Documentation

pt_extlist

Table of Contents

Introduction

A documentation for users, administrators and developers.

The users will learn how to install and setup a demo list. The administrators will learn setting up a list and plugin configuration. The developers will learn what pt_extlist is about and how they can manipulate pt_extlist.

The users and administrators need the basic knowledge about programming.

What does it do?

This extension is intended to generate all sorts of lists. The data sources for the list can be a database or an extbase repository or anything you write a data-backend for (SOAP, CSV, XML, ...).

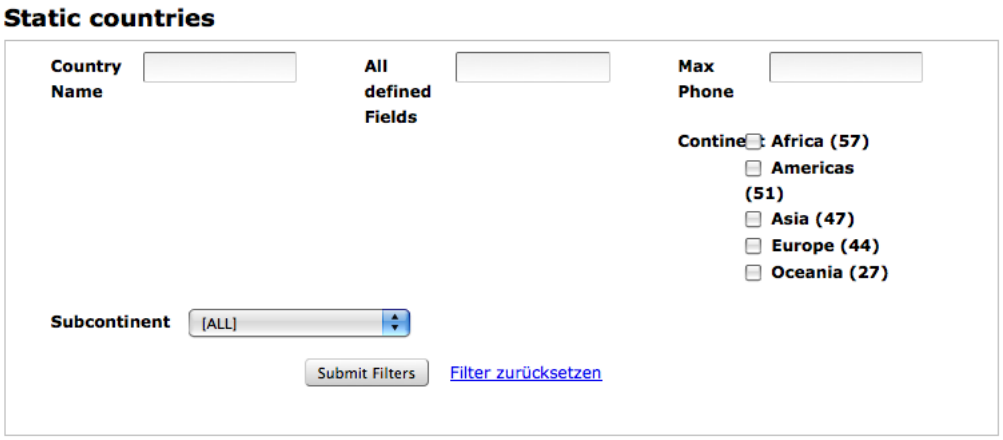
There are different steps you have to do if you want to set up a list. For a detailed example see section "Instruction".

Here are some screenshots to give you an impression of how it looks like.

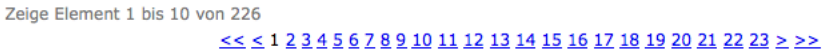
Country name (local country name)	Capital	ISO	Phone	Continent
Andorra Details	Andorra la Vella	AD	376	Europe
Afghanistan (افغانستان) Details	Kabul	AF	93	Asia
Antigua and Barbuda Details	St John's	AG	1268	Americas
Anguilla Details	The Valley	AI	1264	Americas
Albania (Shqipëria) Details	Tirana	AL	355	Europe
Netherlands Antilles (Nederlandse Antillen) Details	Willemstad	AN	599	Americas
Angola Details	Luanda	AO	244	Africa
Argentina Details	Buenos Aires	AR	54	Americas
American Samoa (Amerika Samoa) Details	Pago Pago	AS	685	Oceania
Austria (Österreich) Details	Vienna	AT	43	Europe

List rendered from static_countries table

Besides the list itself, there are some more widgets that can be created by pt_extlist:

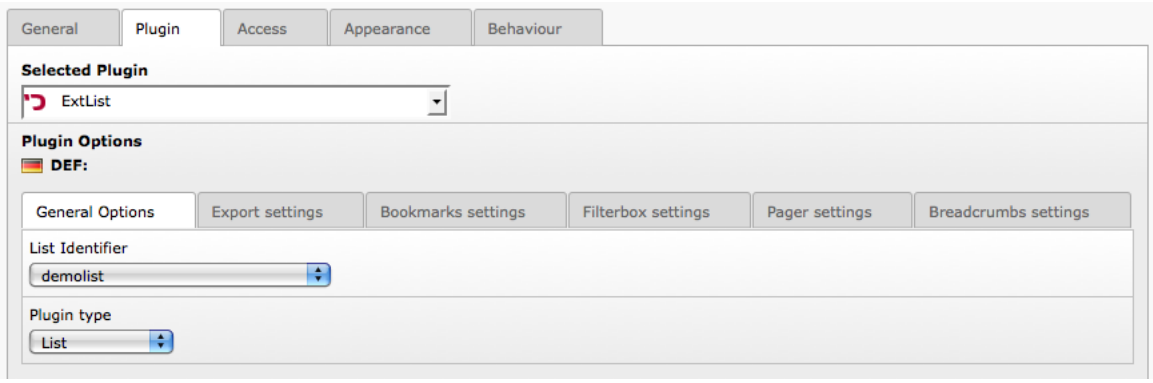


Filters for static_countries table



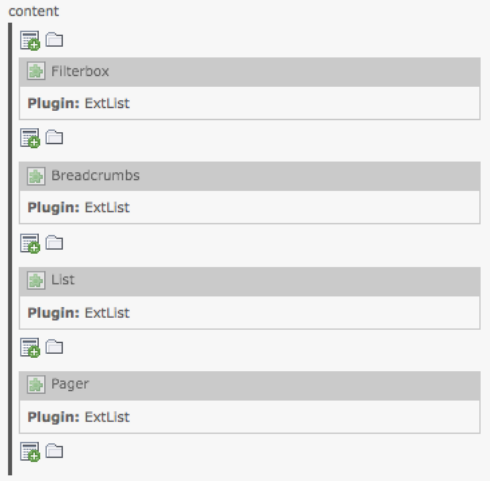
Pager for static_countries table

The plugin's flexform lets you insert a pt_extlist plugin as a content element where you can configure your plugin's appearance:



Flexform for inserting plugin

You can put several content elements on a page for setting up the layout and appearance of your widgets:



