TYPO3 Skinning Reference

Extension Key: doc_core_skinning
Language: en
Version: 1.0.0
Keywords: forDevelopers, forAdvanced

Official documentation

This document is included as part of the official TYPO3 documentation. It has been approved by the TYPO3 Documentation Team following a peer-review process. The reader should expect the information in this document to be accurate - please report discrepancies to the Documentation Team (documentation@typo3.org). Official documents are kept up-to-date to the best of the Documentation Team's abilities.

Core Manual

This document is a Core Manual. Core Manuals address the built in functionality of TYPO3 and are designed to provide the reader with in-depth information. Each Core Manual addresses a particular process or function and how it is implemented within the TYPO3 source code. These may include information on available APIs, specific configuration options, etc.

Core Manuals are written as reference manuals. The reader should rely on the Table of Contents to identify what particular section will best address the task at hand.

Table of Contents

Introduction

About this document

This document explains how CSS classes and icons in the TYPO3 Backend are defined. It is the basis for a consistent naming convention to ease the skinning of the TYPO3 Backend, as well as to ensure consistency.

The goals of the CSS naming convention are the following points:

- The designer should have a clear idea for which purpose a CSS class is intended to be used just by reading the name of the class. He should have a clear understanding which CSS selectors he can use to style elements.
- The developer must be aware where he should add CSS classes to the HTML output. When new CSS classes are required, he should know how to define them, since he has a document explaining the naming conventions.

Developers and designers should stick to these guidelines as much as possible. Following these simple rules will make it easier to have a consistent look and feel all around the TYPO3 Backend. Old pieces of code are still present in TYPO3 and it will almost be impossible to change everything but this document is considered as a starting point for new developments.

Note that this document is provided as guidelines **for TYPO3 v4.4 and above**. TYPO3 versions below v4.4 will stick to the old legacy CSS stylesheet located in `typo3/stylesheet.css` of the TYPO3 package. This document dœs not target TYPO3 v5 either.

What's new

This is the first version of this manual.

Credits

This manual was written by Fabien Udriot with the help of Steffen Ritter and Steffen Gebert.

During the process of refactoring the skin system for TYPO3, these different projects and readings provided inspiration:

- Ext JS for CSS organization and CSS naming
- freedesktop.org Icons Naming Specification
- OOCSS for a maintainable, standards-based CSS code

Feedback

For general questions about the documentation get in touch by writing to documentation@typo3.org.

If you find a bug in this manual, please file an issue in this manual's bug tracker:
http://forge.typo3.org/projects/typo3v4-doc_core_skinning/issues

Maintaining quality documentation is hard work and the Documentation Team is always looking for volunteers. If you feel like helping please join the documentation mailing list (typo3.projects.documentation on lists.typo3.org).

Supported Browsers

The TYPO3 Backend (and thus the CSS code) supports the following browsers:

- All Gecko-based browsers, version 1.8+ (used in Mozilla Firefox 1.5 and higher)
- All Webkit-based browsers; Safari, Chrome and Konqueror
- Opera 9 and higher
- Microsoft Internet Explorer version 6 and higher

CSS Files Organization

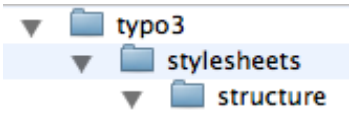
All CSS files used in the Backend are in one of two following folders "**structure**" or "**visual**".

Every CSS file inside the "**structure**" directory deals with the layout, positioning of elements and grid-like structure of a page. Typical attributes are: padding, margin, height, width, position, float, etc...

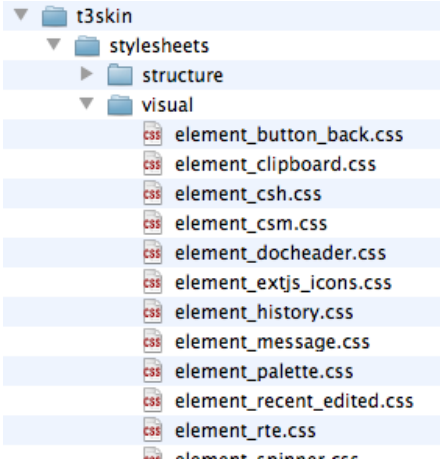
Every CSS file inside the "**visual**" directory basically deals with colors, background images and so on. Typical attributes are: font-size, font-weight, color, background, etc...

