punkt.de

pt_extlist documentation

punkt.de : pt_extlist : documentation

Version: x

punkt.de GmbH

Kaiserallee 13a

AG Mannheim 108285

Tel.: 0721 9109-0

Fax: 0721 9109-100

76133 Karlsruhe

E-Mail: info@punkt.de

Homepage: http://punkt.de

Chapter 1. Introduction

1. About the document

author	Michael Knoll, Daniel Lienert
version	1.0
status	draft
confidentiality	internal
filename	ptextlist_20110210_V5.pdf
start	18.10.2010
last editing	11.02.2011

Table 1.1. Information about the document

2. Reading this

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. (write more!)

What is the author's intention?

Chapter 2. Documentation

1. 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" ###TODO insert link here###.

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

Figure 2.1. List rendered from static_countries table

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

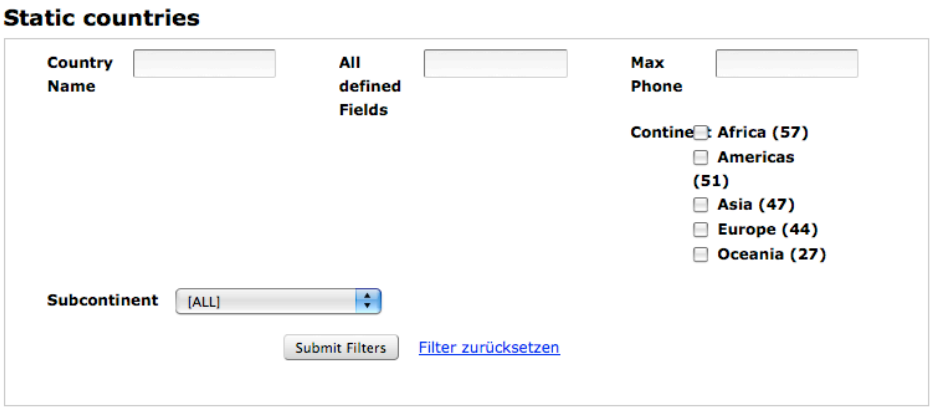

Figure 2.2. Filters for static_countries table

Zeige Element 1 bis 10 von 226

<< < 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 > >>

Figure 2.3. 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:

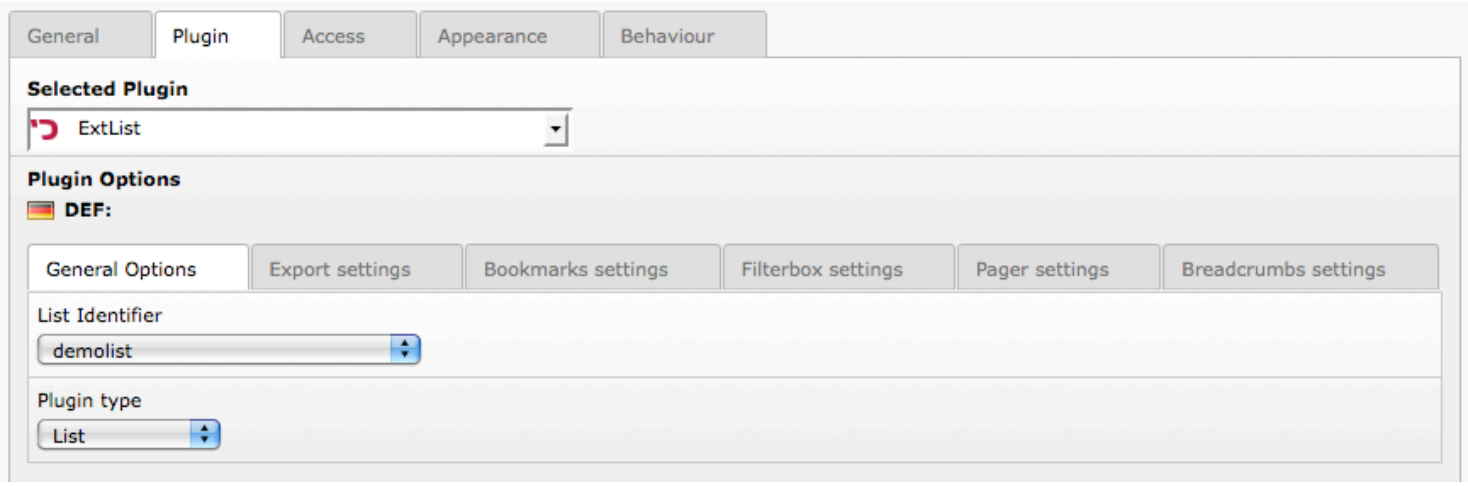

Figure 2.4. Flexform for inserting plugin

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

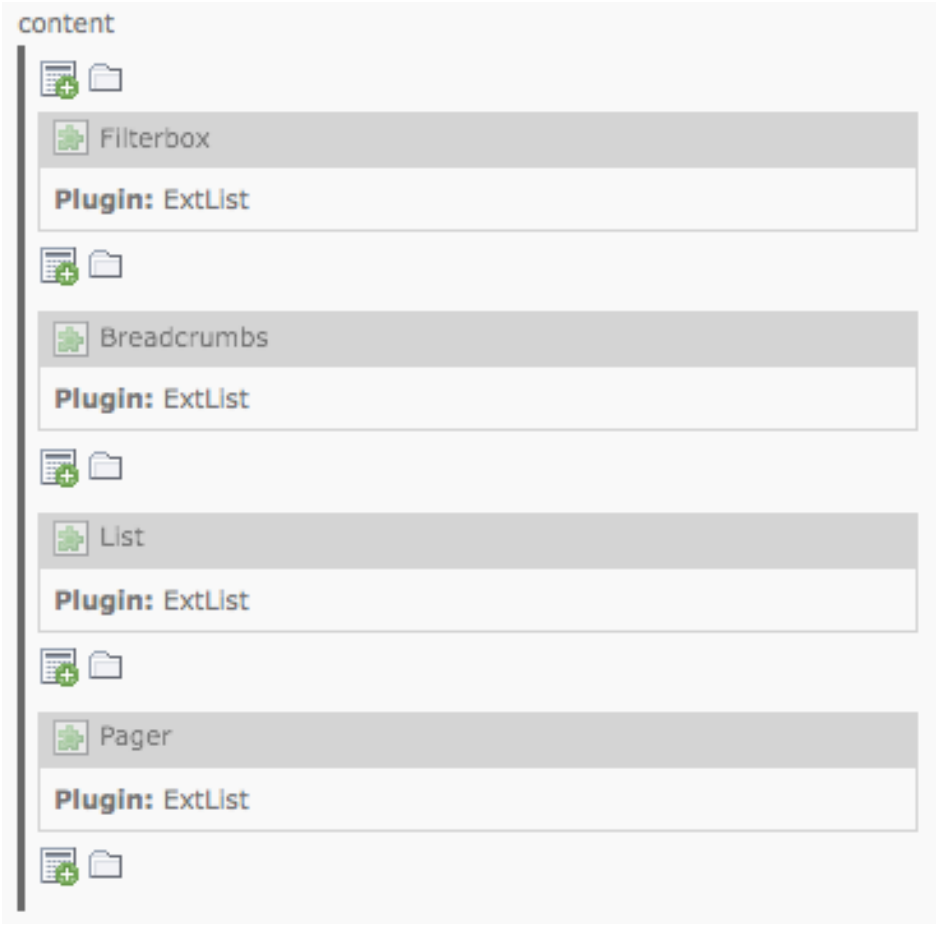

Figure 2.5. Content elements for pt_extlist

2. 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.