Content elements for pt_extlist

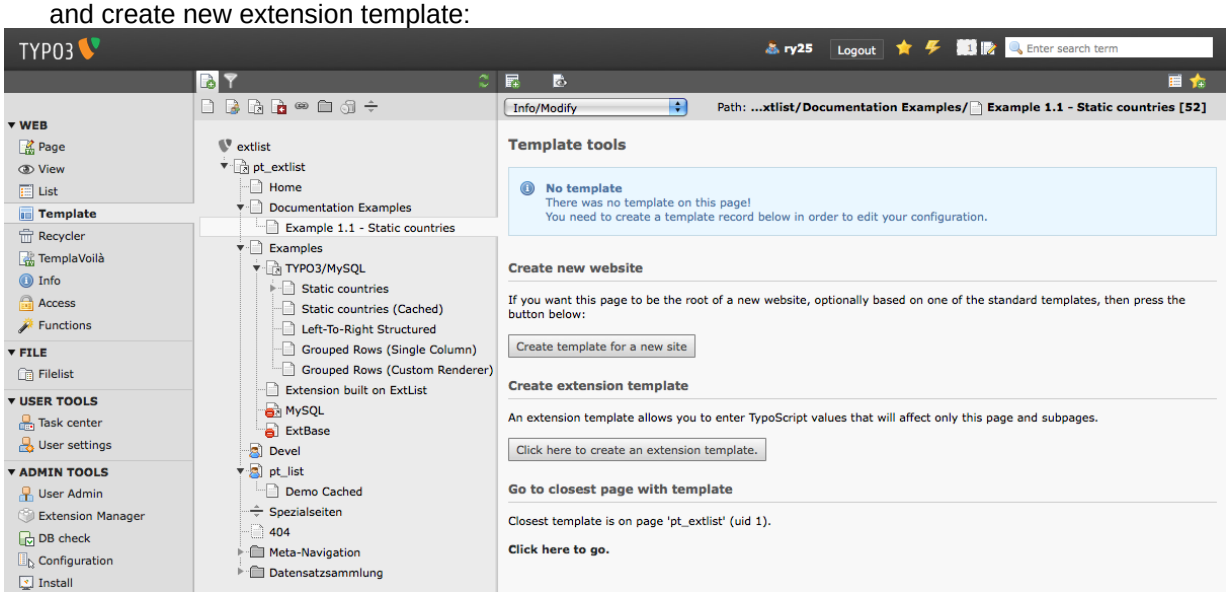
Examples

In this section, some examples will be provided to describe the functionality of pt_extlist. Before you can start, make sure, that pt_extlist is installed and loaded using Extension Manager.

Setting up a demo list based on static_countries table

In this example, you will learn how to create a list by using a TYPO3 table as data source. We will use static_countries table as it is available on all TYPO3 installations. We will set up a page showing filters, list and pager for static countries.

1. Create a new page inside your page tree and open the template module. Open Template module and create new extension template:



Create new extension template ###TODO### insert 1,2,3 for showing what to do in image

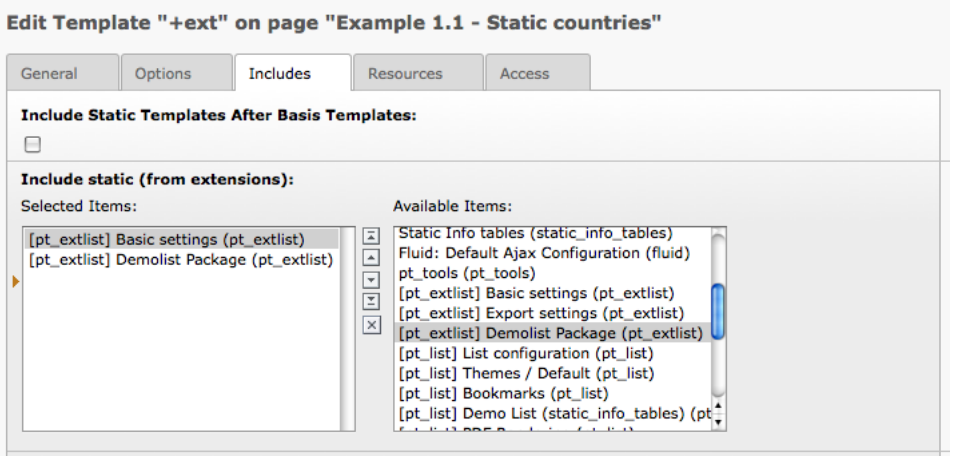
2. Give your extension template a proper name:

Give your extension template a proper name

3. Switch to the "Includes" Tab and select the following templates:

Select Basic settings and demolist package as static templates

4. Save your template and switch to the page module.
5. Select the page you just created and insert a new content element of type "plugin":



Insert plugin as content element

Select ExtList from the page content's "Selected Plugin" list:

Select ExtList as content type

In the flexform for ExtList select "demolist"

Select "demolist" as list identifier

As plugin type select "Filterbox":

Select "Filterbox" as Plugin Type

Switch to the "Filterbox settings" Tab and input "filterbox1" as Filterbox Identifier:

Setting the filterbox identifier

Save your content element and create another one just below. Select "Plugin" as content type and "ExtList" as plugin type just as you did before. Again select "demolist" as list identifier (steps 5 - 7 above), but this time select "list" as plugin type:

Select "List" as Plugin Type

Save and create a third content element. Repeat steps 5 - 7 from above, the select "demolist" as List Identifier and select "Pager" as Plugin Type:

Select "Pager" as Plugin Type

Save content element and take a look at the page in the Frontend. Depending on your CSS Styles, it should look somehow like that:

Frontend view of ExtList widgets

Now let's do a little more advanced stuff and change the number of records shown per page. Therefore switch to the Template module and select the page where you added the content elements above. Write the following line of code into your setup field: plugin.tx_ptextlist.settings.listConfig.demolist.pager.itemsPerPage = 4 Now reload your page in the Frontend and look what's happening - there should be only 4 records per page anymore:

List after changing items per page

So that's it - you just set up your first list! Feel free to test the other sample configurations shipping with pt_extlist to see some more features.

Introduction

A documentation for users, administrators and developers.

The users will learn how to install and setup a demo list. The administrators will learn setting up a list and plugin configuration. The developers will learn what pt_extlist is about and how they can manipulate pt_extlist.

The users and administrators need the basic knowledge about programming.

What does it do?

This extension is intended to generate all sorts of lists. The data sources for the list can be a database or an extbase repository or anything you write a data-backend for (SOAP, CSV, XML, ...).

