EXT: DAM Lightbox

Extension Key: damlightbox

Language: en

Keywords: forAdmins, forDevelopers, forIntermediates, forAdvanced

Table of Contents

Introduction

What does it do?

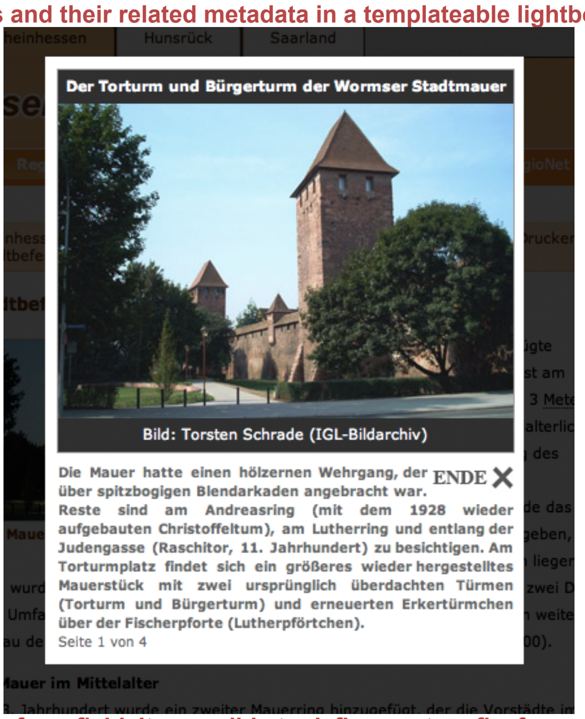
DAM Lightbox provides a comfortable, flexible and universally applicable way to work with DAM images and their related metadata in the TYPO3 back- and frontend. The extension has three main features:

1. It provides you with two generic DAM fields that can be added to any TYPO3 table/record without changing the database: a field for image references and a flexform field for configuration. The fields are configured using the extension's backend module.
2. You can access any metadata of referenced DAM images for any TYPO3 record just using TypoScript.
3. All DAM images and their according metadata can be shown in a templateable popup or lightbox. The extension uses pmkslimbox as an example but it's possible to connect other lightboxes.

Working implementations of DAM Lightbox for tt_content, tt_news, tt_address and pages are shipping with this extension.

Screenshots

Example 1: Showing DAM images and their related metadata in a templateable lightbox



Der Torturm und Bürgerturm der Wormser Stadtmauer

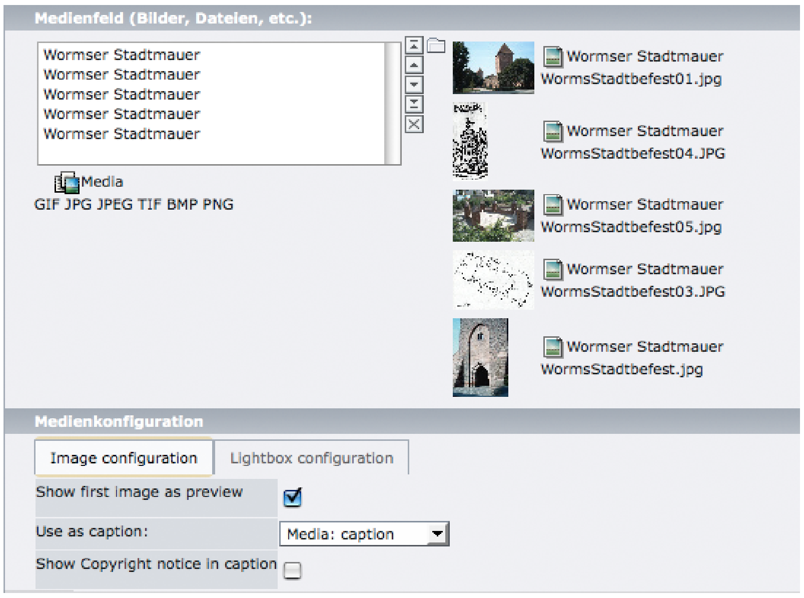
Bild: Torsten Schrade (IGL-Bildarchiv)

ENDE ✕

Die Mauer hatte einen hölzernen Wehrgang, der über spitzbogigen Blendarkaden angebracht war. Reste sind am Andreasring (mit dem 1928 wieder aufgebauten Christoffeltum), am Lutherring und entlang der Judengasse (Raschitor, 11. Jahrhundert) zu besichtigen. Am Torturmplatz findet sich ein größeres wieder hergestelltes Mauerstück mit zwei ursprünglich überdachten Türmen (Torturm und Bürgerturm) und erneuerten Erkertürmchen über der Fischerpforte (Lutherpförtchen).

Seite 1 von 4

Example 2: Generic image and flexform field. Its possible to define custom flexforms for each table with PageTSConfig



Medienfeld (Bilder, Dateien, etc.):

Wormser Stadtmauer
Wormser Stadtmauer
Wormser Stadtmauer
Wormser Stadtmauer
Wormser Stadtmauer

Media
GIF JPG JPEG TIF BMP PNG

Wormser Stadtmauer
WormsStadtbefest01.jpg

Wormser Stadtmauer
WormsStadtbefest04.JPG

Wormser Stadtmauer
WormsStadtbefest05.jpg

Wormser Stadtmauer
WormsStadtbefest03.JPG

Wormser Stadtmauer
WormsStadtbefest.jpg

Medienkonfiguration

Image configuration | Lightbox configuration

Show first image as preview	☑
Use as caption:	Media: caption
Show Copyright notice in caption	☐

Example 3: Backend module to add and configure the generic image and configuration field

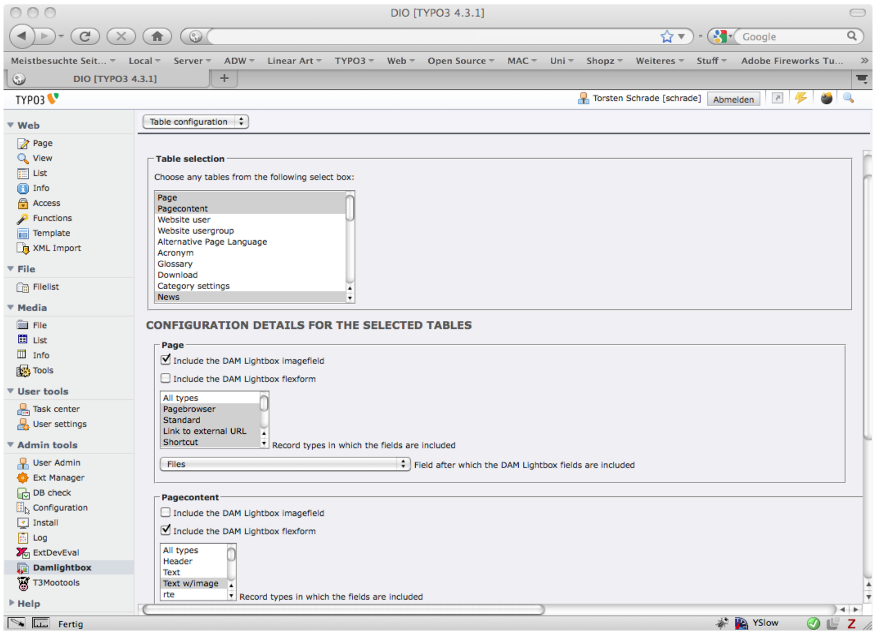


Table configuration

Table selection

Choose any tables from the following select box:

Page
Pagecontent
Website user
Website usergroup
Alternative Page Language
Acronym
Glossary
Download
Category settings
News

CONFIGURATION DETAILS FOR THE SELECTED TABLES

Page

- [✓] Include the DAM Lightbox imagefield
- [ ] Include the DAM Lightbox flexform

All types / Pagebrowser / Standard / Link to external URL / Shortcut — Record types in which the fields are included

Files — Field after which the DAM Lightbox fields are included

Pagecontent

- [ ] Include the DAM Lightbox imagefield
- [✓] Include the DAM Lightbox flexform

All types / Header / Text / Text w/image / rte — Record types in which the fields are included

Feature list

- Generic DAM image & configuration field for any TYPO3 table that is registered in $TCA
- Transfers metadata of DAM images into TSFE with easy TypoScript access
- Templateable HTML popup / lightbox
- Almost anything configurable with TypoScript
- Flexform configuration for each referenced image; general values can be set from TypoScript
- Individual flexforms can be configured on a per table basis via PageTSConfig
- Can also be used without any lightbox
- Full support for pmkslimbox (3.0.1+), including printing and saving of click-enlarged images
- Can be used either standalone or together with dam_ttcontent (1.1.0+), dam_ttnews (0.1.9+) and dam_pages (0.1.7+)
- Sample configuration for tt_content, tt_news, tt_address and pages table.

Requirements

- TYPO3 (4.2+)
- dam (1.1.1+)

Optional extensions

- dam_ttcontent (1.1.0+)
- dam_ttnews (0.1.9+)
- dam_pages (0.1.7+)
- pmkslimbox (3.0.1+) / t3mootools (1.3.0+)

Credits

- Credits and big thanks to René Fritz who first provided us with the DAM.
- Credits and big thanks to the DAM development team. They do an excellent job!
- Credits and big thanks to Peter Klein for pmkslimbox and t3mootools.
- Further credits to the DAM mailinglist for feedback and suggestions.