Structure	Visual
padding, margin width, height position float display visibility top, right, bottom, left z-index clear	font color background border overflow vertical-align white-space cursor text box-shadow list-style line-height

In the TYPO3 Core itself, you will find **mainly** styles that belong to the structure of a page, you will find this under typo3/stylesheets/structure/. There is **no** visual stylesheet because it is the purpose of a skin to "dress up" the TYPO3 Backend. You can experiment by removing every skin extension from the Extension Manager. You will have a "naked" but still usable TYPO3 Backend.

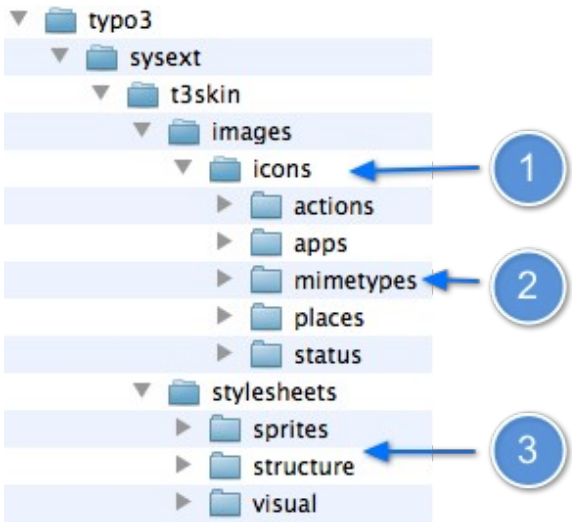


Furthermore, styles should be grouped together to identify them easily. But instead of having one big file that contains all styles, they are in separate files according to their purpose. The stylesheets for the TYPO3 Core are:



For a more in-depth understanding of the structure, have a look into the files within "typo3/stylesheets/".

In a skin extension, it makes sense to have a "visual" folder containing all the icons, backgrounds, stylesheets etc. The "structure" folder will contain styles that may override the default structure styles. Normally, you should have more visual styles than structure styles in a skin.



1. the icons folder
2. the different group of icons
3. the stylesheets containing the sprites, structure and visual information

Backend CSS API

This section shows skin developers how to add their stylesheet information correctly.

Skinning API

TYPO3 v4.4 introduces a new skinning API which enables you to load as many stylesheets as needed in the Backend. Convention over configuration tends to be applied when loading stylesheets. TYPO3 is expecting to find CSS files within two directories:

```
* EXT:t3skin_improved/stylesheets/structure
* EXT:t3skin_improved/stylesheets/visual
```

The example below shows up the most simple way for adding stylesheets in the Backend. Since the system assumes that CSS files are to be found in folder "structure" and "visual", TYPO3 will automatically load CSS files at the latest. Following lines have to be put in file ext_tables.php of the extension.

```
$GLOBALS['TBE_STYLES']['skins']['t3skin_improved'] = array();
$GLOBALS['TBE_STYLES']['skins']['t3skin_improved']['name'] = 'My improved t3skin';
```

If conventions are respected, TYPO3 will use the array as below. If no files are found, it will be simply skipped.

```
$GLOBALS['TBE_STYLES']['skins']['t3skin_improved'] = array();
$GLOBALS['TBE_STYLES']['skins']['t3skin_improved']['name'] = 'My improved t3skin';
$GLOBALS['TBE_STYLES']['skins']['t3skin_improved']['stylesheetDirectories'] = array(
    'structure' => 'EXT:t3skin_improved/stylesheets/structure',
    'visual' => 'EXT:t3skin_improved/stylesheets/visual',
);
```

Conventions may be changed by overriding informations in $GLOBALS['TBE_STYLES'].

```
$GLOBALS['TBE_STYLES']['skins']['t3skin_improved']['stylesheetDirectories'] = array(
    'structure' => 'EXT:t3skin_improved/other_directory/structure', // changes default
directory
    'visual' => '', //removes visual stylesheet
);
```

Additionally, it is possible to load extra groups of stylesheets by completing the array like this:

```
$GLOBALS['TBE_STYLES']['skins']['t3skin_improved']['stylesheetDirectories'] = array(
    'EXT:t3skin_improved/stylesheets/extjs',
    'EXT:t3skin_improved/stylesheets/ie6',
    'EXT:t3skin_improved/stylesheets/rtehtmlarea',
);
```

CSS concatenation

CSS concatenation is the process of combining and minifying all CSS stylesheets that are loaded. This way the browser only has to load a single stylesheet instead of many, which saves many HTTP requests. The compression is done automatically when using TYPO3 v4.4 or greater.