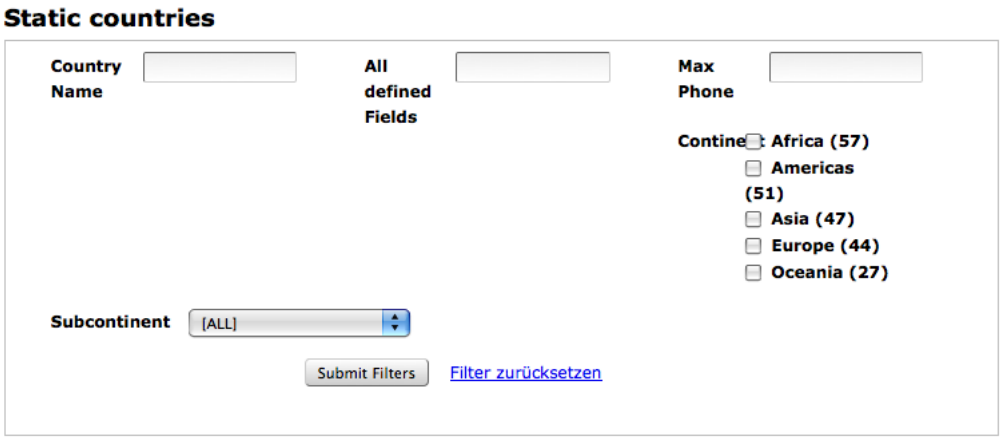
There are different steps you have to do if you want to set up a list. For a detailed example see section "Instruction".

Here are some screenshots to give you an impression of how it looks like.

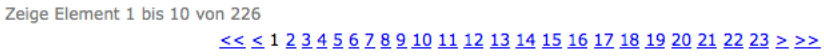
Country name (local country name)	Capital	ISO	Phone	Continent
Andorra Details	Andorra la Vella	AD	376	Europe
Afghanistan (افغانستان) Details	Kabul	AF	93	Asia
Antigua and Barbuda Details	St John's	AG	1268	Americas
Anguilla Details	The Valley	AI	1264	Americas
Albania (Shqipëria) Details	Tirana	AL	355	Europe
Netherlands Antilles (Nederlandse Antillen) Details	Willemstad	AN	599	Americas
Angola Details	Luanda	AO	244	Africa
Argentina Details	Buenos Aires	AR	54	Americas
American Samoa (Amerika Samoa) Details	Pago Pago	AS	685	Oceania
Austria (Österreich) Details	Vienna	AT	43	Europe

List rendered from static_countries table

Besides the list itself, there are some more widgets that can be created by pt_extlist:

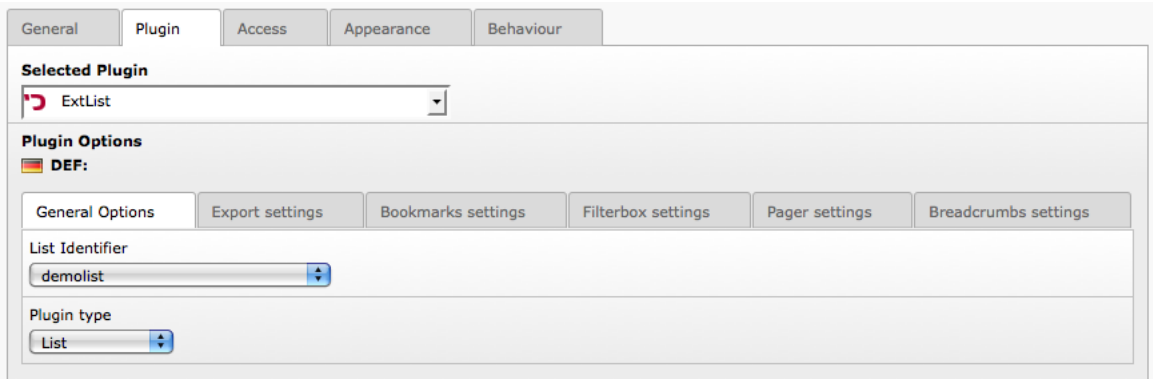


Filters for static_countries table



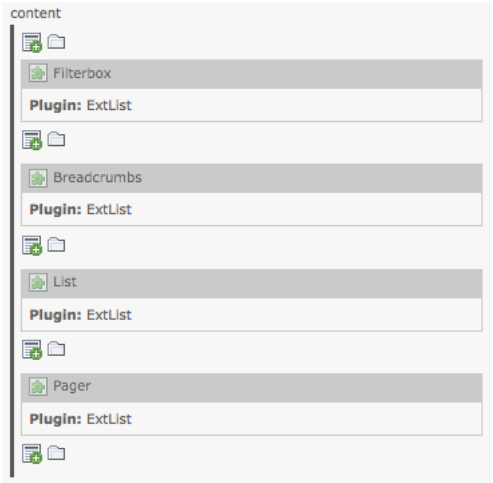
Pager for static_countries table

The plugin's flexform lets you insert a pt_extlist plugin as a content element where you can configure your plugin's appearance:



Flexform for inserting plugin

You can put several content elements on a page for setting up the layout and appearance of your widgets:



Content elements for pt_extlist

Configuraration

Setting up Lists

In this section you will learn how to set up lists using pt_extlist. We will guide you step by step through the TypoScript configuration and show you how to use pt_extlist's widgets as page content.

Widgets overview

Get an overview of what the individual widgets are doing and how they look like in the frontend. All widgets depend on a list identifier set up in TypoScript and selected within the FlexForm of your plugin.

List widget

Renders a list of data set up by configuration. Can use headers for sorting list data by certain columns.

Filter widgets

Renders filterboxes containing multiple filters defined by configuration.

Pager widget

Renders a pager as configured by configuration. Pager limits rows of list to configured amount per page.

Breadcrumbs widget

Breadcrumbs show which filters are activated and which values they have.

Bookmarks widget

Bookmarks enable user to save certain list settings like filters, pager, sortings and reload them again afterwards.

TypoScript configuration

List Identifier and TypoScript namespace Each list has its own identifier.