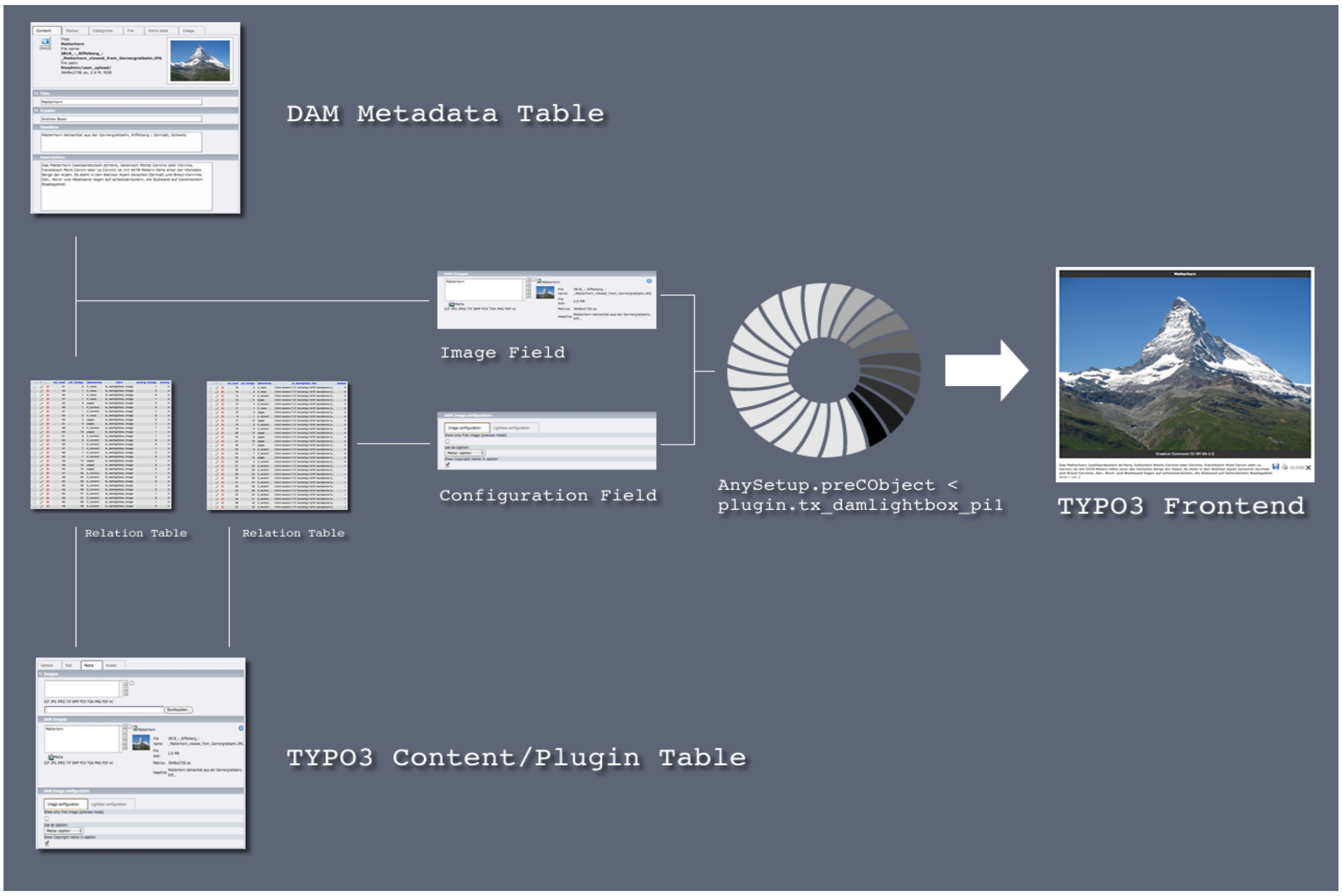
Please rate

No matter if you like or don't like the extension, please rate it in TER. Each good rating motivates me to keep going. Each bad rating motivates me to make the stuff better :)

Users manual

Basic principles

In it's early days DAM Lightbox started out as an extension for DAM and dam_ttcontent. While the lightbox functionality is still a major feature of this extension, the general focus nowadays is to provide a kind of connector between DAM and TSFE. Please have a look at the following sketch:

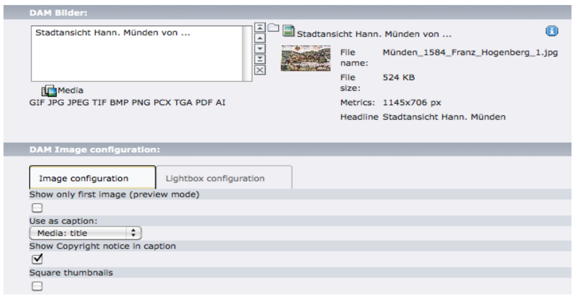


All references from a DAM item to a record in the TYPO3 system are kept within a reference table (tx_dam_mm_ref). Extensions like dam_ttcontent provide dedicated database fields for specific TYPO3 tables with which the references between the records and the DAM items are managed. So far so good. But as soon as you work with a TYPO3 table for which there doesn't exist a "dam_xxx" extension, you have to build a clone that provides this functionality. Generally speaking, those dedicated database fields are not really necessary. They just provide a view on the relationship table and don't contain any useful data by themselves.

Therefore, the approach of DAM Lightbox is different. Here the image references are handled by a "generic" field that is included "on the fly" when a backend form is loaded. This image field can be displayed for any TYPO3 table that is configured in the Table Configuration Array ($TCA). It acts like a normal field, but it's not in the database.

Besides the generic image field the extension provides another "on the fly" field. This second field is a generic flexform field. The idea of the generic flexform field is to provide editors with the possibility to configure the display of the DAM images and their metadata.

Example:



DAM Bilder:

Stadtansicht Hann. Münden von ...

Media
GIF JPG JPEG TIF BMP PNG PCX TGA PDF AI

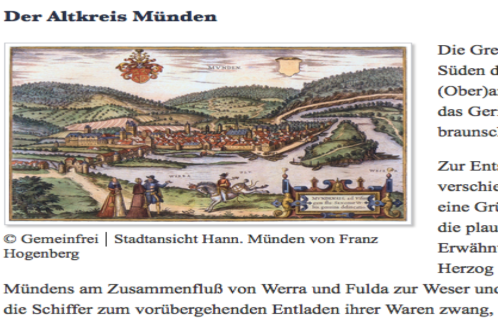
Stadtansicht Hann. Münden von ...

File name:	Münden_1584_Franz_Hogenberg_1.jpg
File size:	524 KB
Metrics:	1145x706 px
Headline	Stadtansicht Hann. Münden

DAM Image configuration:

Image configuration | Lightbox configuration

Show only first image (preview mode)
☐
Use as caption:
Media: title
Show Copyright notice in caption
☑
Square thumbnails
☐



Der Altkreis Münden

© Gemeinfrei | Stadtansicht Hann. Münden von Franz Hogenberg

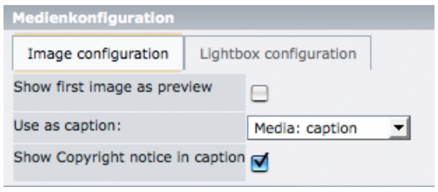
Die Gre
Süden d
(Ober)a
das Ger
braunsc

Zur Ent
verschie
eine Gri
die plau
Erwähn
Herzog
Mündens am Zusammenfluß von Werra und Fulda zur Weser und
die Schiffer zum vorübergehenden Entladen ihrer Waren zwang,

Here you can see a referenced DAM image in the generic image field and it's display on the website. In the generic flexform the editor has chosen to show the title of the DAM record as image caption and to include the DAM copyright notice within the image caption. In case of several images, the editor could decide to display them as square thumbnails or to show only the first of them as a kind of preview image. The functionality of the flexform field is implemented using pure TypoScript. Since you can include your own flexforms, you are free to implement any functionality that you might need.

Basic functionality

Since there are many ways of how you could display your DAM metadata together with your DAM images in the frontend, this extension just implements some basic functionality. The Howto section of this manual provides you with step-by-step guide on how to do more. This is what works out of the box:



Medienkonfiguration

Image configuration | Lightbox configuration

Show first image as preview	☐
Use as caption:	Media: caption
Show Copyright notice in caption	☑

Image previews:

In cases where you have more than one image attached to a record you can use this checkbox to just show the first one while the others will be browsable in the popup/lightbox.

Use as caption:

Gives you the possibility to select between media:title, media:caption and media:description for your image captions. The content can be overridden with the standard caption field.

Show copyright:

If this is checked, the value from media:copyright will be prepended to the image caption.

Lightbox - Show as caption:

Lets you decide if you want to show media:title, media:caption or media:description as caption of the lightbox. The standard value set from TS is media:description.

Lightbox - Custom dimensions:

In some situations the dimensions of the lightbox need to be adjusted. Here you can insert custom dimensions for pmkslimbox using the following notation: imagenumber:pixelwidth,pixelheight;

Administration

Installation & Configuration

Step 1: Installation

Just install the extension from TER using the Extension Manager. Alternatively you can checkout the latest development version from the TYPO3 Forge on https://svn.typo3.org/TYPO3v4/Extensions/damlightbox

Important:

Please make sure that damlightbox is installed **after** any extension with which you want to use it. Otherwise the generic fields will not appear when the records are loaded. To be sure, have a look in the extList variable in localconf.php. DAM Lightbox should come last:

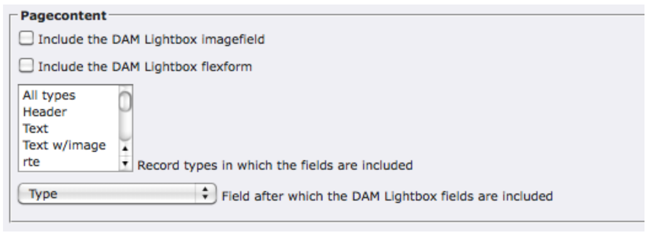
```
$TYPO3_CONF_VARS['EXT']['extList'] = '...OTHER EXTS...,damlightbox';
```

Step 2: Default Flexform

During installation you can chose the default path that points to the standard flexform file for the generic configuration field. This file will be used as long as you don't specify a custom flexform via PageTSConfig (see the administration section on how to do that).

