 backend:lock http://example.com/maintenance.html
```

- For a non-Composer installation:

```
typo3/sysext/core/bin/typo3 backend:lock http://example.com/maintenance.html
```

In-Depth Changes

Cryptographically secure pseudorandom number generator

- A new cryptographically secure pseudo-random number generator (CSPRNG) has been implemented in the TYPO3 core.
  It takes advantage of the new CSPRNG functions in PHP 7.
- The API resides in the class `\TYPO3\CMS\Core\Crypto\Random`
- Example:

```
use \TYPO3\CMS\Core\Crypto\Random;
use \TYPO3\CMS\Core\Utility\GeneralUtility;

// Retrieving random bytes
$someRandomString = GeneralUtility::makeInstance(Random::class)->generateRandomBytes(64);

// Rolling the dice..
$tossedValue = GeneralUtility::makeInstance(Random::class)->generateRandomInteger(1, 6);
```

In-Depth Changes

Wizard component (1)

- A new wizard component has been added. This component may be used for user-guided interactions
- The RequireJS module can be used by including `TYPO3/CMS/Backend/Wizard`
- The wizard supports straight forward actions only (junctions are not possible yet)
- The API resides in class `\TYPO3\CMS\Core\Crypto\Random`
- The wizard component has the following public methods:

```
addSlide(identifier, title, content, severity, callback)
addFinalProcessingSlide(callback)
set(key, value)
show()
dismiss()
getComponent()
lockNextStep()
unlockNextStep()
```

In-Depth Changes

Wizard component (2)

- The event `wizard-visible` is fired when the wizard rendering has finished
- Wizards can be closed by firing the `wizard-dismiss` event
- Wizards fire the `wizard-dismissed` event if the wizard is closed
- You can integrate your own listener by using `Wizard.getComponent()`

In-Depth Changes

Generated asset files moved

- The folder structure within `typo3temp` changed to separate assets that need to be accessed by the client from the files that are temporary created for (e.g. for caching or locking purposes and require server-side access only).
- These assets were moved from folders:
  `_processed_`, `compressor`, `GB`, `temp`, `Language`, `pics`
  and re-organized into:
  - `typo3temp/assets/js/`
  - `typo3temp/assets/css/`,
  - `typo3temp/assets/compressed/`
  - `typo3temp/assets/images/`

In-Depth Changes

ImageMagick/GraphicsMagick changes (1)

- Graphics processor settings for Image- or GraphicsMagick have been renamed (file: `LocalConfiguration.php`).
  OLD: `im_`
  NEW: `processor_`
- Negative naming such as `noScaleUp` have been changed to positive counterparts. During the conversion, the previous configuration values are negated to reflect the changes in semantics of these options.
- In addition, references to specific versions of ImageMagick/GraphicsMagick have been removed from settings names and values.

In-Depth Changes

ImageMagick/GraphicsMagick changes (2)

- The unused configuration option `image_processing` has been removed without replacement
- The processor-specific configuration option `colorspace` has been *namespaced* below the `processor_` hierarchy

In-Depth Changes

Hooks and Signals (1)

- An additional hook has been added to method `BackendUtility::viewOnClick()` to post-process the preview URL
- Register a hook class which implements the method with the name `postProcess`:

```
$GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS']['t3lib/class.t3lib_befunc.php']['viewOnClickClass'][] =
  \VENDOR\MyExt\Hooks\BackendUtilityHook::class;
```

In-Depth Changes

Hooks and Signals (2)

- Prior TYPO3 CMS 7.6, it was possible to override a record overlay in `Web -> List`. A new hook in TYPO3 CMS 8.0 provides the old functionality.
- The hook is called with the following signature:

```
/**
 * @param string $table
 * @param array $row
 * @param array $status
 * @param string $iconName
 * @return string the new (or given) $iconName
 */
function postOverlayPriorityLookup($table, array $row, array $status, $iconName) { ... }
```

- Register the hook class which implements the method with the name `postOverlayPriorityLookup`:

```
$GLOBALS['TYPO3_CONF_VARS']['SC_OPTIONS'][IconFactory::class]['overrideIconOverlay'][] =
  \VENDOR\MyExt\Hooks\IconFactoryHook::class;
```

In-Depth Changes

Hooks and Signals (3)

- A new signal has been implemented before a resource storage is initialized.
- Register the class which implements your logic in `ext_localconf.php`:

```
$dispatcher = \TYPO3\CMS\Core\Utility\GeneralUtility::makeInstance(
  \TYPO3\CMS\Extbase\SignalSlot\Dispatcher::class);
$dispatcher->connect(
  \TYPO3\CMS\Core\Resource\ResourceFactory::class,
  ResourceFactoryInterface::SIGNAL_PreProcessStorage,
  \MY\ExtKey\Slots\ResourceFactorySlot::class,
  'preProcessStorage'
);
```

- The method is called with the following arguments:
  - int `$uid` the uid of the record
  - array `$recordData` all record data as array
  - string `$fileIdentifier` the file identifier

In-Depth Changes

Password hashing algorithm: PBKDF2

- A new password hashing algorithm "PBKDF2" has been added to the system extension "saltedpasswords"
- PBKDF2 stands for: Password-Based Key Derivation Function 2
- The algorithm is designed to be computationally expensive to resist brute force password cracking

Chapter 4:

Extbase & Fluid

Extbase & Fluid

Standalone revised Fluid

- The Fluid rendering engine of TYPO3 CMS is replaced by the standalone capable Fluid which is now included as a composer dependency
- The old Fluid extension is converted to a so-called *Fluid adapter* which allows TYPO3 CMS to use standalone Fluid
- New features/capabilities have been added in nearly all areas of Fluid
- Most importantly: several of the Fluid components which were completely internal and impossible to replace in the past, are now easy to replace and have been fitted with a public API

Extbase & Fluid

RenderingContext (1)

- The most important new piece of public API is the RenderingContext
- The previously internal-only RenderingContext used by Fluid has been expanded to be responsible for a vital new Fluid feature: **implementation provisioning**
- This enables developers to change a range of classes, Fluid uses for parsing, resolving, caching etc.
- This can be achieved by either including a custom RenderingContext or manipulating the default RenderingContext by public methods.

Extbase & Fluid

Rendering Context (2)

- The following behaviours can all be controlled by manipulating the RenderingContext. By default, none of them are enabled - but calling a simple method (via your View instance) allows you to enable them:

```
$view->getRenderingContext()->setLegacyMode(false);
```

Extbase & Fluid

ExpressionNodes (1)

- ExpressionNodes are a new type of Fluid syntax structures which all share a common trait: they only work inside the curly braces

```
$view->getRenderingContext()->setExpressionNodeTypes(array(
    'Class\Number\One',
    'Class\Number\Two'
));
```

- Developers can add their own additional ExpressionNode types
- Each one consists of a pattern to be matched and methods dictated by an interface to process the matches
- Any existing ExpressionNode type can be used as reference

Extbase & Fluid

ExpressionNodes (2)

ExpressionNodeTypes allow new syntaxes such as:

- **CastingExpressionNode**
  allows casting a variable to certain types, for example to guarantee an integer or a boolean. It is used simply with an `as` keyword: `{myStringVariable as boolean}` or `{myBooleanVariable as integer}` and so on. Attempting to cast a variable to an incompatible type causes a standard Fluid error.
- **MathExpressionNode**
  allows basic mathematical operations on variables, for example `{myNumber + 1}`, `{myPercent / 100}` or `{myNumber * 100}` and so on. An impossible expression returns an empty output.

Extbase & Fluid

ExpressionNodes (3)

ExpressionNodeTypes allow new syntaxes such as:

- **TernaryExpressionNode**
  allows an inline ternary condition which only operates on variables. Typical use case is: "if this variable then use that variable else use another variable". It is used as:
  `{myToggleVariable ? myThenVariable : myElseVariable}`
  Note: does not support any nested expressions, inline ViewHelper syntaxes or similar inside it. It must be used only with standard variables as input.

Extbase & Fluid

Namespaces are extensible (1)

- Fluid allows each namespace alias (for example `f:`) to be extended by adding an additional PHP namespaces to it
- PHP namespaces are also checked for the presence of ViewHelper classes
- This also means that developers can override individual ViewHelpers with custom versions and have their ViewHelpers called when the `f:` namespace is used
- This change also implies that namespaces are no longer monadic. When using `{namespace f=My\Extension\ViewHelpers\}` you will no longer receive an "namespace already registered" error. Fluid will add this PHP namespace instead and look for ViewHelpers there as well.

Extbase & Fluid

Namespaces are extensible (2)

- Additional namespaces are checked from the bottom up, allowing the additional namespaces to override ViewHelper classes by placing them in the same scope
- For example: `f:format.nl2br` can be overridden by `My\Extension\ViewHelpers\Format\Nl2brViewHelper`, given the namespace registration on previous slide

Extbase & Fluid

Rendering using `f:render` (1)

Allow default content on optional `f:render`:

- Whenever `f:render` is used and flag `optional = TRUE` is set, rendering a missing section results in an empty output.
- Instead of rendering this empty output, a new attribute `default` (mixed) is added and can be filled with a fallback-type default value.
- Alternatively, the tag content can be used to define this default value like so many other content/attribute-flexible ViewHelpers

Extbase & Fluid

Rendering using `f:render` (2)

Passing of tag content from `f:render` to partial/section:

- Allows a new approach to structuring Fluid template rendering
- Partials and sections can be used as "wrappers" for an arbitrary piece of template code.
- Example:

```
<f:section name="MyWrap">
  <div>
    <!-- more HTML, using variables if desired -->
    <!-- tag content of f:render output: -->
    {contentVariable -> f:format.raw()}
  </div>
</f:section>

<f:render section="MyWrap" contentAs="contentVariable">
  This content will be wrapped. Any Fluid code can go here.
</f:render>
```

Extbase & Fluid

Complex conditional statements

- Fluid now supports any degree of complex conditional statements with nesting and grouping:

```
<f:if condition="({variableOne} && {variableTwo}) || {variableThree} || {variableFour}">
  // Done if both variable one and two evaluate to true,
  // or if either variable three or four do.
</f:if>
```

- In addition, `f:else` has been fitted with an "elseif"-like behavior:

```
<f:if condition="{variableOne}">
  <f:then>Do this</f:then>
  <f:else if="{variableTwo}">
    Do this instead if variable two evals true
  </f:else>
  <f:else if="{variableThree}">
    Or do this if variable three evals true
  </f:else>
  <f:else>
    Or do this if nothing above is true
  </f:else>
</f:if>
```

Extbase & Fluid

Dynamic variable name parts (1)

- Another forced new feature, likewise backwards compatible, is the added ability to use sub-variable references when accessing your variables. Consider the following Fluid template variables array:

```
$mykey = 'foo'; // or 'bar', set by any source
$view->assign('data', ['foo' => 1, 'bar' => 2]);
$view->assign('key', $mykey);
```

- With the following Fluid template:

```
You chose: {data.{key}}.
(output: "1" if key is "foo" or "2" if key is "bar")
```

Extbase & Fluid

Dynamic variable name parts (2)

- The same approach can also be used to generate dynamic parts of a string variable name:

```
$mydynamicpart = 'First'; // or 'Second', set by any source
$view->assign('myFirstVariable', 1);
$view->assign('mySecondVariable', 2);
$view->assign('which', $mydynamicpart);
```

- With the following Fluid template:

```
You chose: {my{which}Variable}.
(output: "1" if which is "First" or "2" if which is "Second")
```

Extbase & Fluid

New ViewHelpers

- A few new ViewHelpers have been added as part of standalone Fluid and as such are also available in TYPO3 from now on:
  - `f:or`
    This is a shorter way of writing (chained) conditions. It supports the following syntax, which checks each variable and outputs the first one that is not empty:

    ```
    {variableOne -> f:or(alternative: variableTwo) -> f:or(alternative: variableThree)}
    ```

  - `f:spaceless`
    This can be used in tag-mode around template code to eliminate redundant whitespace and blank lines for example caused by indenting ViewHelper usages

Extbase & Fluid

ViewHelper namespaces can be extended also from PHP

- By accessing the ViewHelperResolver of the RenderingContext, developers can change the ViewHelper namespace inclusions on a global (read: per View instance) basis:

```
$resolver = $view->getRenderingContext()->getViewHelperResolver();
// equivalent of registering namespace in template(s):
$resolver->registerNamespace('news', 'GeorgRinger\News\ViewHelpers');
// adding additional PHP namespaces to check when resolving ViewHelpers:
$resolver->extendNamespace('f', 'My\Extension\ViewHelpers');
// setting all namespaces in advance, globally, before template parsing:
$resolver->setNamespaces(array(
  'f' => array(
    'TYPO3Fluid\\Fluid\\ViewHelpers', 'TYPO3\\CMS\\Fluid\\ViewHelpers',
    'My\\Extension\\ViewHelpers'
  ),
  'vhs' => array(
    'FluidTYPO3\\Vhs\\ViewHelpers', 'My\\Extension\\ViewHelpers'
  ),
  'news' => array(
    'GeorgRinger\\News\\ViewHelpers',
  );
));
```

Extbase & Fluid

ViewHelpers can accept arbitrary arguments (1)

- This feature allows your ViewHelper class to receive any number of additional arguments using any names you desire
- It works by separating the arguments that are passed to each ViewHelper into two groups: those that are declared using `registerArgument` (or render method arguments), and those that are not
- Those that are not declared, are passed to a special function `handleAdditionalArguments` on the ViewHelper class, which in the default implementation throws an error if additional arguments exist

Extbase & Fluid

ViewHelpers can accept arbitrary arguments (2)

- By overriding this method in your ViewHelper, you can change if and when the ViewHelper should throw an error on receiving unregistered arguments
- This feature is also the one allowing TagBasedViewHelpers to freely accept arbitrary `data-` prefixed arguments without failing
- on TagBasedViewHelpers, the `handleAdditionalArguments` method simply adds new attributes to the tag that gets generated and throws an error if any additional arguments which are neither registered nor prefixed with `data-` are given.

Extbase & Fluid

Argument "allowedTags" for f:format.stripTags

- The argument `allowedTags` containing a list of HTML tags which will not be stripped can now be used on `f:format.stripTags`
- Tag list syntax is identical to second parameter of PHP function `strip_tags` (see: http://php.net/strip_tags)

Extbase & Fluid

Allow accessing ObjectStorage as array in Fluid

- Creates an alias of `toArray()` allowing the method to be called as `getArray()` which in turn allows the method to be called transparently from `ObjectAccess::getPropertyPath`, enabling access in Fluid and other places
- By creating a very simple aliasing of `toArray()` on ObjectStorage, allowing it to be called as `getArray()`
- Example: get the 4th element

```
// in PHP:
ObjectAccess::getPropertyPath($subject, 'objectstorageproperty.array.4')

// in Fluid:
{myObject.objectstorageproperty.array.4}
{myObject.objectstorageproperty.array.{dynamicIndex}}
```

Chapter 5:

Deprecated/Removed Functions

Deprecated/Removed Functions

Miscellaneous

- The following configuration options have been removed:
  - `$TYPO3_CONF_VARS['SYS']['t3lib_cs_utils']`
  - `$TYPO3_CONF_VARS['SYS']['t3lib_cs_convMethod']`

  (functionality is now auto-detected and `mbstring` is used by default if available)
- The deprecated TypoScript property `page.includeJSlibs` has been removed. Use the TypoScript property `page.includeJSLibs` (capital "L") instead
- The TypoScript option `config.renderCharset`, which was used as character set for internal conversion within a frontend request, has been removed

Chapter 6:

Sources and Authors

Sources and Authors

Sources

TYPO3 News:

- http://typo3.org/news

Release Infos:

- http://wiki.typo3.org/TYPO3_CMS_8.0.0
- INSTALL.md and ChangeLog
- typo3/sysext/core/Documentation/Changelog/8.0/*

TYPO3 Bug-/Issuetracker:

- https://forge.typo3.org/projects/typo3cms-core

TYPO3 and Fluid Git Repositories:

- https://git.typo3.org/Packages/TYPO3.CMS.git
- https://github.com/TYPO3Fluid/Fluid

TYPO3

Sources and Authors

TYPO3 CMS What's New Team:

Andrey Aksenov, Pierrick Caillon, Sergio Catala, Jigal van Hemert, Patrick Lobacher, Michel Mix, Sinisa Mitrovic, Angeliki Plati, Nena Jelena Radovic, Michael Schams and Roberto Torresani

http://typo3.org/download/release-notes/whats-new