Sample Configuration:

```
The following listing shows a sample configuration as it ships with pt_extlist.
plugin.tx_ptextlist.settings {

_LOCAL_LANG.default.emptyList = empty list

listConfig.demolist < plugin.tx_ptextlist.prototype.listConfig.default
listConfig.demolist {

backendConfig < plugin.tx_ptextlist.prototype.backend.typo3
backendConfig {

datasource {
# no configuration required here
}

tables (
static_countries,
static_territories st_continent,
static_territories st_subcontinent
)

baseFromClause (
static_countries
LEFT JOIN static_territories AS st_subcontinent ON (static_countries.cn_parent_tr_iso_nr = st_subcontinent.tr_iso_nr)
LEFT JOIN static_territories AS st_continent ON (st_subcontinent.tr_parent_iso_nr = st_continent.tr_iso_nr)
)

baseWhereClause (
st_continent.tr_name_en <> ""
AND st_subcontinent.tr_name_en <> ""
)
}

fields {
name_local {
table = static_countries
field = cn_short_local
isSortable = 1
}

name_en {
table = static_countries
field = cn_short_en
}

uno_member {
table = static_countries
field = cn_uno_member
}

capital {
table = static_countries
field = cn_capital
}

iso2 {
```

```
table = static_countries
field = cn_iso_2
isSortable = 0
}

phone {
table = static_countries
field = cn_phone
}

isoNo {
table = static_countries
field = cn_currency_iso_nr
}

continent {
table = st_continent
field = tr_name_en
```

```
}

subcontinent {
table = st_subcontinent
field = tr_name_en
}

countryuid {
table = static_countries
field = uid
}
}

pager {
pagerConfigs {
second {
enabled = 1
pagerClassName = Tx_PtExtlist_Domain_Model_Pager_DefaultPager
templatePath = EXT:pt_extlist/Resources/Private/Templates/Pager/second.html

showNextLink = 1
showPreviousLink = 1
showFirstLink = 0
showLastLink = 0
}
}
}

columns {

10 {
columnIdentifier = nameColumn
label = LLL:EXT:pt_extlist/Configuration/TypoScript/Demolist/locallang.xml:column_nameColumn

fieldIdentifier = name_local, name_en, countryuid, uno_member
isSortable = 1
sorting = name_local

renderObj = COA
renderObj {
5 = IMAGE
5.if {
value.data = field:uno_member
equals = 1
}
5.file = EXT:pt_list/typoscript/static/demolist/un.gif
5.stdWrap.typolink.parameter = http://www.un.org
5.stdWrap.typolink.ATagParams = class="un-link"

10 = TEXT
10.data = field:name_en
10.append = TEXT
10.append {
data = field:name_local
if {
value.data = field:name_local
```

```
equals.data = field:name_en
negate = 1
}
}
10.append.noTrimWrap = | (\|)|
10.wrap3 = \| 

20 = TEXT
```

```
20.value = Details
20.typolink.parameter = 1
20.typolink.additionalParams.dataWrap = &tx_unseretolleextension_controller_details[countryuid]={field:countryuid}
}
}

11 {
label = Capital
columnIdentifier = capital
fieldIdentifier = capital
cellCSSClass {
renderObj = TEXT
renderObj.dataWrap = {field:capital}
}
}

20 {
label = LLL:EXT:pt_extlist/Configuration/TypoScript/Demolist/locallang.xml:column_isoNoColumn
columnIdentifier = isoNoColumn
fieldIdentifier = iso2
isSortable = 1
}

30 {
label = Phone
columnIdentifier = phoneColumn
fieldIdentifier = phone
}

40 {
label = Continent
columnIdentifier = continent
fieldIdentifier = continent
}

50 {
label = Subcontinent
columnIdentifier = subcontinent
fieldIdentifier = subcontinent
accessGroups = 3
}
}

aggregateData {
sumPhone {
fieldIdentifier = phone
method = sum
}
avgPhone {
fieldIdentifier = phone
method = avg
}
maxPhone {
fieldIdentifier = phone
method = max
}
```

```
minPhone {
fieldIdentifier = phone
method = min
}
}

aggregateRows {
10 {
phoneColumn {
aggregateDataIdentifier = sumPhone, avgPhone, maxPhone, minPhone
renderObj = TEXT
renderObj.dataWrap (
Min.: <b>{field:minPhone}</b><br />
&empty;: <b>{field:avgPhone}</b><br />
Max.: <b>{field:maxPhone}</b><br />
&sum;: <b>{field:sumPhone}</b><br />
)
```

```
}
}
}

filters {
filterbox1 {
filterConfigs {
10 < plugin.tx_ptextlist.prototype.filter.string
10 {
filterIdentifier = filter1
label = LLL:EXT:pt_extlist/Configuration/TypoScript/Demolist/locallang.xml:filter_nameField
fieldIdentifier = name_local
}

11 < plugin.tx_ptextlist.prototype.filter.string
11 {
filterIdentifier = allFields
label = All defined Fields
fieldIdentifier = *
}

15 < plugin.tx_ptextlist.prototype.filter.max
15 {
filterIdentifier = filter15
label = Max Phone
fieldIdentifier = phone
accessGroups = 3
}

20 < plugin.tx_ptextlist.prototype.filter.checkbox
20 {
filterIdentifier = filter2
label = Continent
fieldIdentifier = continent
filterField = continent
displayFields = continent
showRowCount = 1
submitOnChange = 0
invert = 0
invertable = 0

excludeFilters = filterbox1.filter3
}

30 < plugin.tx_ptextlist.prototype.filter.select
30 {
filterIdentifier = filter3
label = Subcontinent
fieldIdentifier = subcontinent
filterField = subcontinent
displayFields = continent, subcontinent
multiple = 0
showRowCount = 1
submitOnChange = 0
inactiveOption = \[ALL]
invert = 0
```

```
invertable = 0
}

}
}

filterbox2 {
```

```
showSubmit = 0
showReset = 0
filterConfigs {
10 < plugin.tx_ptextlist.prototype.filter.firstLetter
10 {
filterIdentifier = filter4
label = Capital
fieldIdentifier = capital
}
}
}
}
}
}

plugin.tx_ptextlist.settings.listConfig.demoListProxyFilter {
backendConfig < plugin.tx_ptextlist.prototype.backend.typo3
backendConfig {
tables (
static_territories
)

continent {
table = static_territories
field = tr_name_en
}
}

fields {
continent {
table = static_territories
field = tr_name_en
}
}

filters {
filterbox1 {
filterConfigs {
10 < plugin.tx_ptextlist.prototype.filter.select
10 {
filterIdentifier = continent
label = Subcontinent
fieldIdentifier = continent
filterField = continent
displayFields = continent
showRowCount = 0
multiple = 0
inactiveOption = \[ALL]

renderObj = TEXT
renderObj {
dataWrap = {field:allDisplayFields}
}
}
}
}
```

```
}
}

################################
# Localization Override
################################
plugin.tx_ptextlist._LOCAL_LANG{
default {
emptyList = List is empty.
}
de {
emptyList = Liste ist leer.
}
}
```