The concatenated files are stored in folder /typo3temp/compressor/, typically:

```
../typo3temp/compressor/merged-43184ce406ccfb7c04df66f024414129-
403f03ea0692e1f848b48ae4da48b005.css?1278079667
```

Segment	Description
First md5: 43184ce406ccfb7c04df66f024414129	Md5 of the whole content (not yet concatenated).
Second md5: 403f03ea0692e1f848b48æ4da48b005	Md5 of the file name + file path + file size
Time stamp: 1278079667	The creation time stamp of the file itself.

CSS compression

CSS compression enables to reduce drastically the size of the exchanged data between the server and the browser.

Steps can be enabled in the Backend as follows:

1. In the module "Install" > "All configuration", set compression level between 1 and 9, where 1 is least compression and 9 is greatest compression. Suggested and most optimal value is 5.

```
$GLOBALS['TYPO3_CONF_VARS']['BE']['compressionLevel'] = 5;
```

2. In a `.htaccess` file or in virtual host add some configuration provided by file `misc/advanced_htaccess`.

```
<FilesMatch "\.js\.gzip$">
      AddType "text/javascript" .gzip
</FilesMatch>
<FilesMatch "\.css\.gzip$">
      AddType "text/css" .gzip
</FilesMatch>
AddEncoding gzip .gzip
```

Following those 2 steps will generate a compressed file and therefore add a "gzip" suffix to the file.

```
merged-43184ce406ccfb7c04df66f024414129-5c86564215e4bad82a1955b74b639532.css.gzip?1278152902
```

It may happen that the browser dœs not support GZIP compression for some reason. Typically, it can be behind a proxy server which dœs not support GZIP headers. In this case, the compressor will detect it and send the un-compressed files.

Icons API

New methods have been added for the purpose of icons generation. The HTML should not be written manually but rather should be generated by the means of the API. The official format for icons is PNG 24 bits. There is a special fix for IE6 that solves the PNG transparency's issue.

The tables bellow gives an overview of the most common classes that are involved in the icons API.

Classes	Description
final t3lib_iconWorks t3lib/class.t3lib_iconworks.php	Icon generation, backend This library has functions that returns - and if necessary creates - the icon for an element in TYPO3
t3lib_SpriteManager t3lib/class.t3lib_spritemanager.php	TYPO3 sprite manager, used in BE and in FE if a BE user is logged in. This class builds CSS definitions of registered icons, writes TCA definitions and registers sprite icons in a cache file. A configurable handler class dœs the business task.
interface t3lib_spritemanager_SpriteIconGenerator t3lib/interfaces/interface.t3lib_spritemanager_spriteicongenerator.php	Interface all handlers for t3lib_spritemanager have to implement.
t3lib_spritemanager_SimpleHandler t3lib/spritemanager/class.t3lib_spritemanager_simplehandler.php	A class with an concrete implementation of t3lib_spritemanager_SpriteIconGenerator. It is the standard / fallback handler of the sprite manager. This implementation won't generate sprites at all. It will just render css-definitions for all registered icons so that they may be used through t3lib_iconWorks::getSpriteIcon. Without the css classes generated here, icons of for example TCA records would be empty.

The section below gives examples how to use method of t3lib_iconWorks

Method name	Description
t3lib_iconWorks::getSpriteIconClasses($iconName)	generic method to create the final CSS classes based on the sprite icon name with the base class and splits the name into parts is usually called by the methods that are responsible for fetching the names out of the file name, or the record type

Example

```
####
t3lib_iconWorks::getSpriteIconClasses('actions-document-new')
Result:
t3-icon t3-icon-actions t3-icon-actions-document t3-icon-document-new
```

Method name	Description
t3lib_iconWorks::getSpriteIcon($iconName, array $options = array(), array $overlays = array())	This generic method is used throughout the TYPO3 Backend to show icons in any variation which are not bound to any file type (see getSpriteIconForFile) or database record (see getSpriteIconForRecord)

Examples

```
####
t3lib_iconWorks::getSpriteIcon('actions-document-new')
Result:
<span class="t3-icon t3-icon-actions t3-icon-actions-document t3-icon-document-new"></span>

####
t3lib_iconWorks::getSpriteIcon('actions-document-new', array('title' => 'foo'))
Result:
<span title="foo" class="t3-icon t3-icon-actions t3-icon-actions-document t3-icon-document-
new"></span>

####
t3lib_iconWorks::getSpriteIcon('actions-document-new', array(), array('status-overlay-
hidden' => array()))
Result: notice the additional "t3-icon-overlay" class
<span class="t3-icon t3-icon-actions t3-icon-actions-document t3-icon-document-new">
  <span class="t3-icon t3-icon-status t3-icon-status-overlay t3-icon-overlay-hidden t3-icon-
overlay"></span>
</span>
```

Method name	Description
t3lib_iconWorks::getSpriteIconForRecord($table, array $row, array $options = array())	This method is used throughout the TYPO3 Backend to show icons for a DB record. Generates a HTML tag with proper CSS classes. The TYPO3 skin has defined these CSS classes already to have a pre-defined background image, and the correct background-position to show the necessary icon.