2.1. 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:

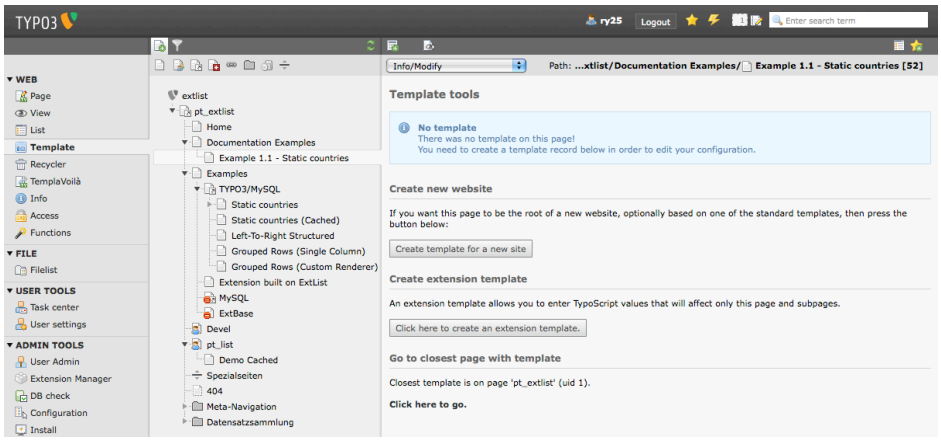

Figure 2.6. Create new extension template ###TODO### insert 1,2,3 for showing what to do in image

2. Give your extension template a proper name:

Figure 2.7. Give your extension template a proper name

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

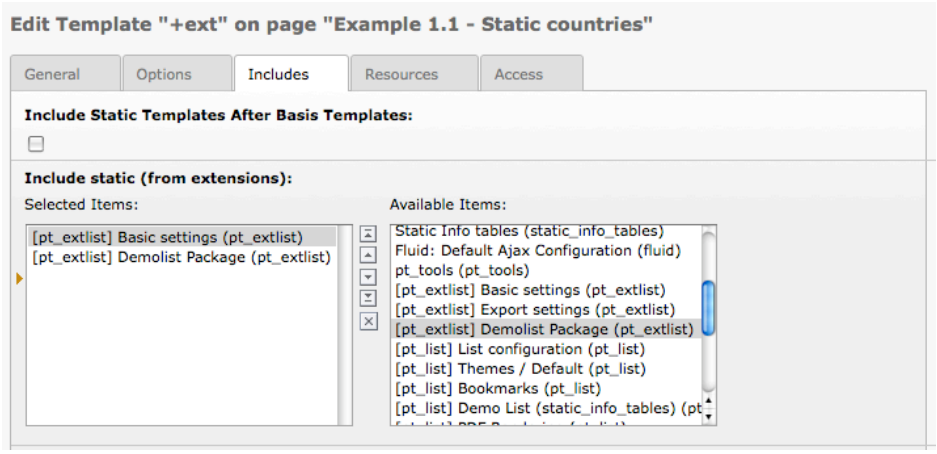

Figure 2.8. Select Basic settings and demolist package as static templates

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":