backendConfig section

Setting up a MySQL backend

Setting up a TYPO3 backend

Setting up a Extbase backend

fields section

Setting up fields for database backends

Setting up fields for Extbase domain objects

columns section

filters section

pager section

aggregates

aggregateData section

aggregateRow section

Localization override

Setting up widgets as content elements

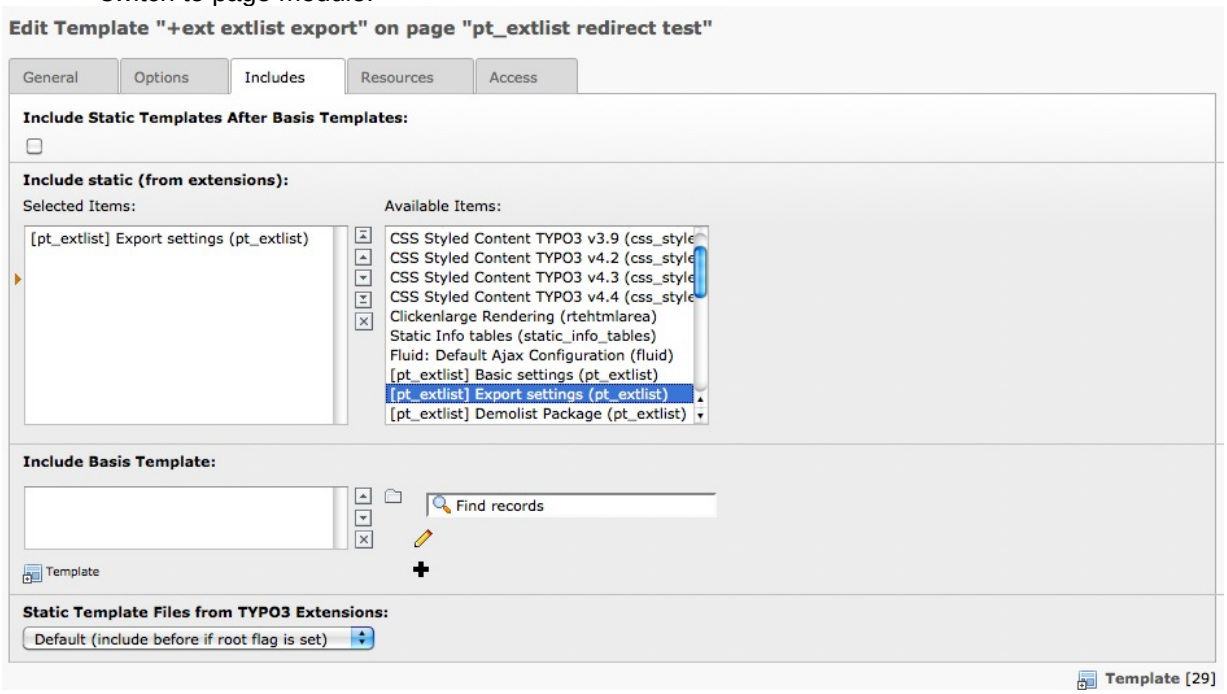
Integrators Guide

Setting up export

pt_extlist offers several exporters for lists. You can select from a list of pre-defined export formats or implement your own exporters. Here is a step-by-step explanation on how to set up export:

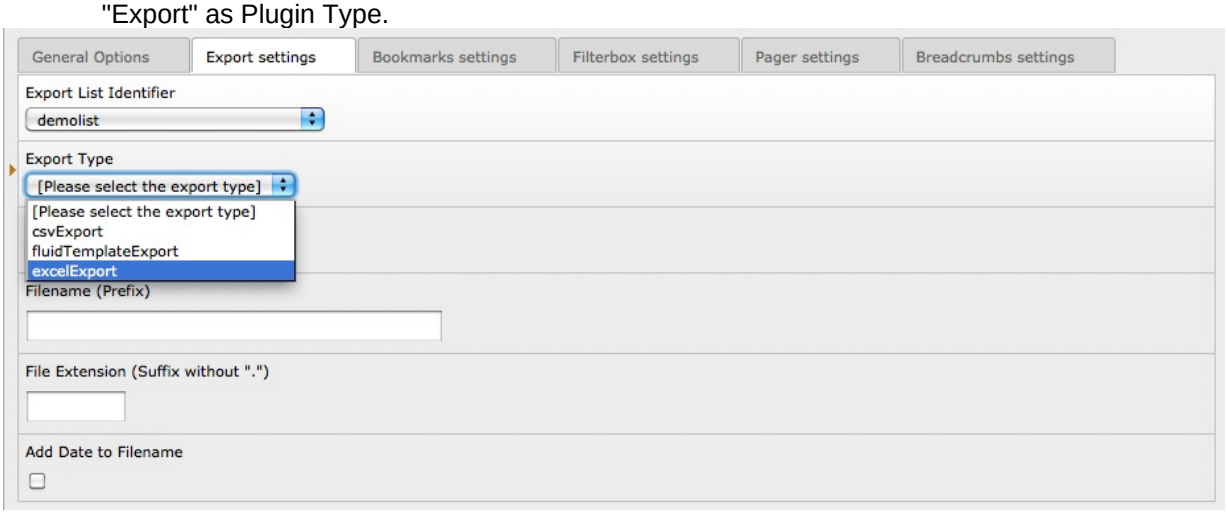
1. Go to the page on which you have your list set up.

2. You have to include a static template for export settings on the page you want to export list data. Go to the Template module and modify the template of the page. Switch to the "Includes" tab of your template record and select "[pt_extlist] Export settings (pt_extlist)". Save your template and switch to page module.



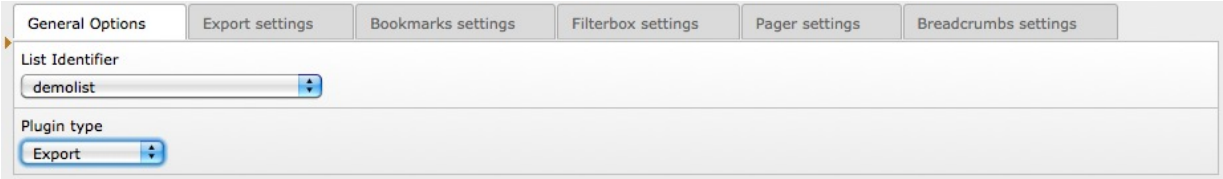
Inclusion of static template for export

1. Insert a new content element of type "General Plugin". Select "ExtList" as selected Plugin.
2. Switch to the "General Options" tab and select list identifier of the list you want to export. Select "Export" as Plugin Type.