Examples

```
####
t3lib_iconWorks::getSpriteIconForRecord('tt_content', array())
Result:
<span class="t3-icon t3-icon-mimetypes t3-icon-mimetypes-x t3-icon-x-content-text"></span>
####
t3lib_iconWorks::getSpriteIconForRecord('tt_content', array('hidden' => 1))
Result:
<span class="t3-icon t3-icon-mimetypes t3-icon-mimetypes-x t3-icon-x-content-text">
 <span class="t3-icon t3-icon-status t3-icon-status-overlay t3-icon-overlay-hidden t3-icon-
overlay"></span>
</span>
####
t3lib_iconWorks::getSpriteIconForRecord('tt_content', array(), array('class' => 'foo',
'title' => 'bar'))
Result:
```

```
<span class="t3-icon t3-icon-mimetypes t3-icon-mimetypes-x t3-icon-x-content-text foo"
title="bar"></span>
```

Method name	Description
t3lib_iconWorks::getSpriteIconForFile($file, array $options = array())	This method is used throughout the TYPO3 Backend to show icons for a file type. Generates a HTML tag with proper CSS classes. The TYPO3 skin has defined these CSS classes already to have a pre-defined background image, and the correct background-position to show the necessary icon.

Examples

```
####
t3lib_iconWorks::getSpriteIconForFile('pdf')
Result:
<span class="t3-icon t3-icon-mimetypes t3-icon-mimetypes-pdf t3-icon-pdf"></span>
####
t3lib_iconWorks::getSpriteIconForFile('filename.pdf')
Result:
<span class="t3-icon t3-icon-mimetypes t3-icon-mimetypes-pdf t3-icon-pdf"></span>
####
t3lib_iconWorks::getSpriteIconForFile('pdf', array('title' => 'bar'))
Result:
<span title="bar" class="t3-icon t3-icon-mimetypes t3-icon-mimetypes-pdf t3-icon-
pdf"></span>
```

Sprite Generation

In version 4.4, the generation of sprites was assigned to the developer who should shipped skin extensions with sprites already "compiled". As from version 4.5, TYPO3 provides a mechanism that "compile" icons into sprites automatically. The mechanism is fully transparent and is fairly swift. The sprite generation dœs not modify the quality of the icons meaning the color's depth remains unchanged.

The sprite generation process involves different mechanism which are supporting by the **Sprite Manager** and the **Sprite Generator Handler**.

Sprite Manager

the **Sprite Manager** is instantiated each time the Backend is loaded and one of its main role is to verify whether new icons have been added or not by comparing md5 value from the file cache with the serialized icon list. The file cache md5 is to be found at typo3temp/sprites/*.inc

Sprite Generator Handler

Sprite Generator Handler are configurable "workers" of the spriteManager. Their tasks is to collect informations about icons from extensions and have to make it usable for the Backend. In other words, there are in charge of generating sprites and CSS files.

If new icons have been added, **Sprite Generator Handler** is triggered and can "enter the stage" in three different manners. Within the install tool, there is the key "**spriteIconGenerator_handler**" that is used to configure the way the **Sprite Generator Handler** is acting. Currently it accepts:

- **simple** which turns out stylesheet in typo3temp/sprites/ but dœs **not** generate any sprite at all. Each image is "linked" independently. This handler is already present in TYPO3 v4.4.
- **auto-generating** which extends the "simple" handler by producing a sprite additionally as stylesheet. This handler is present upon TYPO3 v4.5.
- **manual auto-generating** - instead of letting TYPO3 taking care of the CSS / Sprite generation automatically, this option will activate a new icon in the Clear Cache menu that will enable to manually control the Sprite Manager. This handler is present upon TYPO3 v4.5.

Notice that the Sprite Generator only generate sprite from icons that have been added afterwards through extensions. The sprite and stylesheet shipped with the Core will **never** be changed by the Sprite Generator. Therefore, icons are pre-compiled into **one** "big" sprite for the whole Backend.