Step 3: Backend configuration

After installation configure the tables for which the generic fields should be included. Change from the Extension Manager to the DAM Lightbox backend module. Select all tables for which you would like to include the generic fields and hit "Refresh Configuration". Now you will see some configuration details for the tables of your choice. You can set four options:


Pagecontent

☐ Include the DAM Lightbox imagefield

☐ Include the DAM Lightbox flexform

All types
Header
Text
Text w/image
rte

Record types in which the fields are included

Type — Field after which the DAM Lightbox fields are included

1. Include the DAM Lightbox imagefield: Activate this checkbox and the generic image field will be included in the chosen table.
2. Include the DAM Lightbox flexform: Activate this checkbox and the generic flexform field will be included in the chosen table.
3. Record types: Here you can decide in which of the record types of the chosen table the generic fields should be included. For example, in tt_content it makes sense to include the fields only for the image and text w/image types.
4. After field: Here you can define the position where the generic fields will be included in the backend form.

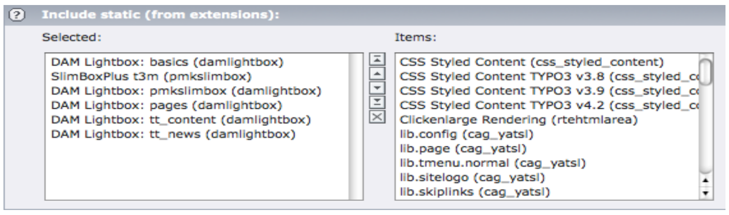
Note:

Which of the generic fields you should include really depends on your setup. For example, if you are using dam_ttcontent it makes no sense to include the generic imagefield since you already have a reference field for tt_content in place. Same goes for dam_ttnews and dam_pages usage. There may also be situations in which you just want to include the generic image field and not the generic flexform field.

Once you have finished the configuration for all selected tables hit "Write configuration!" and clear the configuration cache. In case you need to modify your configuration later on, you can always come back to the module. It will load the last written DAM Lightbox configuration.

Step 4: TypoScript

Now go to the root template of your website. Depending on your setup you have to include several TypoScript snippets in a specific order. Please have a look at the following picture:


Include static (from extensions):

Selected:
DAM Lightbox: basics (damlightbox)
SlimBoxPlus t3m (pmkslimbox)
DAM Lightbox: pmkslimbox (damlightbox)
DAM Lightbox: pages (damlightbox)
DAM Lightbox: tt_content (damlightbox)
DAM Lightbox: tt_news (damlightbox)

Items:
CSS Styled Content (css_styled_content)
CSS Styled Content TYPO3 v3.8 (css_styled_c
CSS Styled Content TYPO3 v3.9 (css_styled_c
CSS Styled Content TYPO3 v4.2 (css_styled_c
Clickenlarge Rendering (rtehtmlarea)
lib.config (cag_yatsl)
lib.page (cag_yatsl)
lib.tmenu.normal (cag_yatsl)
lib.sitelogo (cag_yatsl)
lib.skiplinks (cag_yatsl)

1. You always need to include the snippet "DAM Lightbox: basics" in first place
2. If you have installed pmkslimbox, include the "DAM Lightbox: pmkslimbox" snippet in second place right after the static TypoScript from pmkslimbox. Important: You have to use the SlimBoxPlus variant!
3. Include all table specific DAM Lightbox TypoScript snippets in third place

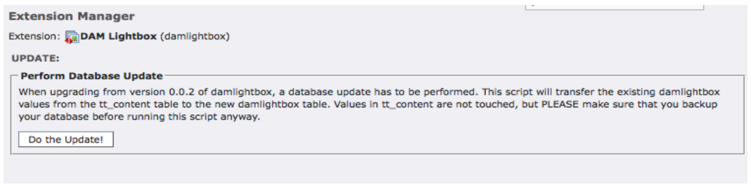
Note:
DAM Lightbox comes with standard libraries for the pages, tt_news, tt_address and tt_content table. If you wish to use DAM Lightbox for another table you have to write your own TypoScript. Don't be afraid, it's quite simple once you understand the basics of this extension. For further information check out the second HowTo in this manual that gives you an example on how to implement DAM Lightbox in another extension.

Deinstallation

1. Drop the TypoScript from your template.
2. Deinstall the extension with EM.
3. Drop the tx_damlightbox_ds table from the database using the COMPARE function of the install tool.

Upgrading from version 0.0.2

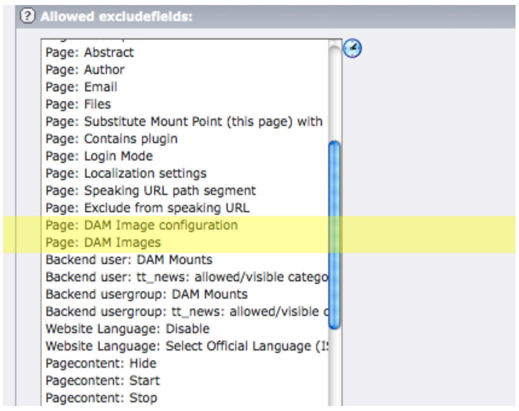
In case you already use an older version of DAM Lightbox (very likely 0.0.2), you will have to perform a database update before you can continue to use the extension. First of all, please make a backup of your database! Next, go to the extension manager, select DAM Lightbox and check the UPDATE! function in the upper left dialogue:


Extension Manager
Extension: DAM Lightbox (damlightbox)
UPDATE:
Perform Database Update
When upgrading from version 0.0.2 of damlightbox, a database update has to be performed. This script will transfer the existing damlightbox values from the tt_content table to the new damlightbox table. Values in tt_content are not touched, but PLEASE make sure that you backup your database before running this script anyway.
Do the Update!

A notice appears that tells you some details about the DB update. Press "Do the Update" and continue with step 3 of the Installation & Configuration section.

Controlling field access

The two generic fields (image & flexform) are defined as exclude fields. Once you have configured them in the DAM Lightbox backend module, they will appear in the exclude list of BE group records for the configured tables. You can exclude/include the generic fields just like you do with dedicated database fields.


Allowed excludefields:
Page: Abstract
Page: Author
Page: Email
Page: Files
Page: Substitute Mount Point (this page) with
Page: Contains plugin
Page: Login Mode
Page: Localization settings
Page: Speaking URL path segment
Page: Exclude from speaking URL
Page: DAM Image configuration
Page: DAM Images
Backend user: DAM Mounts
Backend user: tt_news: allowed/visible catego
Backend usergroup: DAM Mounts
Backend usergroup: tt_news: allowed/visible c
Website Language: Disable
Website Language: Select Official Language (I
Pagecontent: Hide
Pagecontent: Start
Pagecontent: Stop

Defining custom flexforms

You can define custom flexforms for each configured DAM Lightbox table using PageTSConfig. This comes in handy when you need different configuration options on a per table basis. Its also possible to make the field look different for the same table depending on the section of the website. Here is the syntax:

```
TCEFORM.[table].tx_damlightbox_flex.config.ds.default = FILE:path/to/your/custom_flexform.xml
```

Please make sure that you always prepend your path with "FILE:". Your custom flexform needs to be in a correct T3DataStructure format. You can learn more about that in the core documentation. The extension's default flexform (flexform_ds.xml) may also serve as an example.

Note:

Please remember that any of your custom options will need some equivalent TypoScript implementation. Check out the tutorial section of this manual that demonstrates how to implement a custom feature.

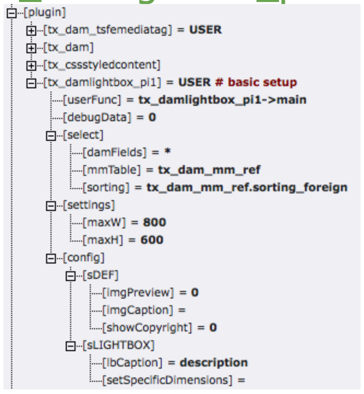
Configuration

DAM Lightbox comes with a set of TypoScript libraries. Because the extension has three areas of functionality (fetching metadata, accessing metadata, displaying metadata) you have to keep three different TS locations in mind. They are:

- **plugin.tx_damlightbox_pi1:** The plugin fetches the metadata for the current record
- **lib.damlightbox:** Here you will find several TypoScript libraries concerned with getting the metadata from TSFE
- **damlightbox:** This pagetype (313) is responsible for the display of images and metadata in a popup/lightbox.

The following sections will give you a detailed overview over each of the three areas.