Set plugin type to "Export"

1. Switch to the "Export Settings" tab and select list identifier of the list you want to export. Select Export Type and Download Type. Hint: If you cannot select an Export Type, you most likely forgot to include static template for export on the page you are currently working. See step 2!



Configuration for exporting a list as Excel sheet

1. Save your content element and switch to frontend view.
2. You will now see a download in your Frontend that enables you to download configured export document with your list data.

Why are there 2 list identifiers?

Almost everytime you want to export some data from your list, you also want to change the way the list looks like in your export. Therefore you can select a different list identifier for export than for your "normal" list. This way you can configure the changes for the exported list on the same page you have your "normal" list. In previous versions of pt_list, you had to create a special page with special TS-settings for your exported list. By chosing a second identifier for your exported list, this is no longer required!

Requirements for Excel export

There are special requirements for setting up Excel Export. You have to install the module PHPExcel which is available via a special PEAR channel. In order to install PHPExcel, refer to their website: http://phpexcel.codeplex.com/releases/view/45412 The current installation process looks like this:

- Set up PEAR on your system.
- Use the following command to make PEAR channel known to your

```
pear channel-discover pear.pearplex.net
```

- Use the following command to install PHPExcel on your system:

```
pear install pearplex/PHPExcel
```

Configure the excel export

Cell styling

Column header and column body can be styled differently for every column with typoscript. For doing this, we added another protype for column settings that extends the default column settings. In contrast to the default column prototype, which is merged automatically to every column, we have to assign the excel export column manually:

```
10 < plugin.tx_ptextlist.prototype.column.excel
10 {
    fieldIdentifier = field1
    ...
}
```

It is best to have a look into the prototype settings for the column to see which options are available.

These are the configuration values that are offered by PHPExcel.

excelExport.<section>.vertical:

- bottom
- top
- center

- justify

excelExport.<section>.style:

- none
- dashDot
- dashDotDot
- dashed
- dotted
- double
- hair
- medium
- thick
- thin

excelExport.<section>.fill:

- none
- solid
- linear
- path
- darkDown
- darkGray
- darkGrid
- darkHorizontal
- darkTrellis
- darkUp
- darkVertical
- gray0625
- gray125
- lightDown
- lightGray
- lightGrid
- lightHorizontal
- lightTrellis
- lightUp
- lightVertical
- mediumGray

Define all columns to a common style

To define a common style that fits to the page CD just add the following typoscript to a basic typoscript file:

```
# Page CI Excel Settings
plugin.tx_ptextlist.settings.prototype.column.default < plugin.tx_ptextlist.settings.prototype.column.excel
plugin.tx_ptextlist.settings.prototype.column.default {
    # Define all custom configuration for all fields here
    excelExport.header.fill.color = ffcc00
}
```

Developpers Guide

Pt_extlist can be extended in multiple ways. Many of its classes are configured via TypoScript so you can easily exchange them with your own classes to fit your needs. Common types of extensions are changing Data-Backends or writing your own filter classes. We will start with the latter one.

Writing your own filter classes

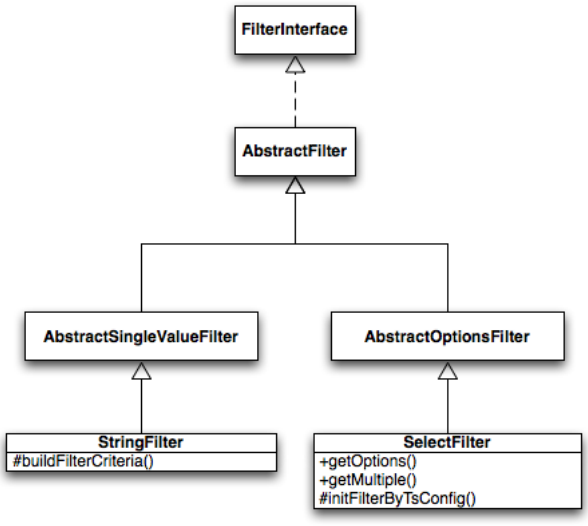
Recapitulating what has been told about filters in the Architecture chapter, we reintroduce the following class diagram to understand what filters are actually doing:

Filter Interface

By taking a look at the Interface for filters, you see that there are mainly three main purposes:

1. Configuration and State-related stuff
2. Returning a filter query that determines what the filter is actually filtering on the data
3. Creating a filter breadcrumb information

Keeping in mind that there are some helpers - namely abstract classes - that do a lot of work for us we do not have to implement much logic when creating a new filter class:



So as you can see - all that's left for you to implement in your concrete filter class is a method that creates the actual filter criteria.

String Filter Example

One of the most simple filters shipping with pt_extlist is the String filter. It can filter a string value based on a user input which is also a string. You can find the String-Filter class in the Classes/Domain/Model/Filter/StringFilter.php file. Here is the PHP source code: class Tx_PtExtlist_Domain_Model_Filter_StringFilter extends:

```
Tx_PtExtlist_Domain_Model_Filter_AbstractSingleValueFilter {

/\**
* Creates filter query from filter value and settings
*
* @return Tx_PtExtlist_Domain_QueryObject_Criteria Criteria for current filter value (null, if empty)
\*/
protected function buildFilterCriteria(Tx_PtExtlist_Domain_Configuration_Data_Fields_FieldConfig $fieldIdentifier) {

    if ($this->filterValue == '') {
        return NULL;
    }

    $fieldName = Tx_PtExtlist_Utility_DbUtils::getSelectPartByFieldConfig($fieldIdentifier);
    $filterValue = '%'.$this->filterValue.'%';

    $criteria = Tx_PtExtlist_Domain_QueryObject_Criteria::like($fieldName, $filterValue);

    return $criteria;
    }
}
```

The most important function is *buildFilterCriteria()* where the filter creates a constraint on how the data filtered by this filter should look like. We use our generic query criteria

Tx_PtExtlist_Domain_QueryObject_SimpleCriteria

with an operator like here to implement a string filter that uses a LIKE-comparison in its built criteria.