Sprites and stylesheets provided by the Core are located in **t3skin** as follows:

Path	Description
typo3/sysext/t3skin/images/sprites/t3skin.png typo3/sysext/t3skin/images/sprites/t3skin.gif	The sprites provided by t3skin for the whole backend. The gif sprite is for IE6 compatibility.
typo3/sysext/t3skin/stylesheets/sprites/t3skin.css	Contains the CSS stylesheets that will be used for positioning the sprite.

Sprites and stylesheets that are generated afterwards by the means of the Sprite Generator Handler are located in typo3temp as follows:

Path	Description
...	The sprite / icons location will depend on how the handler is configured in the install tool under key "spriteIconGenerator_handler". It may accept thre different values: simple – autogenerating – manual autogenerating
typo3temp/srpites/zextensions.css	Contains the CSS stylesheets that will be used for positioning the sprite.

Use case #2 gives a good insight of the API to be used to add new icon in the Backend. The common way, is to call method addSingleIcons as follows:

```
// Gives the $icon array to the sprite manager
t3lib_SpriteManager::addSingleIcons($icons, 'foo');
```

CSS Coding Guidelines

- All CSS selectors (classes or IDs) are lowercase, multiple words are separated with a hyphen, no underscore nor camel-case.
- All CSS definitions should take place in a CSS file or in a CSS part at the top of the page, inline styles are highly discouraged.
- All CSS inclusions should be done through the appropriate <style> tag in HTML, and not through the @import statement, they also should have proper “media” attributes.
- It is encouraged to use generic classes for common styling issues instead of “id”.
- It is discouraged to prepend the HTML tag in front of a generic CSS selector, however if you need to specify one, write it in lowercase (span.t3-icon instead of SPAN.t3-icon).
- Use as little cascading as possible. “It is encourage to use cascading when there is no class defined”
- Also, it is strongly discouraged to use multiple class definitions in one selector due to a bug in Internet Explorer 6 (e.g. .t3-icon.t3-icon-blue).
- Attribute selectors (e.g. input[type=submit]) should be avoided also due to certain limitations in some browsers.

CSS statements are written as below:

```
.t3-icon,
.t3-link {
	display: inline;
	overflow: hidden;
	height: 18px;
	padding-bottom: 5px;
	padding-left: 18px;
}
```

Notes

1. If you apply a style to multiple selectors, separate the selectors with a comma and a line-break.

2. The opening curly brace is divided by a space from the selector and a line-break for the statements.

3. Every CSS command is indented with a tab, every property is followed by a colon and then a white-space, every statement is separated by a line-break.

4. When using the short version for properties like “padding”, “margin” or “border”, instead of “padding-left”, “padding-right”..., all four values are preferred (padding: 10px 0 10px 0). Two values are also allowed in this case (padding: 10px 0).

5. Null-values should be abbreviated with “0” instead of “0em” or “0px”.

CSS Naming Conventions

A CSS class **name** is defined according to whatever the element **is** or dœs rather than being linked to a specific context. The purpose is to be able to reuse a name anytime in the TYPO3 Backend while keeping a consistent look & feel.

Nevertheless, to avoid conflicts in the naming scheme, all news styles are expected to use a "t3-" prefix. This will prevent naming collisions when mixing up stylesheets with another application or styles from the old skinning parts.

To be more concrete, let's give a good example of CSS class names:

```
<input class="t3-form-text t3-form-field" type="text" />
```

At the first glance, it seems to be redundant to have multiple classes, but in fact it allows to have very fine-grained CSS selectors. The "t3-form-field" class is the base class for every input elements within the TYPO3 backend, and enables TYPO3 to give a default style to every input elements. In addition, there is the "t3-form-text" class to make it possible to have additional decorations on the input. Please notice the dash "-" which is used as separator inside the names.

Now let's have a look at a **bad** example:

```
<input class="t3-input-line-table">
```

This is a **bad** example as one can't guess the purpose of the "t3-input-line-table" selector. It contains the word “table” but is used within an input field which makes it semantically hard to understand the purpose of the class. Furthermore it creates confusion with the “table” HTML tag.

Icons naming conventions

This section aims to explain how classes apply to links. As reminder, icons are merged dynamically in sprites (cf. chapter "Sprite Generator"). Additionally, CSS classes are generated and outputted for each image which will be used for positioning the sprite correctly.

To put the CSS classes in context, let's consider the HTML code that is necessary for displaying an icon.

```
<a href="#" class="t3-link">
      <span class="t3-icon t3-icon-actions t3-icon-actions-document t3-icon-document-
new"></span>
</a>
```

The table below clarify the purpose of each class.