Fetching metadata: plugin.tx_damlightbox_pi1



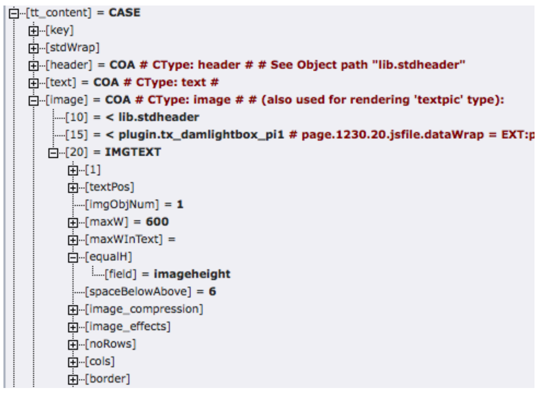
```
[plugin]
  [tx_dam_tsfemediatag] = USER
  [tx_dam]
  [tx_cssstyledcontent]
  [tx_damlightbox_pi1] = USER # basic setup
    [userFunc] = tx_damlightbox_pi1->main
    [debugData] = 0
    [select]
      [damFields] = *
      [mmTable] = tx_dam_mm_ref
      [sorting] = tx_dam_mm_ref.sorting_foreign
    [settings]
      [maxW] = 800
      [maxH] = 600
    [config]
      [sDEF]
        [imgPreview] = 0
        [imgCaption] =
        [showCopyright] = 0
      [sLIGHTBOX]
        [lbCaption] = description
        [setSpecificDimensions] =
```

The plugin part of the extension is responsible for fetching the metadata of any referenced DAM item for the current record. During page rendering it builds a database query for the current record in question and writes its DAM metadata into TSFE. For this to work, the plugin always needs to be executed in the right context. "Context" means that the call to plugin.tx_damlightbox_pi1 should always happen **within** the setup of the TYPO3 record in question but **before** the record itself is rendered. In other words, you always need to execute the plugin in the context of the $cObj->data of the record you want to match with DAM images.

The tricky thing is that the locations will always be a bit different – it depends on how the content generation was implemented for a specific extension. Don't worry to much, due to the fact that DAM Lightbox already has an implementation for tt_content, tt_news, tt_address and pages there are good examples for the different possibilities at hand.

To illustrate this point, please have a look at the following example.

Example: Implementation for tt_content



```
[tt_content] = CASE
  [key]
  [stdWrap]
  [header] = COA # CType: header # # See Object path "lib.stdheader"
  [text] = COA # CType: text #
  [image] = COA # CType: image # # (also used for rendering 'textpic' type):
    [10] = < lib.stdheader
    [15] = < plugin.tx_damlightbox_pi1 # page.1230.20.jsfile.dataWrap = EXT:p
    [20] = IMGTEXT
      [1]
      [textPos]
      [imgObjNum] = 1
      [maxW] = 600
      [maxWInText] =
      [equalH]
        [field] = imageheight
      [spaceBelowAbove] = 6
      [image_compression]
      [image_effects]
      [noRows]
      [cols]
      [border]
```

The image and textpic content elements are configured with a COA object below tt_content.image. On position 10 of the COA, the standard headers are configured. On position 20 there is the IMGTEXT object that will render our images. Keeping the above rule for DAM Lightbox in mind, the correct position to execute plugin.tx_damlightbox_pi1 within the context of tt_content but before the images are rendered is at position 15.

For plugins or pages things work a bit different, but the basic principle always remains the same. Sometimes it's easy to determine the correct position, sometimes you have to think a bit harder. But this is how TYPO3 works anyway ;)

Reference

Constants: plugin.damlightbox (20)

Property:	Data type:	Description:	Default:
damFields	string	Specifies the fields that are fetched from DAM for TSFE display. This can either be all fields ('*') of the DAM table or a comma separated list with specific database fieldnames	*
debugData	boolean	Displays the global register that is filled with query, metadata and config information. Don't forget to adapt the value of devIPmask in the install tool to see the debug output	0
maxW	int	Maximum width of the images in the popup/lightbox and at the same time the basis value for the calculation of the lightbox dimensions	800
maxH	int	Maximum height of the images in the lightbox and at the same time the basis value for the calculation of the lightbox dimensions	600
templateFile	string	Standard template file for the lightbox if pmkslimbox is not used.	EXT:damlightbox/tmpl/standard.html
cssFile	string	The standard CSS file for the popup window if pmkslimbox is not used.	EXT:damlightbox/tmpl/standard.css
jsParams	string	JS parameters: These are the JS parameters for the standard popup window if no lightbox is used	950x780:status=0,menubar=0,scrollbars=1,resizable=1,location=0,directories=0,toolbar=0
imageLinkAttributes	string	Additional link attributes: Here you can enter additional attributes for the links to the popup/lightbox	
imgPreview	int	1st Image is preview: If set to 1 only the first image is shown on the page while the rest is opened in the popup/lightbox. Value will be overridden by default flexform	0
imgCaption	string	Use as imagecaption: 1=media:title, 2=media:caption, 3=media:description. Value will be overridden by default flexform	
showCopyright	int	Show Copyright notice: If set to 1 the copyright field from DAM is prepended to the imagecations. Value will be overridden by default flexform	0
lbCaption	string	Lightbox caption: Use this DAM field for the lightbox/popup caption. Value will be overridden by default flexform	description
headerField	string	The values for the headlines above the imagebrowser are taken from these fields	header // subheader
damTitleWrap	wrap	DAM title wrap: Wraps the DAM title in the lightbox/popup	<span id="damtitle">|</span>
imgBrowserWrap	wrap	Classic imagebrowser: The imagebrowser is only shown in the standard popup window	<div id="imagebrowser">|</div>
prevWord	string	Classic imagebrowser: Label for link to previous image in classic popup	Previous image
nextWord	string	Classic imagebrowser: Label for link to next image in classic popup	Next image
imgCountWrap	wrap	Classic imagebrowser: Wraps the n / n part in the imagebrowser	<span id="imgcount">|</span>
copyrightWrap	wrap	This wraps the copyright notice in the caption and the lightbox	<span class="copyright">|</span>
damLocationWrap	wrap	DAM location wrap: Wraps the DAM location in the lightbox	<span id="damlocation">|</span>

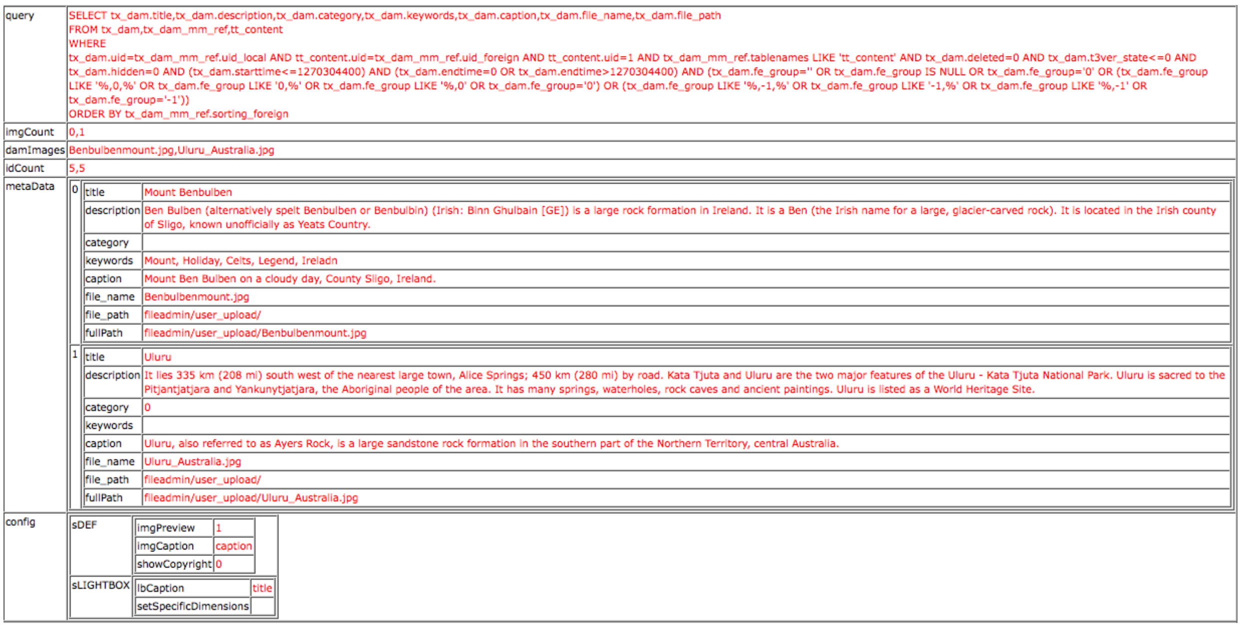
Setup: plugin.tx_damlightbox_pi1

Property:	Data type:	Description:	Default:
select.damFields	string	Can also be set with constant editor. Comma separated list of fields from the DAM table that should be fetched by the database query.	*
select.mmTable	string	Tablename of the MM reference table that is used in the select query. Should normally be tx_dam_mm_ref. But who knows, maybe there are situations where this needs to be configured... ;)	tx_dam_mm_ref
select.foreignTable	string/stdWrap	Tablename to which the record that is currently being rendered belongs. The "local" table is always tx_dam, the foreign table could be any other TYPO3 table.	
select.whereClause	string/stdWrap	Sometimes it might be necessary to influence the WHERE part of the MM query. **Example:** `AND tx_dam_mm_ref.tablenames='tx_my_extension' AND tx_my_extension.uid={field:uid}`	
select.sorting	string	DB field used for sorting the result of the query	tx_dam_mm_ref.sorting
settings.		Below this property are the maxW and maxH values that are used to determine the max width/height of the images in the popup/lightbox and serve as a base for the calculation of the lightbox dimensions.	

config.		This property is the TS equivalent for the generic flexform field. Below this property you can set standard values that will be used if the flexform field of the current record holds no value. The notation used here reflects the XML structure of the flexform:
		`config.SHEETNAME.field = value`
		Example:
		`config.sDEF.imgPreview = 0`
		This sets a standard value for the "First image as preview" field in the default flexform.	