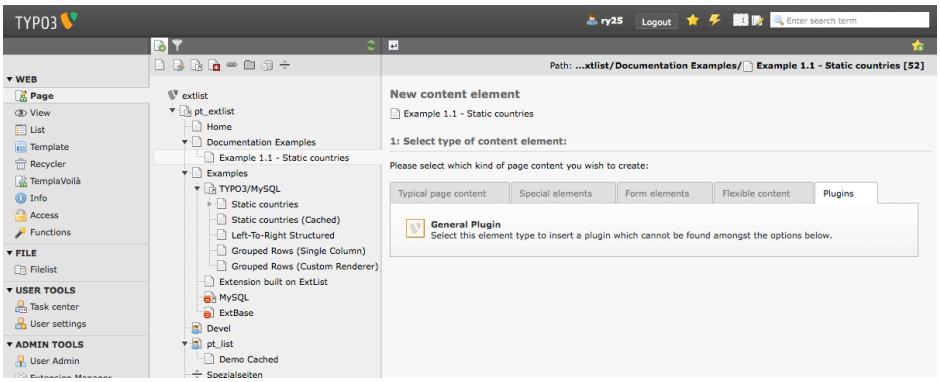

Figure 2.9. Insert plugin as content element

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

Figure 2.10. Select ExtList as content type

7. In the flexform for ExtList select "demolist"

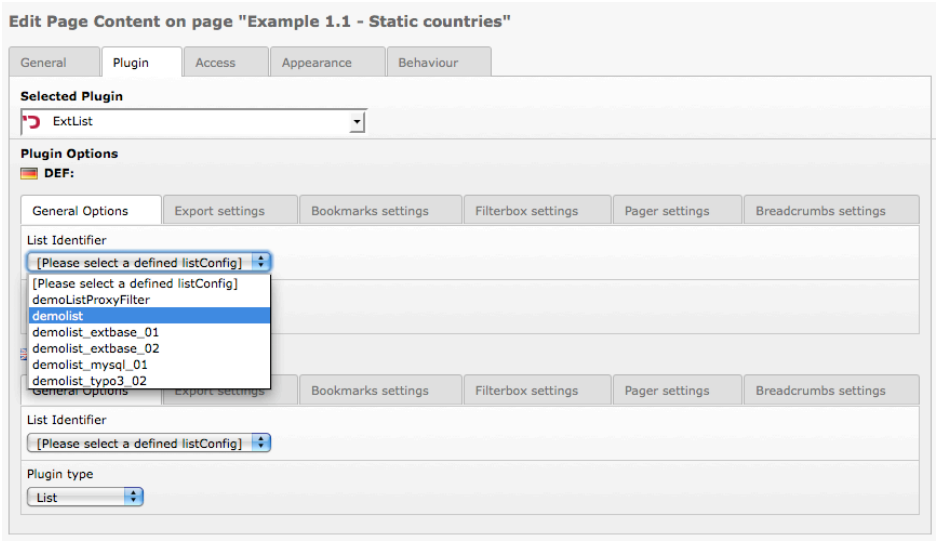

Figure 2.11. Select "demolist" as list identifier

8. As plugin type select "Filterbox":

Figure 2.12. Select "Filterbox" as Plugin Type

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

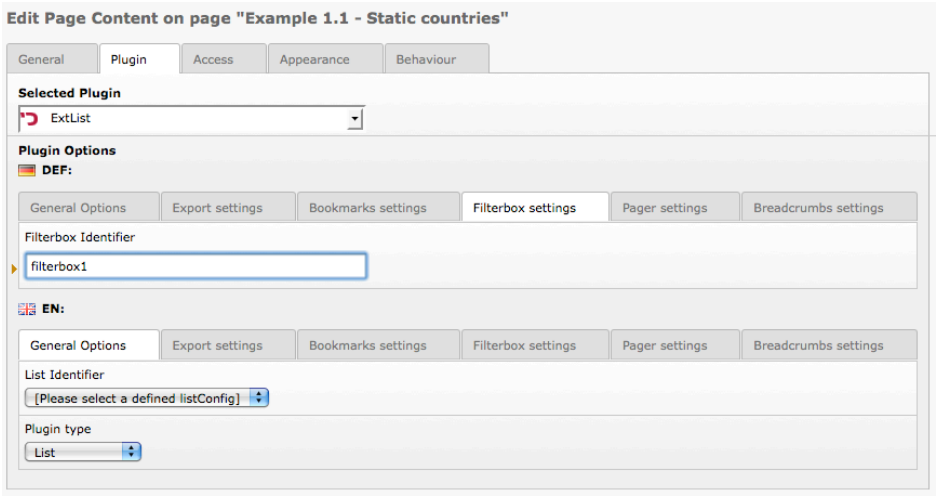

Figure 2.13. Setting the filterbox identifier

10. 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:

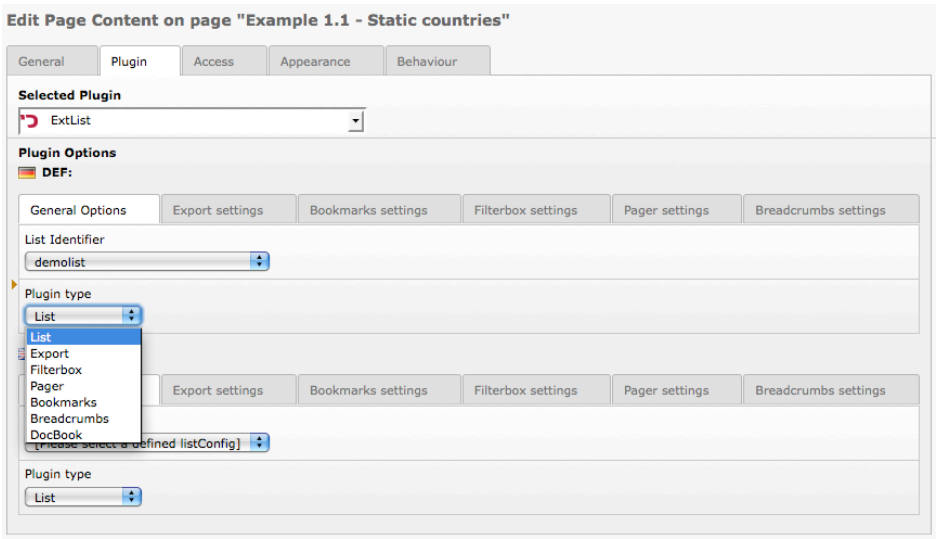

Figure 2.14. Select "List" as Plugin Type

11. 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:

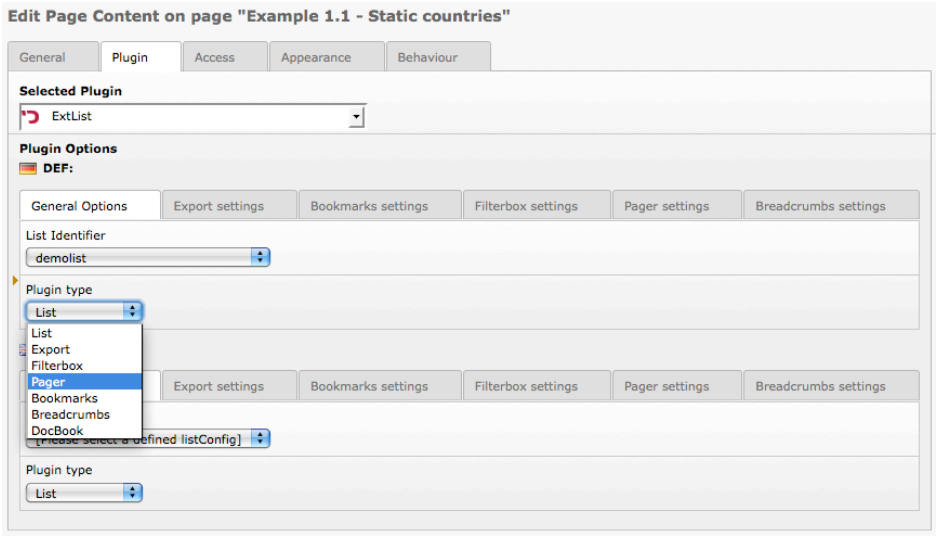

Figure 2.15. Select "Pager" as Plugin Type

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

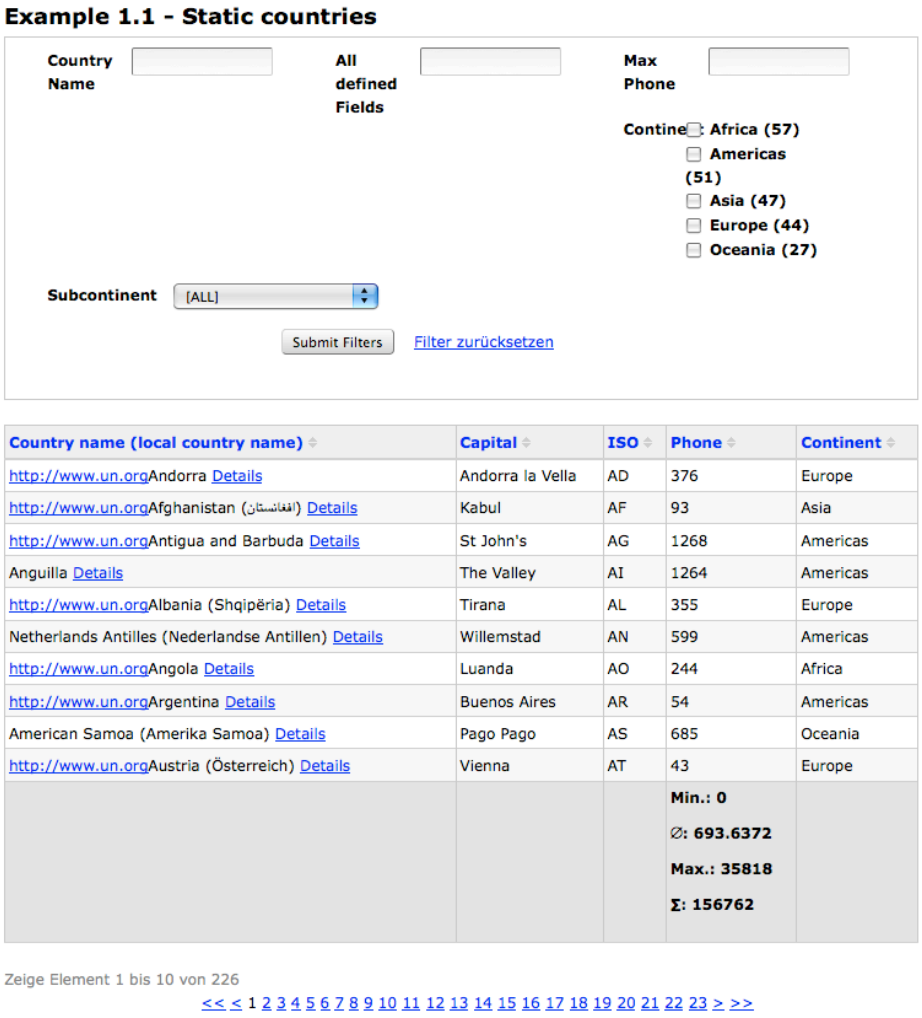

Figure 2.16. Frontend view of ExtList widgets

13 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:

Country name (local country name)	Capital	ISO	Phone	Continent
http://www.un.orgAndorra Details	Andorra la Vella	AD	376	Europe
http://www.un.orgAfghanistan (افغانستان) Details	Kabul	AF	93	Asia
http://www.un.orgAntigua and Barbuda Details	St John's	AG	1268	Americas
Anguilla Details	The Valley	AI	1264	Americas
			Min.: 0 **∅: 693.6372** **Max.: 35818** **Σ: 156762**	

Figure 2.17. 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.

3. 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.

3.1. 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.

3.1.1. List widget

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

3.1.2. Filter widgets

Renders filterboxes containing multiple filters defined by configuration.

3.1.3. Pager widget

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

3.1.4. Breadcrumbs widget

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

3.1.5. Bookmarks widget

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

3.2. TypoScript configuration

3.2.1. List Identifier and TypoScript namespace

Each list has its own identifier.