Class name	Description
t3-link	The base class of all links.
t3-icon	The base class of all icons.
t3-icon-actions	Defines what sprite is going to be used. In version 4.4, there is 5 main sprites. As from version 4.5, there is only one.
t3-icon-document-new	Defines the background position of the sprite.

Why are there so many classes rather than cascading?

Using many classes might burden the HTML output. But on the other hand, it is a fast way for the browser to decorate elements, rather than dealing with complicated cascades and selectors. This is especially true when dealing with a big XML tree structure like in the TYPO3 Backend. It is encouraged to use cascading when there is no class defined and styling for a certain element is needed. This is mostly true for inline elements like `legend`, `span`, `a`, `strong`, `blockquote`, `img`, `em`, `li`, etc.

Just a few examples:

```
.t3-form-fieldset legend {
      ...
}

.t3-form-element span {
      ...
}
```

When to use an id rather than a class attribute?

It makes sense to use the "id" attribute (additionally to a generic class if possible), especially when JavaScript is or could be involved in some way. Every “id” needs to be prefixed by "t3-" and should be used rather on block element like `div`, `form`, `p`, etc.

Example with block tag:

```
<div class=”t3-module” id="t3-module-page-container" />
<form class=”t3-form t3-form-fileupload” id="t3-fileupload" />
```

Special cases exist where an ID is needed to reference another element within the HTML document like "label" and "input" tags.

```
<label class=”t3-form-label” for="t3-myinput" />
<input class=”t3-form-input” id="t3-myinput" />
```

Use Cases

Use case 1: load additional stylesheets to skin the Backend

Step 1: add following line into ext_tables.php to register the extension

```
$GLOBALS['TBE_STYLES']['skins'][$EXTKEY]['name'] = $EXTKEY;
```

Step2: save CSS files with your extension. The name of the CSS files dœs not matter really. More importantly, CSS files need to be saved in a specific location to be automatically added onto the Backend:

- EXT:extension/stylesheets/structure/
- EXT:extension/stylesheets/visual/

Example of CSS

```
table.t3-page-columns {
    width:100%; // 800px
}

td.t3-page-column-2 {
    min-width:200px;
    Width:20%;
    background-color:red;
}

td.t3-page-column-0{
    width:78%;
    min-width:400px;
}
```

As an example, have a look at the extension pagemodulecss at Forge.

Use case 2: registering a new icon with the Backend

Save icons within your extension whenever it makes sense. As example, it can be placed in `EXT:extension/ Resources/Public/images/icons`.

Here are the steps:

1. copy icon in EXT:foo/Resources/Public/images/icons

2. declare your icon by adding following lines into EXT:foo/ext_tables.php

```
// Defines $icon array()
$pathToExtension = t3lib_extMgm::extRelPath('foo');
$icons = array(
      'error' => $pathToExtension . 'Resources/Public/images/icons/error.png',
      'information' => $pathToExtension . 'Resources/Public/images/icons/information.png',
      'notification' => $pathToExtension . 'Resources/Public/images/icons/notification.png',
      'ok' => $pathToExtension . 'Resources/Public/images/icons/ok.png',
      'warning' => $pathToExtension . 'Resources/Public/images/icons/warning.png',
);

// Gives the $icon array to the sprite manager
t3lib_SpriteManager::addSingleIcons($icons, 'foo');
```

3. Clear the "Configuration" cache to take into account the changes done in ext_tables.php

4. Finished! It should be possible to call the new icons like:

```
t3lib_iconWorks::getSpriteIcon('extensions-foo-warning')
```

will turn out:

```
<span class="t3-icon t3-icon-extensions t3-icon-extensions-devlog t3-icon-devlog-
warning"></span>
```

Use case 3: migration steps from legacy to new API

As TYPO3 4.4 removes all hardcoded icons from the Core, all icons in `typo3/gfx` have become superfluous as from now. However they are still present in the Core, mainly for backwards compatibility reasons. The same happens to t3ksin with its `icons/gfx`.

They are due to be removed in 4.6 and so developers are strongly encouraged to migrate the code basis to the sprite system.

As a first step, icons should be copied / pasted into your extension, as long as there are used in a different context as the ones in the Backend. Then, following "Use Case 2" it should be possible to call icons by the means of the API (cf. Chapter Icons API).

Bellow, find the equivalence of legacy API versus new API.