Accessing metadata: lib.damlightbox

During page rendering plugin.tx_damlightbox_pi1 fetches the DAM metadata of the images that belongs to the current record and writes it into the global register tx_damlightbox. This ensures that all DAM data is globally accessible with TypoScript. The following picture (produced with plugin.tx_damlightbox_pi1.debugData = 1) illustrates this:



```
query      SELECT tx_dam.title,tx_dam.description,tx_dam.category,tx_dam.keywords,tx_dam.caption,tx_dam.file_name,tx_dam.file_path
           FROM tx_dam,tx_dam_mm_ref,tt_content
           WHERE
           tx_dam.uid=tx_dam_mm_ref.uid_local AND tt_content.uid=tx_dam_mm_ref.uid_foreign AND tt_content.uid=1 AND tx_dam_mm_ref.tablenames LIKE 'tt_content' AND tx_dam.deleted=0 AND tx_dam.t3ver_state<=0 AND
           tx_dam.hidden=0 AND (tx_dam.starttime<=1270304400) AND (tx_dam.endtime=0 OR tx_dam.endtime>1270304400) AND (tx_dam.fe_group='' OR tx_dam.fe_group IS NULL OR tx_dam.fe_group='0' OR (tx_dam.fe_group
           LIKE '%,0,%' OR tx_dam.fe_group LIKE '0,%' OR tx_dam.fe_group LIKE '%,0' OR tx_dam.fe_group='0') OR (tx_dam.fe_group LIKE '%,-1,%' OR tx_dam.fe_group LIKE '-1,%' OR tx_dam.fe_group LIKE '%,-1' OR
           tx_dam.fe_group='-1'))
           ORDER BY tx_dam_mm_ref.sorting_foreign
imgCount   0,1
damImages  Benbulbenmount.jpg,Uluru_Australia.jpg
idCount    5,5
metaData   0  title        Mount Benbulben
              description  Ben Bulben (alternatively spelt Benbulben or Benbulbin) (Irish: Binn Ghulbain [GE]) is a large rock formation in Ireland. It is a Ben (the Irish name for a large, glacier-carved rock). It is located in the Irish county
                           of Sligo, known unofficially as Yeats Country.
              category
              keywords     Mount, Holiday, Celts, Legend, Ireladn
              caption      Mount Ben Bulben on a cloudy day, County Sligo, Ireland.
              file_name    Benbulbenmount.jpg
              file_path    fileadmin/user_upload/
              fullPath     fileadmin/user_upload/Benbulbenmount.jpg
           1  title        Uluru
              description  It lies 335 km (208 mi) south west of the nearest large town, Alice Springs; 450 km (280 mi) by road. Kata Tjuta and Uluru are the two major features of the Uluru - Kata Tjuta National Park. Uluru is sacred to the
                           Pitjantjatjara and Yankunytjatjara, the Aboriginal people of the area. It has many springs, waterholes, rock caves and ancient paintings. Uluru is listed as a World Heritage Site.
              category     0
              keywords
              caption      Uluru, also referred to as Ayers Rock, is a large sandstone rock formation in the southern part of the Northern Territory, central Australia.
              file_name    Uluru_Australia.jpg
              file_path    fileadmin/user_upload/
              fullPath     fileadmin/user_upload/Uluru_Australia.jpg
config     sDEF       imgPreview      1
                      imgCaption      caption
                      showCopyright   0
           sLIGHTBOX  lbCaption              title
                      setSpecificDimensions
```

Three areas are relevant:

- **query:** This holds the query that was built to fetch the metadata for the current record. Useful for debugging purposes
- **metadata:** The sets of DAM metadata that were fetched for the images of the current record. They are stored using the imagenumber from the record as key (starting with "0")

- **config:** This config array holds the values of the backend flexform of the current record merged with the standard values set from TypoScript

TSref tells us that we can read values from an array using the “pipe syntax”. You can therefore get the metadata for the images by using the following notation:

```
TSFE:register|tx_damlightbox|metaData|IMAGENUMBER|FIELDNAME
```

To make life easier, DAM Lightbox comes with a predefined TypoScript library that does this job for you. Its called lib.getDAMvalues:

```
lib.getDAMvalues = TEXT
lib.getDAMvalues {
	wrap = {|}
	cObject = TEXT
	cObject {
		value = TSFE:register|tx_damlightbox|metaData|IMAGENUMBER|FIELDNAME
		insertData = 1
	}
	insertData = 1
	htmlSpecialChars = 1
}
```

It consists of two nested TEXT objects. The inner one dynamically builds the string that is needed to fetch the values from the TSFE array using the insertData property. The outer one executes the inner TEXT object using insertData property.

Example: Set the DAM description as caption in tt_content

We want to insert the DAM description as caption for each tt_content image. We do:

```
tt_content.image.20.caption {
	10 < lib.getDAMvalues
	10.cObject.value = TSFE:register|tx_damlightbox|metaData|{register:IMAGE_NUM_CURRENT}|description
}
```

In this example we used the IMAGE_NUM_CURRENT register that is available in tt_content during parsetime to determine the right metadata set. In other rendering context you don't have this value at hand. In that case you need to check the extension's documentation. Maybe it supplies a similar variable like IMAGE_NUM_CURRENT when it generates it's images.

If that's not the case then you can use the **frontendImageIterator** function from DAM Lightbox. Have a look at the second tutorial in this manual that focuses on how to implement the DAM Lightbox functionality for another extension. The usage of frontendImageIterator is explained there.

Reference

Setup: lib.damlightbox.

Property:	Data type:	Description:	Default:

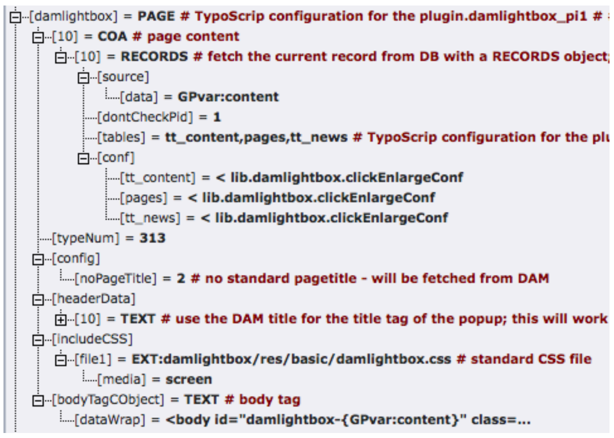
getDAMvalues	TEXT	TypoScript text object that retrieves the metadata values from TSFE. Needs to know the image number for which to retrieve the metadata.
		Example:
		`tx_my_extension.imagecaption < lib.getDAMvalues`
		`tx_my_extension.imagecaption.cObject.value = TSFE:register|tx_damlightbox|metaData|{register:currentImg}|caption`	

imageRegisters	LOAD_REGISTER	Register used in the popup/lightbox to determine which click-enlarged image is shown	
clickEnlargeConf	COA	This library controls the HTML content for click enlargement - usable within a configured lightbox or the classic image popup. At position 5 of the library the damlightbox plugin is executed. At position 10 some registers are filled with the current image numbers. The most interesting part is position 20 where you'll find a TEMPLATE object. This is the actual lightbox/popup content that can be adapted to your needs	

Click-Enlargement: pagetype 313

Why 313? Well, its the number plate of Donald Duck's car... :) And in our case its also the typeNum of the damlightbox PAGE object. Regardless if you use pmkslimbox or if just open the images in a classic popup, the basic principle remains is the same: The click-enlarged images are opened using a PAGE object that is called by pagetype 313. This pagetype is responsible for displaying the lightbox/popup content.

In comparison to the classic TYPO3 showpic function this approach has the advantage that every part of the HTML content of the popup can be influenced. For opening a DAM image in the DAM Lightbox pagetype three parameters are relevant:



```
[damlightbox] = PAGE # TypoScrip configuration for the plugin.damlightbox_pi1 # 
  [10] = COA # page content
    [10] = RECORDS # fetch the current record from DB with a RECORDS object;
      [source]
        [data] = GPvar:content
      [dontCheckPid] = 1
      [tables] = tt_content,pages,tt_news # TypoScrip configuration for the pl
      [conf]
        [tt_content] = < lib.damlightbox.clickEnlargeConf
        [pages] = < lib.damlightbox.clickEnlargeConf
        [tt_news] = < lib.damlightbox.clickEnlargeConf
  [typeNum] = 313
  [config]
    [noPageTitle] = 2 # no standard pagetitle - will be fetched from DAM
  [headerData]
    [10] = TEXT # use the DAM title for the title tag of the popup; this will work
  [includeCSS]
    [file1] = EXT:damlightbox/res/basic/damlightbox.css # standard CSS file
      [media] = screen
  [bodyTagCObject] = TEXT # body tag
    [dataWrap] = <body id="damlightbox-{GPvar:content}" class=...
```

1. **type:** This makes TYPO3 using the DAM Lightbox pagetype for record display
2. **content:** This contains the record & table that should be displayed
3. **img:** This is the image number that should be displayed

Example: How a link should look like

```
http://mywebsite.com/index.php?id=5&type=313&content=tt_content_2&img=0&cHash=ef8a779cdc
```

Here the first image of the record with the uid 2 from table tt_content is opened using pagetype 313. The content is cached using a cHash parameter. Its best to use the typolink function either from TypoScript or from your extension to build the links to the DAM Lightbox pagetype. In TypoScript its very easy:

Example: Building a link to the DAM Lightbox pagetype using TypoScript

```
imageLinkWrap = 1
imageLinkWrap {
      enable = 1
      typolink {

            ## caching ##
            no_cache = 0
            useCacheHash = 1

            parameter.data = TSFE:id

            additionalParams.cObject = TEXT
            additionalParams.cObject {
                  htmlspecialchars = 1
                  dataWrap = |&type=313&content=tt_content_{field:uid}&img={register:IMAGE_NUM_CURRENT}
            }
      }
}
```

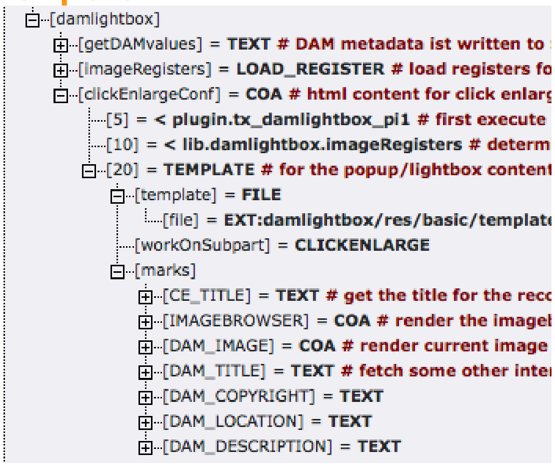
If you look at the TypoScript configuration for pagetype 313 you can see that the content is fetched using a RECORDS object. This makes the whole approach quite versatile because all that is needed to display any record from any table in the popup is to register the table in question in the RECORDS object and to give it some configuration. In the above screenshot you can see how this is done for the tt_content, tt_news and pages table.

Example: Register my table in the click-enlargement pagetype

```
damlightbox {
      10.10.tables := addToList(tx_mytable)
      10.10.conf.tx_mytable =< lib.damlightbox.clickEnlargeConf
}
```

When you include lib.damlightbox.clickEnlargeConf you will have a TEMPLATE object at hand that enables you to adapt the popup/lightbox content completely to your liking.

Working with the lightbox template



```
[damlightbox]
  [getDAMvalues] = TEXT # DAM metadata ist written to
  [imageRegisters] = LOAD_REGISTER # load registers fo
  [clickEnlargeConf] = COA # html content for click enlarg
    [5] = < plugin.tx_damlightbox_pi1 # first execute
    [10] = < lib.damlightbox.imageRegisters # determ
    [20] = TEMPLATE # for the popup/lightbox content
      [template] = FILE
        [file] = EXT:damlightbox/res/basic/templat
      [workOnSubpart] = CLICKENLARGE
      [marks]
        [CE_TITLE] = TEXT # get the title for the reco
        [IMAGEBROWSER] = COA # render the imageb
        [DAM_IMAGE] = COA # render current image
        [DAM_TITLE] = TEXT # fetch some other inte
        [DAM_COPYRIGHT] = TEXT
        [DAM_LOCATION] = TEXT
        [DAM_DESCRIPTION] = TEXT
```

The DAM Lightbox template object uses the good old ###MARKER### approach. Example templates are in the res/ folder of the extension. The default template defines some example markers in the HTML template and the according TypoScript. The rest is up to you. If you want a new marker for displaying a particular DAM field, just add a marker name of your choice to the HTML template and implement the according markername in TS.

Example: Adding a custom marker to the HTML template

We want a new marker for showing the creator of each image.

1) We insert a marker ###DAM_CREATOR### in our HTML template.

2) We adapt the TypoScript:

```
lib.damlightbox.clickEnlargeConf.20.marks.DAM_CREATOR {

       10 < lib.damlightbox.getDAMvalues
       10.cObject.value = TSFE:register|tx_damlightbox|metaData|{register:currentImg}|creator
       10.outerWrap = <span id="creator">|</span>

}
```

That's it. Sometimes it doesn't make sense to change lib.damlightbox.clickEnlargeConf because this affects the global configuration of the extension. If you want to display the creator field only for a specific table, you could adapt the configuration like this:

```
damlightbox.10.10.conf.tx_mytable.20.marks.DAM_CREATOR.[...]
```

Reference

Setup: damlightbox (pagetype 313)

Property:	Data type:	Description:	Default:
config.		CONFIG object for the DAM Lightbox pagetype. See TSRef for the properties.	
headerData	COA	See TSRef. At position 10 the title tag is set to the title of the DAM image using a TEXT object.	
includeCSS.file1	string	Here the default CSS file for the popup/lightbox content is included. You'll find the file in the res/ folder of the extension. Custom CSS files can be set either here or using the Constant Editor	
bodyTagCObject	TEXT	The bodytag of a HTML popup/lightbox always gets a unique id attribute ("CSS signature") so that it's possible to style it with CSS	
10.10	RECORDS	This RECORDS object will render the click-enlarged image record. See TSRef for possible properties of this TS object. Your custom tables have to be registered in the "table" property using **addToList**, the configuration for your custom tables has to be inserted below the **conf** property. I recommend referencing lib.damlightbox.clickEnlargeConf here since it contains the basic functionality.	

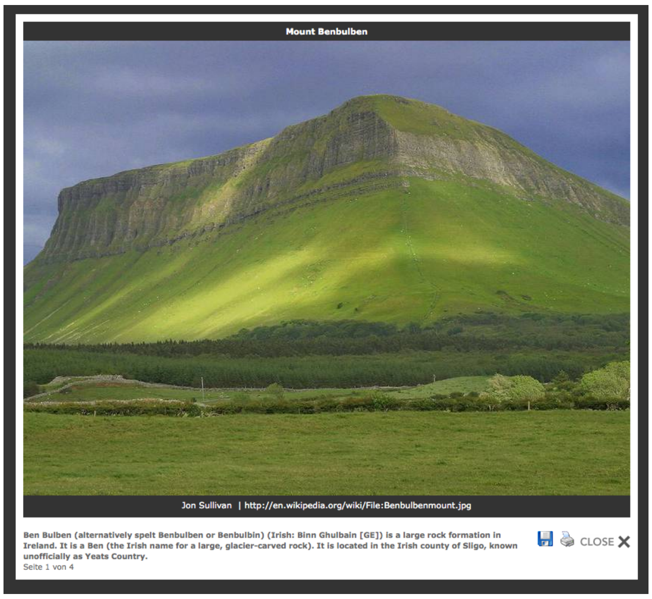
pmkslimbox

There are many very good lightbox in extensions in the TER. Some of them based on jquery, some of them based on mootools, some of them based on other cool JavaScript libraries. Why pmkslimbox as implementation example for DAM

Lightbox? Well,

1. It's coding style is very clean and it is very configurable with TypoScript
2. It offers to open up external HTML content within an iframe, which is a crucial feature for the click-enlargement strategy of DAM Lightbox
3. Its very stable, cross browser and can be used together with t3mootools
4. Its impossible to do justice to all ;)

That doesn't mean that its not possible to customize other lightbox extension for DAM Lightbox. In most cases you will only need to study how a specific lightbox extension utilizes the a tag parameters and/or link attributes that lead to the activation of it's JavaScript functionality. Once you know, its only peace of cake to adapt the image links generated by DAM Lightbox to fit the specific needs of your lightbox extension. In future versions it is planned to add support to other lightbox extensions from TER besides pmkslimbox. Just send in your candidates or do a feature request on the TYPO3 forge.



Jon Sullivan | http://en.wikipedia.org/wiki/File:Benbulbenmount.jpg

Ben Bulben (alternatively spelt Benbulben or Benbulbin) (Irish: Binn Ghulbain [GE]) is a large rock formation in Ireland. It is a Ben (the Irish name for a large, glacier-carved rock). It is located in the Irish county of Sligo, known unofficially as Yeats Country.
Seite 1 von 4

This screenshot shows pmkslimbox in action together with damlightbox. The caption besides the print and save buttons was generated from the DAM description field and inserted into the title attribute of the link tag. The content inside pmkslimbox (title, image, creator, url) is "external" HTML content generated by pagetype 313 and opened inside an iframe. The print and save buttons reference the original file from the fileadmin directory.

Image width/height and pmkslimbox

In some situations the width/height of your DAM images might become important. Normally the slimbox script uses the width/height attribute of the click-enlarged images to dynamically fit itself to the given proportions. But in our case we open TYPO3 pages with images and additional text content. In some situations this might result in a lightbox that doesn't seemlessly fit it's content (for instance when you have portrait images and a very very long DAM title).

Since the height that the text might consume is not really calculable, a "width/height bonus" is sometimes necessary to make the lightbox fit seamlessly. This is what can be achieved with the "Custom dimensions" field in the default flexform. Here you can enter a custom size using the notation "Nr:width,height;".

Reference

Constants: plugin.damlightbox.pmkslimbox (4)

Property:	Data type:	Description:	Default:
templateFile	string	Path to the HTML template that is used for pmkslimbox content	EXT:damlightbox/res/pmkslimbox/template.html
cssFile	string	Path to the CSS file that is used for pmkslimbox content	EXT:damlightbox/res/pmkslimbox/damlightbox.css
hCalc	int/wrap + calc	The value will be added to the horizontal lightbox dimensions.	
vCalc	int/wrap + calc	The value will be added to the vertical lightbox dimension. Used to create some space above and below the image.	|+54

Setup: lib.damlightbox.pmkslimbox.