3.2.2. Sample Configuration

The following listing shows a sample configuration as it ships with pt_extlist.

```
plugin.tx_ptextlist.settings {

_LOCAL_LANG.default.emptyList = blabla

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.
}
}
```

3.2.3. backendConfig section

3.2.3.1. Setting up a MySQL backend

3.2.3.2. Setting up a TYPO3 backend

3.2.3.3. Setting up a Extbase backend

3.2.4. fields section

3.2.4.1. Setting up fields for database backends

3.2.4.2. Setting up fields for Extbase domain objects

3.2.5. columns section

3.2.6. filters section

3.2.7. pager section

3.2.8. aggregateData section

3.2.9. aggregateRow section

3.2.10. Localization override

3.3. Setting up widgets as content elements

4. Architecture

This section gives you an overview of the architecture of pt_exlist. It shows its design principles and gives you a hint on where to make your hands dirty if you want to extend pt_extlist or use it in third party extensions.

4.1. Who should read this?

If you are just interested in how pt_extlist works and want to see how things are put together, the following chapter could be a nice thing to read. If you are a developer and want to extend pt_extlist or want to use it as an API inside your extension, it's necessary to read this!

4.2. Overview

Take a look at the following figure to get a first impression of pt_extlist's architecture:

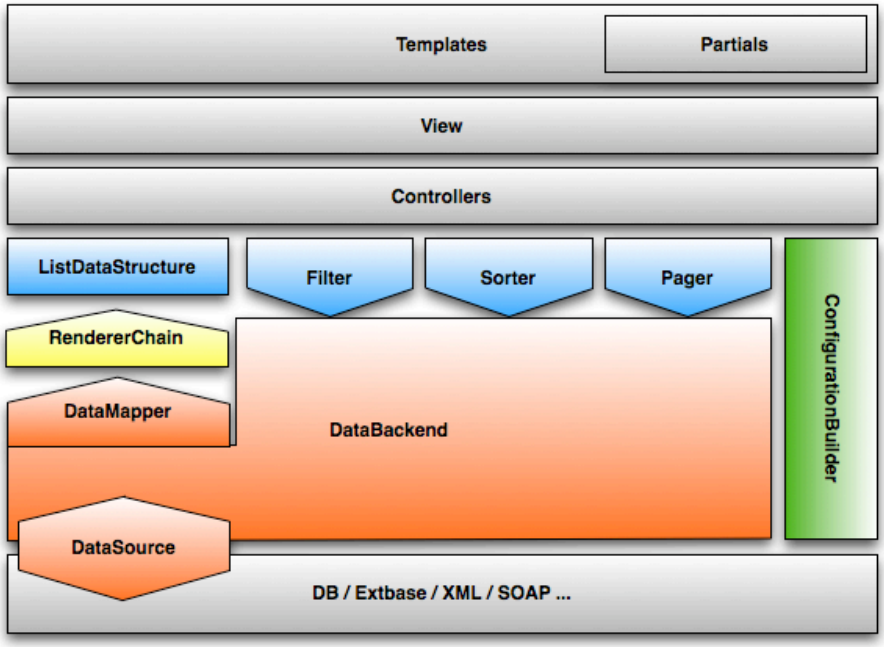

Figure 2.18. Basic Architecture of pt_extlist

The red elements stand for data-related stuff. DataSource gets our data from whatever source we use. Sources can be TYPO3 databases as well as arbitrary MySQL databases and Extbase repositories.

The green elements are configuration-specific which means that they pull our configuration from TypoScript or Flexforms and do various merging on them to generate some nice-to-handle objects that are used for all configuration inside pt_extlist.

The blue elements are what we would like to call our Domain Models as they handle things like lists, filters, pagers etc. which is the core domain of our extension.

Finally the yellow elements stand for all objects that handle the rendering of our list data. As we will see, this can get quite complex.

Our extension architecture is surrounded by some out-of-the box framework stuff provided mainly by Extbase and TYPO3.