Legacy API > TYPO3 4.3	New API < TYPO3 4.4
	t3lib_iconWorks::getSpriteIcon('actions-document-new')
t3lib_iconWorks::getIconImage(...)	t3lib_iconWorks::getSpriteIconForRecord('tt_content')
	t3lib_iconWorks::getSpriteIconForFile('pdf')

CSS generic elements

Notice that HTML element are currently not implemented in this way throughout the Backend. However following examples can be taken as guidelines for further developments.

"A" tag with icon

API is meant be used to generate the link icon: getSpriteIcon() - getSpriteIconForRecord() - getSpriteIconForFile().

```
<a href="#" class="t3-link">
      <span class="t3-icon t3-icon-actions-edit t3-icon-edit-add"></span>
</a>
```

Button with icon

```
<input id="t3-generated-1" class="t3-icon t3-icon-actions-edit t3-icon-edit-add t3-button"
type="submit" value="Submit"/>
```

or

```
<button id="t3-generated-1" class="t3-icon t3-icon-actions-edit t3-icon-edit-add t3-button"
type="submit">Submit</button>
```

Input

```
<input id="t3-generated-1" class="t3-form-text t3-form-field" type="text" />
<input id="t3-generated-2" class="t3-form-checkbox t3-form-field" type="checkbox"/>
<input id="t3-generated-3" class="t3-form-radio t3-form-field" type="radio"/>
<select id="t3-generated-4" class="t3-form-select t3-form-field"></select>
```

Button without icon

```
<input type="submit" class="t3-button-submit t3-button" value="Submit"/>
```

or

```
<button type="submit" class="t3-button-text t3-button">Submit</button>
```

Form

```
<form id="t3-generated-1" class="t3-form" method="POST">
    <div class="t3-form-container">

        <!-- FIELDSET -->
        <fieldset class="t3-form-fieldset">
            <legend>Contact Information</legend>

            <!-- INPUT FIELD -->
            <div class="t3-form-item">
                <label class="t3-form-item-label" for="t3-component-1">First Name</label>
                <div class="t3-form-element">
                    <input id="t3-component-1" class="t3-form-text t3-form-field"
type="text" name="tx_extension[text_example]" size="20"/>
                </div>
                <div class="t3-form-clear-left"/>
            </div>

            <!-- CHECKBOX FIELD -->
            <div class="t3-form-item">
                <div class="t3-form-element">
                    <div class="t3-form-check-wrap">
                        <input id="t3-component-2" class="t3-form-checkbox t3-form-field"
type="checkbox" name="tx_extension[checkbox_example]"/>
                        <label class="t3-form-checkbox-label" for="t3-component-2">Item
1</label>
                    </div>
                    </div>
                <div class="t3-form-clear-left"/>
            </div>
```

```
            <!-- RADIO FIELD -->
            <div class="t3-form-item">
                <label class="t3-form-item-label" for="t3-component-3"/>
                <div class="t3-form-element">
                    <div class="t3-form-check-wrap">
                        <input id="t3-component-3" class="t3-form-radio t3-form-field"
type="radio" name="tx_extension[radio_example]" value="1"/>
                        <label id="t3-component-3" class="t3-form-checkbox-label" for="t3-
component-3">Item 1</label>
                    </div>
                </div>
                <div class="t3-form-clear-left"/>
            </div>

            <!-- SELECT FIELD -->
            <div class="t3-form-item">
                <label class="t3-form-item-label" for="t3-component-4">First Name</label>
                <div class="t3-form-element">
                    <select id="t3-component-4" class="t3-form-select t3-form-field">
                        <option value="1">Item 1</option>
                    </select>
                </div>
                <div class="t3-form-clear-left"/>
            </div>

            <!-- TEXTAREA FIELD -->
            <div class="t3-form-item">
                <label for="t3-component-5">Address</label>
                <div class="t3-form-element">
                    <textarea id="t3-component-5" class="t3-form-textarea t3-form-field"
name="tx_extension[textarea_example]"/>
                </div>
                <div class="t3-form-clear-left"/>
            </div>

            <div id="t3-generated-26" class="t3-form-clear"/>
            </fieldset>

        <!-- BUTTONS -->
        <div class="t3-form-buttons-container">
            <div class="t3-form-buttons t3-form-buttons-center">
                <button id="t3-generated-x" class="t3-button-text t3-button"
type="submit">TEXT</button>
                <div class="t3-clear"/>
            </div>
        </div>
    </div>
</form>
```