Tx_PtExtlist_Domain_QueryObject_Criteria::like($fieldName, $filterValue)

is nothing more but a factory method that returns a criteria object. As we mentioned above, a lot of functionality is given to us by our abstract classes, so to get some more information about what the String-Filter does and how it is configured, ake a look at its TypoScript prototype located in Configuration/TypoScript/BaseConfig/Prototype/Filter.txt:

```
string {
    filterClassName = Tx_PtExtlist_Domain_Model_Filter_StringFilter
    partialPath = Filter/String/StringFilter
    defaultValue =
    accessGroups =

    breadCrumbString = TEXT
    breadCrumbString {
    # Fields that can be used are "label" and "value"
    dataWrap = {field:label} equals {field:value}
    }
}
```

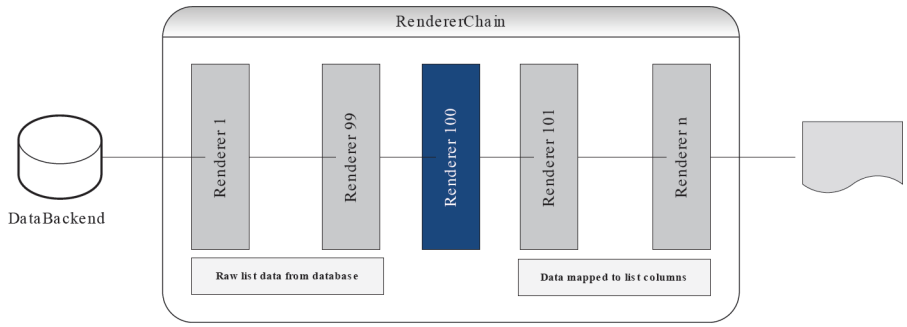
You find a lot more configuration possibilities here than you would assume after looking at the filter class above. First of all, there is a filterClassName, that determines which filter class to instantiate in order to create a string filter object. The partial path leeds us to the HTML template that is used for the filter's user interface. defaultValue lets us set a predefined value when the filter is shown for the first time and accessGroups restricts the filter to certain fe_groups that are allowed to see the filter. breadCrumbString

enables us to create a TS template for rendering the breadcrumb text of the filter. The last thing we have to know, when we want to implement our own filter class is how to actually configure them within our list configuration. Therefore you should take a look at one of the demolists' filterbox configurations. There we find something like this:

```
filters {
    filterbox1 {
        filterConfigs {
            10 < plugin.tx_ptextlist.prototype.filter.string
            10 {
            filterIdentifier = filter1
            label = LLL:EXT:pt_extlist/Configuration/TypoScript/Demolist/locallang.xml:filter_nameField
            fieldIdentifier = name_local
            }
        }
    }
}
```

All the filters of a list configuration are configured in the filters section of your configuration. Within this section you have to set up a arbitrary key for the name of your filterbox. In the example above, this is filterbox1. For each filterbox, you have to set up a list of filters within filterConfigs and in there we finally have our String-Filter. The basic settings are copied from the prototype above, then we have to change the settings that are unique for our usasge of the filter like filterIdentifier, label and the fieldIdentifier we want to let our filter operate on.

Extending the RenderChain



RenderChain

Using extlist in the TYPO3 backend

Extbase enables you to write backend modules the same easy way as you do in the frontend. The main difference however is that in the frontend you can have multiple plugins with controller / action pair fired at each rendering, whereas in the backend you can only call one controller / action at a time. As pt_extlist in the frontend by default uses one plugin each for filter, list and pager, we have to use the extension in the backend in a different way to cope with the one controller/action restriction.

Derive from the Tx_PtExtlist_Controller_AbstractBackendListController

Use pt_extlist to render lists within your own extension

It is also posible to use pt_extlist to render the list inside your own extension. This is done by the extlistContext, an object that encapsulates all parts of extlist models. This is a step by step example on how to integrate an extlist into your extension.

1. Define the lists typoscript inside your extensions scope

The easiest way to access the typoscript within your controller is to define it inside your extensions typoscript scope:

```
plugin.<YOUREXTENSION>.settings.extlist.<YOURLISTIDENTIFIER> < plugin.tx_ptextlist.prototype.list
plugin.<YOUREXTENSION>.settings.extlist.<YOURLISTIDENTIFIER> {

... your extlist config goes here ...

}
```

2. Instantiate extlist in your controller-action

The following example shows the instantiation and usage of pt_extlist in your own controller and action. Your Controller should extend the *Tx_PtExtbase_Controller_AbstractActionController*, if you want to use the cross-extension partial usage. The method *getListContext()* cals the factory with the factory comamnd *getContextByCustomConfiguration* which accepts your extlist configuration as the first parameter and the listIdentifier, that should be used in the second parameter. If you want to display the extlist in the *listAction*, all you have to do is to assign all variables to the view by using:

```
$this->view->assignMultiple($this->getListContext()->getAllListTemplateParts());
```

That is all you have to do to display a list within your extension. If you also want to interact with your list, for example page, sort or filter it, you have to add some more actions to handle this to your controller.

The complete example controller:

```
class Tx_<YOUREXTENSION>_Controller_AbstractController extends Tx_PtExtbase_Controller_AbstractActionController {

	/**
	* @retutn Tx_PtExtlist_ExtlistContext_ExtlistContext
	*/
	protected function getListContext() {
		return Tx_PtExtlist_ExtlistContext_ExtlistContextFactory::getContextByCustomConfiguration($this->settings['extlist']['<YOURLISTIDENTIFIER>'], '<YOURLISTIDENTIFIER>');
	}

	/**
	* List action to render the extlist
	*/
	public function listAction() {
		$this->view->assignMultiple($this->getListContext()->getAllListTemplateParts());
	}

	/**
	 * Sorting action used to change sorting of a list
	 */
	public function sortAction() {
		$this->getListContext()->getDataBackend->resetListDataCache();
		$this->getListContext()->getDataBackend->getSorter()->reset();

		$this->forward('list');
	}


	/**
	 * Resets all filters of filterbox
	 *
	 * @param string $filterboxIdentifier Identifier of filter which should be reset
	 * @return string Rendered reset action
	 */
	public function resetAction($filterboxIdentifier) {
		if ($this->getListContext()->getFilterBoxCollection()->hasItem($filterboxIdentifier)) {
			$this->getListContext()->getFilterBoxCollection()getFilterboxByFilterboxIdentifier($filterboxIdentifier)->reset();
		}

		$this->getListContxt()getPagerCollection()->reset();

		$this->redirect('list');
	}
}
```

3. Configure ext_localconf / flexform

Dno't forget to alter the Tx_Extbase_Utility_Extension::configurePlugin() in your ext_localconf to allow the extlist specific actions to be executed. The same holds if you configured switchableControllerActions in your flexform.