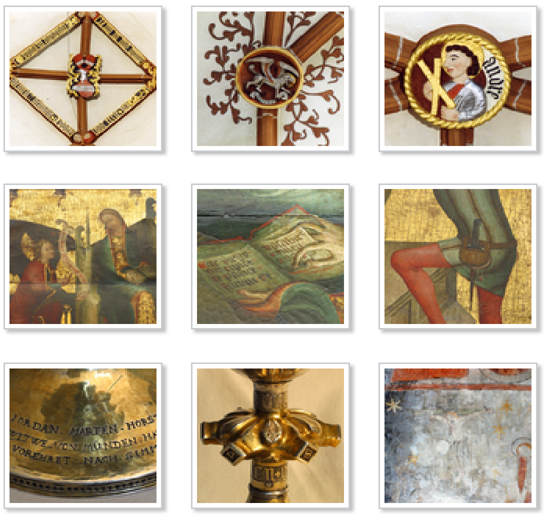
Property:	Data type:	Description:	Default:
imageLinkWrap.typolink	typolink	Below this property are the necessary changes to ATagParams that are needed for making DAM Lightbox work with pmkslimbox. You can use this library in your own extensions. **Example:** `myImage.imageLinkWrap.typolink < lib.damlightbox.pmkslimbox.imageLinkWrap.typolink.ATagParams`	

Tutorials & Examples

How to: Implement a custom flexform feature

Task: Square thumbnails for tt_content image & text/w image elements

We would like to give our editors the possibility to generate square thumbnails for all DAM images in tt_content records. For this reason we would like to include an additional checkbox in the generic flexform field. Once checked, the result for the content element should look like this:



Step 1: Customizing the flexform

Go to the extension's folder an copy the file flexform_ds.xml to your fileadmin directory. Rename the copy to tt_content_flexform.xml. Open up the copied file with an editor of your choice. We want the additional checkbox on the “Image configuration” tab. Therefore go to the <sDEF> part of the flexform file just before the closing </el> tag and insert the following XML.

Example:

```
<showSquares>
      <TCEforms>
            <label>LLL:fileadmin/custom_flexform_locallang.xml:tx_damlightbox.showSquares</label>
            <config>
                  <type>check</type>
            </config>
      </TCEforms>
</showSquares>
```

Next you need to create a language file in which you keep the labels for your customized flexform. Best to put it in fileadmin right next to the customized flexform. Name it custom_flexform_locallang.xml, open it and insert the following XML:

Example:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<T3locallang>
      <meta type="array">
            <type>database</type>
            <description>Language labels for the flexform belonging to damlightbox</description>
      </meta>
      <data type="array">
            <languageKey index="default" type="array">
                  <label index="tx_damlightbox.showSquares">Square thumbnails</label>
            </languageKey>
            <languageKey index="de" type="array">
                  <label index="tx_damlightbox.showSquares">Quadratische Thumbnails</label>
            </languageKey>
      </data>
</T3locallang>
```

Finally we need to specify the custom flexform for tt_content using PageTSConfig. Go to the rootpage of your website and insert the following TS into the PageTSConfig field:

```
TCEFORM.tt_content.tx_damlightbox_flex.config.ds.default = FILE:fileadmin/tt_content_flexform.xml
```

Now open up a content element and check if the customized flexform is correctly loaded. If that's the case, insert some example images, set 100px as width and activate the new checkbox. You should see the images in the frontend, but so far nothing happens. In the next step we're going to implement the functionality for our new flexform feature with TypoScript.

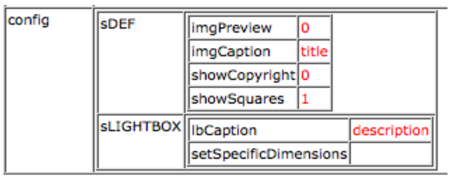
Step 2: Implementing the functionality

Generating square thumbnails from images is fairly easy using the width/height properties of the IMG_RESOURCE object. TSRef tells us that we can crop scale images from the middle by appending a "c" to the pixel values. We know that the width and height of tt_content images are set in tt_content.image.20.1.file. To achieve the desired effect we need to change these values depending on the status of our custom flexform checkbox.

To find out what values we have at hand it's always a good idea to turn on the debug feature of DAM Lightbox. Set

```
plugin.damlightbox.debugData = 1
```

in the constant field of your root template. Reload the page and you'll see some interesting information. Besides the metadata for the images you will find a section labeled "config" in the debug output:



config	sDEF	imgPreview	0
		imgCaption	title
		showCopyright	0
		showSquares	1
	sLIGHTBOX	lbCaption	description
		setSpecificDimensions	

When plugin.tx_damlightbox_pi1 is called it takes the standard config values set from TypoScript and the values from the current flexform and merges them into one config array that equals the structure of the flexform in question. This means that any property that you define in your flexform will be available under the same name in the frontend register. In our case we now have a boolean property "showSquares" available.

From TSref we learn that we can read values from an array using the "pipe" syntax. So to get our value we have to write:

```
TSFE:register|tx_damlightbox|config|sDEF|showSquares
```

Now it becomes clear how our feature can be implemented. Insert the following into your TS template.

Example:

```
## first clear the equalH object; its in our way because it overrides dimension settings ##
tt_content.image.20.equalH >

## implement new dimensions ##
tt_content.image.20.1.file {

      # clear old calculation
      width >

      # new width calculation including crop scaling depending on showSquares value
      width.cObject = TEXT
      width.cObject {
            field = imagewidth
            outerWrap = |c
            outerWrap.if.isTrue.data = TSFE:register|tx_damlightbox|config|sDEF|showSquares
      }

      # new height object with two possibilities
      height.cObject = COA
      height.cObject {

            # 1: in case showSquares is not checked use the height field
            10 = TEXT
            10 {
                  field = imageheight
                  if.isFalse.data = TSFE:register|tx_damlightbox|config|sDEF|showSquares
            }

            # 2: in case showSquares is checked take the width as basis and do crop scaling
            20 = TEXT
            20 {
                  field = imagewidth
                  wrap = |c
                  if.isTrue.data = TSFE:register|tx_damlightbox|config|sDEF|showSquares
            }
      }
}
```

Next time you reload the page, you will have nice square thumbnails 100x100px in size.

How to: DAM Lightbox & other extensions

Due to the fact that almost every functionality of DAM Lightbox is customizable with TypoScript, in most cases it becomes straightforward to connect other extensions it. The following tutorial is a brief step-by-step guide that shows you an example implementation for the tt_address table. Since every extension is coded differently, the steps might sometimes be a bit different. Nevertheless tt_address can serve as a good example to illustrate the basic principles.

Step 1: tt_address installation

Please install the latest tt_address version from TER. Open your localconf.php and make sure that damlightbox comes after tt_address in the extList variable:

```
$TYPO3_CONF_VARS['EXT']['extList'] = '...OTHER EXTS...,tt_address,damlightbox';
```

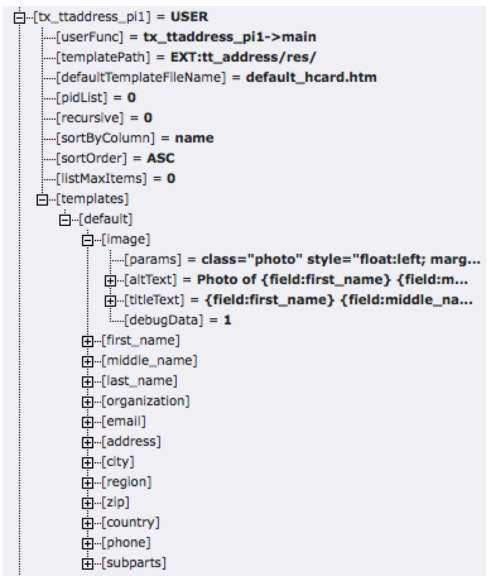
Now that we have tt_address installed, add a page somewhere in your pagetree. Insert a +ext template on that page and include the static TypoScript for the tt_address extension. In our tutorial we will work with that template and the example page.

Step 2: Generic field configuration

Go to the DAM Lightbox backend module. Choose the "Address" table in the table selection and press "Refresh configuration". The detail configuration for the tt_address table appears. Check both generic fields, leave the types configuration untouched and choose "Image" in the select box specifying the field order. Press "Write configuration". Now go to your example page and create 3-5 test records with DAM images. The generic DAM Lightbox fields should appear right after the classic image field of the tt_address records. Once that is done insert a tt_address plugin on the example page and insert the address records that you just created.

Step 3: Analysis

Now we check the way tt_address handles it's image generation to find out the right spot to connect DAM Lightbox.