Table

```
<div id="t3-generated-1">
    <table class="t3-list">
        <thead>
                <tr class="t3-row-header">
                    <th class="t3-column t3-cell t3-td-1">
                        <div class="t3-cell-inner">Examples</div>
                    </th>
            </tr>
        </thead>
        <tbody>
            <tr class="t3-row t3-row-first t3-row-even">
                <td class="t3-cell t3-td-1">
                    <div class="t3-cell-inner">Examples</div>
                </td>
            </tr>
            <tr class="t3-row t3-row-odd">
                <td class="t3-cell t3-td-1">
                    <div class="t3-cell-inner">Examples</div>
                </td>
            </tr>
            <tr class="t3-row t3-row-last t3-row-even">
```

```
                    <td class="t3-cell t3-td-1">
                        <div class="t3-cell-inner">Examples</div>
                    </td>
                </tr>
            </tbody>
        </table>
    </div>
```

FAQ

Where can I find other or missing icons?

Although we tried to replace all icons, there might be some icons missing. In this case please refer to the fam fam fam icon set: http://www.famfamfam.com/lab/icons/silk/

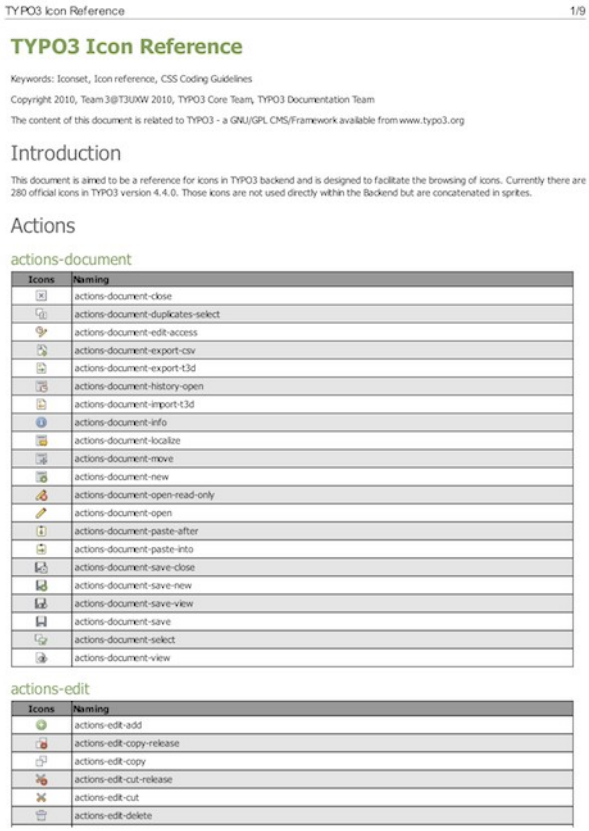
Next steps

This manual describes only the skinning API. To learn more about the many other APIs provided by TYPO3, please refer to the “Core API” manual.

Appendix A – Icon reference

The Icon Reference is a PDF document that brings together all icons from the Core into one file and can be used to know how to address icons without having to skim the whole TYPO3 source code. This document is **available along the current document** (i.e. inside extension doc_core_skinning) and is auto-generated for each TYPO3 version.

As an insight here's what the beginning of that document looks like:



TYPO3 Icon Reference 1/9

TYPO3 Icon Reference

Keywords: Iconset, Icon reference, CSS Coding Guidelines

Copyright 2010, Team3@T3UXW 2010, TYPO3 Core Team, TYPO3 Documentation Team

The content of this document is related to TYPO3 - a GNU/GPL CMS/Framework available from www.typo3.org

Introduction

This document is aimed to be a reference for icons in TYPO3 backend and is designed to facilitate the browsing of icons. Currently there are 280 official icons in TYPO3 version 4.4.0. Those icons are not used directly within the Backend but are concatenated in sprites.

Actions

actions-document

Icons	Naming
	actions-document-close
	actions-document-duplicates-select
	actions-document-edit-access
	actions-document-export-csv
	actions-document-export-t3d
	actions-document-history-open
	actions-document-import-t3d
	actions-document-info
	actions-document-localize
	actions-document-move
	actions-document-new
	actions-document-open-read-only
	actions-document-open
	actions-document-paste-after
	actions-document-paste-into
	actions-document-save-close
	actions-document-save-new
	actions-document-save-view
	actions-document-save
	actions-document-select
	actions-document-view

actions-edit

Icons	Naming
	actions-edit-add
	actions-edit-copy-release
	actions-edit-copy
	actions-edit-cut-release
	actions-edit-cut
	actions-edit-delete