4. Add the extlist partials to your Template

Last thing to do: Add the extlist fluid template part to your template:

```
... your template code ...

<table class="table table-bordered table-striped tx-ptextlist-list tx-ptextlist-list-standard" id="tx-ptextlist-list-{config.listConfiguration.listIdentifier}">
	<thead>
		<f:render partial="{config.listConfiguration.headerPartial}" arguments="{listHeader:listHeader, listCaptions:listCaptions}" />
	</thead>
	<tbody>
		<f:render partial="{config.listConfiguration.bodyPartial}" arguments="{listData:listData}" />
		<f:render partial="{config.listConfiguration.aggregateRowsPartial}" arguments="{aggregateRows:aggregateRows}" />
	</tbody>
</table>
```

Cookbook

This chapter is a collection of recipes for common tasks in pt_extlist.

Insert the output of an extlist / extbase plugin via typoscript.

It is quite simple to insert an extlist / extbase plugin in any mode as a widget directly via typoscript. The following typoscript snippet defines a widget to render a pager anywhere you like.

controller / action

Defines the called controller / action pair. That would be Pager / show for a pager, Filterbox / show for a filterbox and List / list for a list.

switchableControllerActions

Defines all callable controller / action combinations within this widget. Must contain at least the default controller / action defined above.

listIdentifier

listIdentifier is the only needed configuration. Set it to the desired list configuration.

Typoscript:

```
pagerWidget = USER
pagerWidget {
    userFunc = tx_extbase_core_bootstrap->run
    pluginName = Pi1
    extensionName = PtExtlist
    controller = Pager
    action = show
    switchableControllerActions {
            Pager {
                    1 = show
            }
    }

    persistence =< plugin.tx_ptextlist.persistence
    view =< plugin.tx_ptextlist.view
    settings =< plugin.tx_ptextlist.settings

    settings {
            listIdentifier = YOURLISTIDENTIFIER
    }
}
```

Exchange the default template by controller / action. In default extbase it is only possible to exhenage the complete template folder. Pt_extlist enables you to exchange a single template per list. The example below shows you how to exchange a template. In general it is

`controller.THECONTROLLERNAME.THEACTIONNAME.template = PATHTOTHETEMPLATE`

This holds for any controller / action pair. Check the URL or the template folder for the controller and action names.

```
plugin.tx_ptextlist.settings {
    listConfig.demolist {
```

```
        controller.List.list.template = EXT:my_extension/Resources/Private/Templates/NewTemplate.html
    }
}
```

Export the list data as a PDF document

The pdf exporter uses the domPdf library to render the list in HTML-format which is generated by a special fluid template to PDF. It is recommended to use the extension pt_dompdf, which on the one hand provides the dompdf sources and on the other hand fluid-viewHelpers to wrap special methods provided by the domPdf library. So all you have to do, is to install the pt_dompdf extension and select the export type: PDF export and you get the PDF table in a nice default layout.

Change size and orientation

You can change the page size and the orientation of the PDF in the settings (have a look into the TSRef for the posible values):

```
plugin.tx_ptextlist.settings.export.exportConfigs.pdfExport {
    paperSize = a4
    paperOrientation =  portrait
}
```

Change the appearance of the document

To change the appearance of the document, you can first edit the CSS stylesheet of the document and second alter the fluid template as well. It is recommended to make a copy of the files and change the paths in the settings:

```
plugin.tx_ptextlist.settings.export.exportConfigs.pdfExport {
    templatePath = typo3conf/ext/pt_extlist/Resources/Private/Templates/Export/PDF/Default.html
    cssFilePath =  EXT:pt_extlist/Resources/Public/CSS/Export/Pdf.css

    headerPartial = Export/PDF/ListHeader
    bodyPartial = Export/PDF/ListBody
    aggregateRowsPartial = Export/PDF/AggregateRows
}
```

The cssFilePath parameter can either be set as an absolute path on the server or by an URL.

Add a page number to the document

In domPdf it is a little bit ugly to add page numbers as dynamic changing values to the document. It is done by an inline PHP script in the template. Pt_dompdf provides a viewHelper which generates this inline PHP for you. Add this viewHelper to the BODY part of your HTML template. The markers {PAGE_NUM} and {PAGE_COUNT} can be used to add the dynamic values to the page.

```
{namespace dompdf=Tx_PtDompdf_ViewHelpers}

<dompdf:staticText position="bottom-right">
    Page {PAGE_NUM} of {PAGE_COUNT}
</dompdf:staticText>
```

Filter by an empty value

You cannot filter by an empty value, because empty values are not submitted at all via the POST array. So what you have to do is cast your empty value in the result set of you MySQL query.

To do this, you define the field where you assume an empty value like this:

```
fields {
    fieldWithEmtyValue {
        special = if(table.field = "", "emptyField", table.field)
    }
}
```

This way the field is not longer empty in the result set and the filter works as expected.

Changelog

Before updating your version of pt_extlist, you should read this to get an impression on what has changed. Make sure to follow advices on which steps are neccessary to make pt_extlist run again after updating.

Changes

0.4.0

ADD: Template for filterbox can now be overwritten in Flexform

ADD: Export is refactored. Export widget now generates direct link for export, page for export list settings is no longer required.

ADD: Excel export is implemented. Requires PHPExcel PEAR package!

FIX: ConfigurationBuilder is no longer singleton, if Flexform matters.

RFT: Format of documentation changed from article to book. Now we have TOC!

FIX: Removed non-required spaces in pager template to fix CSS rendering.

RFT: Introduced sorter to handle sorting in data backend.

CHG: It is now possible to reset filterboxes individually. If you use your own Filterbox templates, you have to update them! Make sure to submit filterbox identifier on sending reset request.

ADD: ListHeader viewhelper now accepts single columns / fields for sorting. If you use your own Header templates, you have to update them!

ADD: Filterboxes can now exclude each other. If Filterbox A is configured to exclude Filter b from Filterbox B, than B.b is not respected when filtering data, if Filter B is active.

ADD: List can now be resetted if no GPvars are send. This enables resetting a page whenever a user visits it for the first time.

ADD: Improved Extbase data backend. Now respects sorting of columns and translates NOT criteria.

ADD: Query for creating list data is now written to devlog.

ADD: Added new dateSelect filter. See documentation for configuration settings.

ADD: Added documentation on how to use pt_extlist as TypoScript widget.

ADD: Added fulltext-filter (only works with MySQL data backend).

ADD: Added redirect on submit for filterboxes.