```
[tx_ttaddress_pi1] = USER
    [userFunc] = tx_ttaddress_pi1->main
    [templatePath] = EXT:tt_address/res/
    [defaultTemplateFileName] = default_hcard.htm
    [pidList] = 0
    [recursive] = 0
    [sortByColumn] = name
    [sortOrder] = ASC
    [listMaxItems] = 0
    [templates]
        [default]
            [image]
                [params] = class="photo" style="float:left; marg...
                [altText] = Photo of {field:first_name} {field:m...
                [titleText] = {field:first_name} {field:middle_na...
                [debugData] = 1
            [first_name]
            [middle_name]
            [last_name]
            [organization]
            [email]
            [address]
            [city]
            [region]
            [zip]
            [country]
            [phone]
            [subparts]
```

At first glance it looks very promising. The ###IMAGE### marker is filled with a standard IMAGE object and offers full stdWrap support. But looking into the PHP class of tt_address reveals that this IMAGE object uses a hardcoded path to "uploads/pics". Unfortunately this makes the marker unusable for our purpose since our images are in fileadmin and referenced by DAM. If the path was not hardcoded, we could have done the whole implementation just using TypoScript. Luckily, a few lines down in the PHP class we see that tt_address offers to insert custom markers with a hook function. We will therefore create a marker ###DAM_IMAGE### and use it with DAM Lightbox.

Step 4: Hooking tt_address

First create a new file in your fileadmin folder and save it as class.tx_damlightbox_ttaddress.php.inc. Next, open your localconf.php and register the hook:

```
// tt_address hook
$TYPO3_CONF_VARS['EXTCONF']['tt_address']['extraItemMarkerHook']['tx_damlightbox_pi1'] =
'fileadmin/class.tx_damlightbox_ttaddress.php.inc:tx_damlightbox_ttaddress';
```

We see from the tt_address class that the hook function should be called "extraItemMarkerProcessor". In our new hook file we therefore put the following code:

```
class tx_damlightbox_ttaddress extends tx_ttaddress_pi1 {
	public function extraItemMarkerProcessor($markerArray, $address, &$lConf, &$pObj) {

		// prepare the context: set the address record as current cObject
		$pObj->cObj->data = $address;
		$pObj->cObj->currentRecord = 'tt_address:'.$address['uid'];

		// insert marker for DAM images with stdWrap properties
		$markerArray['###DAM_IMAGE###'] = $pObj->cObj->stdWrap($lConf['dam_image.'],
$lConf['dam_image.']);

		return $markerArray;
	}
}
```

It has already been mentioned that plugin.tx_damlightbox_pi1 always needs the current table/record ($cObj->data) context to build the correct MM query which joins DAM and the current record. Within our hook function we therefore feed the current address record to the $cObj->data array of the calling parent object. Then we set the current record to be from tt_address appending it's uid with a colon. Now the proper context is prepared. Note that we use the stdWrap function to have full flexibility.

That finished, open your tt_address HTML template and insert the new marker ###DAM_IMAGE### instead of the ###IMAGE### marker. Refresh the page. The marker gets displayed, but is not yet filled with content. This is what we're going to do in the next step.

Step 5: Connect DAM Lightbox with TypoScript

Since we have our new marker in place we can come to the fun part: connecting DAM Lightbox using TypoScript. Go to your +ext template on the example page. In our hook function we configured that everything below [dam_image.] will be parsed through stdWrap. Remember: plugin.tx_damlightbox_pi1 needs to be executed first, then we can work on the DAM metadata that has been fetched. Its therefore best to use a COA object:

```
plugin.tx_ttaddress_pi1.templates.default.dam_image.cObject = COA
plugin.tx_ttaddress_pi1.templates.default.dam_image.cObject {

	10 =< plugin.tx_damlightbox_pi1
	10.debugData = 1
}
```

As soon as you refresh the page, you'll see lots of debug output. You can see the database query that was built, and, if things were configured correctly, you also see the metadata sets for the images referenced in the tt_address records.

Now we need a TypoScript object to generate the images. Let's reflect on that for a moment. It could very well be that there is more than one image referenced by an address record. In that case, we need to iterate through all images of the record and access the metadata belonging to it by the current image number (just like it's done in tt_content using IMAGE_NUM_CURRENT).

class.tx_damlightbox_pi1.php provides a handy function that can be called from TypoScript. It will do the iteration and image generation for us: **frontendImageIterator**. Since the metadata is fetched at position 10, we continue with a USER object that calls the function at position 20:

```
plugin.tx_ttaddress_pi1.templates.default.dam_image.cObject {
20 = USER
20 {
	userFunc = tx_damlightbox_pi1->frontendImageIterator

	file {
		import.cObject < lib.damlightbox.getDAMvalues
		import.cObject {
			cObject.value = TSFE:register|tx_damlightbox|metaData|{register:currentImg}|fullPath
		}
		maxW = 300
		maxH = 200
	}

	## alt texts from DAM ##
	altText.cObject < lib.damlightbox.getDAMvalues
	altText.cObject.cObject.value  = TSFE:register|tx_damlightbox|metaData|{register:currentImg}|
alt_text

	## title texts from DAM ##
	titleText.cObject < lib.damlightbox.getDAMvalues
	titleText.cObject.cObject.value  = TSFE:register|tx_damlightbox|metaData|{register:currentImg}|
title

	## image link for click enlarge ##
	imageLinkWrap = 1
	imageLinkWrap {
		enable = 1
```

```
				typolink {

					## caching ##
					no_cache = 0
					useCacheHash = 1

					parameter.data = TSFE:id

					parameter.append = TEXT
					parameter.append {
						if.isFalse.field = image_link
						value = {$plugin.damlightbox.jsParams}
						noTrimWrap = |  ||
					}

					additionalParams.cObject = TEXT
					additionalParams.cObject {
						htmlspecialchars = 1
						dataWrap = |&type=313&content=tt_address_{field:uid}&img={register:currentImg}
					}

					ATagParams.cObject = TEXT
					ATagParams.cObject {
						value = {$plugin.damlightbox.imageLinkAttributes}
					}
				}
			}
}
```

When you refresh the page, you will now see all DAM images from the tt_address records.

What's still missing is the click-enlargement. Before we do that, just some explanations to the above TypoScript. The USER object acts just like a normal IMAGE object. All configuration values from TSref will work with a frontendImageIterator object.

Also note how lib.damlightbox.getDAMvalues is used to access the according metadata in the TSFE register using the currentImg register that is handled by the frontendImageIterator function.

And finally have a look at how the additional params which are responsible for opening up the DAM image in a popup/lightbox context are set:

- The pagetype is specified (313),
- the table context is set (tt_address),
- the uid of the current record is dynamically inserted in the parameter,
- and finally the current image number is appended.

Step 6: pagetype configuration

Now we need to configure the damlightbox pagetype. This is quickly done:

```
## register tt_address table in the click-enlargement pagetype ##
damlightbox {
	10.10.tables := addToList(tt_address)
	10.10.conf.tt_address =< lib.damlightbox.clickEnlargeConf
}
```

First we register the tt_address table in the table property of the RECORDS object that is used by damlightbox. Next we reference the click-enlargement library for the tt_address table. Refresh the page and click on one of the DAM images. They will now open in the “classic” image popup.

Step 7: Lightbox configuration

Since we want to open the images in the much nicer pmkslimbox, this is our last step. For pmkslimbox you should have included the DAM Lightbox: pmkslimbox script in your root template. To activate pmkslimbox, we need to modify the link parameters generated by DAM Lightbox. Using the appropriate DAM Lightbox library, this also becomes quite easy:

```
plugin.tx_ttaddress_pi1.templates.default.dam_image.cObject {

	20.imageLinkWrap.typolink {

		parameter.append >

		ATagParams >
		ATagParams < lib.damlightbox.pmkslimbox.imageLinkWrap.typolink.ATagParams
	}
	stdWrap.postCObject = TEXT
	stdWrap.postCObject {
```

```
            if.isTrue.data = TSFE:register|tx_damlightbox|config|sDEF|imgPreview
            cObject = USER
            cObject.userFunc = tx_damlightbox_pmkslimbox->addHiddenImgs
            cObject {
                  content.data = field:uid
                  content.table = tt_address
                  hCalc.cObject = TEXT
                  hCalc.cObject.value = {$plugin.damlightbox.pmkslimbox.hCalc}
                  vCalc.cObject = TEXT
                  vCalc.cObject.value = {$plugin.damlightbox.pmkslimbox.vCalc}
            }
      }
}

damlightbox {
      10.10.conf.tt_address.20 {

            template.file = {$plugin.damlightbox.pmkslimbox.templateFile}
            marks.DAM_TITLE.outerWrap >
      }
}
```

We simply kick out the parameters that are used for the classic popup and reference the imageLinkWrap configuration that is provided by lib.damlightbox.pmkslimbox. But what's that stdWrap object? Well, if several images are appended to an address record and firstImageIsPreview is checked, the other images need to be inserted into a hidden div in the HTML, otherwise they will not be browsable when pmkslimbox is opened. This is what the stdWrap.postCObject does.

Conclusion

Even if all this looks a bit tricky at first glance, it becomes quite feasible once you remember the basic principles of DAM Lightbox. Since there exist so many different ways how TYPO3 extensions work, I rather doubt that one will ever find THE golden way that will be valid for all conceivable scenarios regarding DAM and TYPO3. This is the reason why DAM Lightbox tries to utilize the one constant factor that is at hand throughout the whole system: TypoScript.

Known problems

Nothing that can't be fixed ;). Suggestions and improvements always welcome. If you find an issue, please report it in the extension's bugtracker:

http://forge.typo3.org/projects/extension-damlightbox/issues

To-Do list

Check out the roadmap of the extension on the TYPO3 forge:

http://forge.typo3.org/wiki/extension-damlightbox

ChangeLog

Also check out the recent development activities on forge: http://forge.typo3.org/projects/activity/extension-damlightbox

Version:	**Changes:**
Beta branch	
0.1.0	Rewritten manual with examples (#6053) Generic damlightbox field for any backend table (#6060, #6075, #6533) Backend module for damlightbox configuration (#6398) Sample integration of damlightbox for tt_content (#6055), for tt_news (#6056), for pages (#6057), for tt_address Reintegration of pmkslimbox (#6054) Print & Save function for pmkslimbox (#3261, #3689) Bugfix for t3_mootools issue (#3645) Update Script to migrate flexform field from tt_content to tx_damlightbox_ds (#6438)
Alpha branch	
0.0.2	Tiny bugfix: Wrong sorting field was set as standard value in static TS
0.0.1	First public version. Alpha, but fully usable.