4.3. Basic Architecture

4.3.1. Modell-View-Controller (MVC)

4.3.2. Independent Components

4.3.3. pt_extlist Lifecycle

4.4. Configuration

Regarding our extension in a bottom-up fashion, we can easily start with configuration as its the basic thing to have before we can set up anything else. As you might have suggested, most of our configuration is using in TypoScript. Besides there are some settings coming from Flexform or surrounding extensions. We covered this problem inside pt_extlist using what we call a configurationBuilder. Roughly speaking, it is an object that handles the creation of all configuration stuff we need. As we did not like fiddling around with arrays, we started to implement so called configuration objects for all the configuration that we needed. The main ascpects of those objects is to make sure that the required settings are there and correct. Configuration objects are passed to all other objects that require some kind of configuration.

Take a look at this diagram to get an idea of how everything works together:

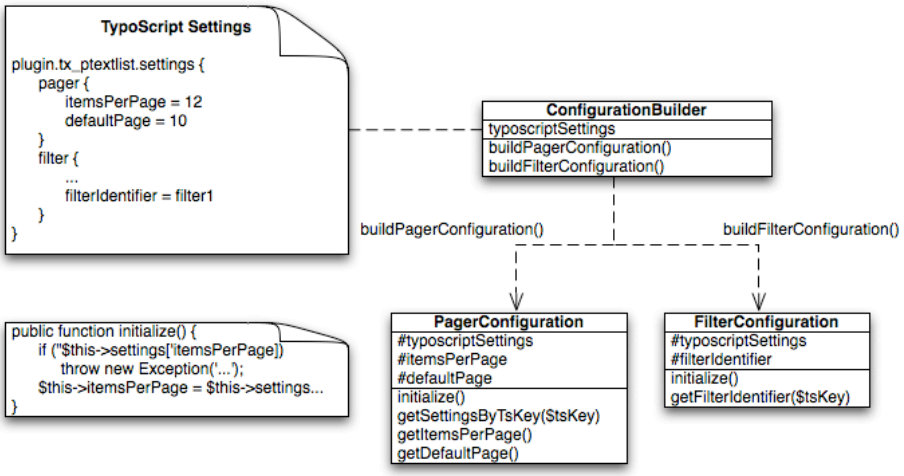

Figure 2.19. Configuration Builder scheme

So what's the basic idea of our so-called configuration objects?

1. TypoScript is a mighty tool for setting up configuration stuff. Unfortunatly there is no way to check for existence and correctness of settings. Our configuration objects jump in the gap here and present a single point of checking configuration.
2. Whenever you need a configuration set in TypoScript in your code, you have to check whether it is set or not. This probably has to be done several times and has to be changed several times whenever your configuration needs to be changed. Configuration objects enable you to keep your TS and your program logic synchronized by only changing code in one single class.
3. Configurations objects provide you with an object-oriented style of accessing configuration settings. You no longer have to fiddle around with arrays when you want to access configuration, now there are Getters and Setters and things like code-completion.

Before we had what we call the configuration builder, objects had to handle their configuration by themselves which made it necessary to implement huge init()-methods for setting up all configuration stuff. Whenever an object was not a singleton, the configuration stuff had to be run time and time again for every instantiation of an object.

As the configuration builder is a extension-wide singleton class which caches configuration objects once they are created, we now can simply grab a configuration from it, every time we need it and do not have to cope with checking the configuratin within our domain objects which makes them a lot simplier to implement and understand.

4.5. Handling State

Since Extbase has been around, handling with persistence of domain objects isn't that a big problem anymore. We use our repositories for finding, updating and deleting objects and that's it. But what about session persistence. Let's say, a user filters a list with some complicated criterias, selects one single record of it to look at and then comes back to his filtered list. Shouldn't it show the state it had before he opened the single record?

Such functionality is handled in pt_extlist with what we called a "Session-Persistence-Manager". Roughly speaking it is a container to which you can register objects that get a portion of data from the session when the object is created and can store data back to the session when the lifecycle ends.

Here is a sketch of how session persistence is working within pt_extlist:

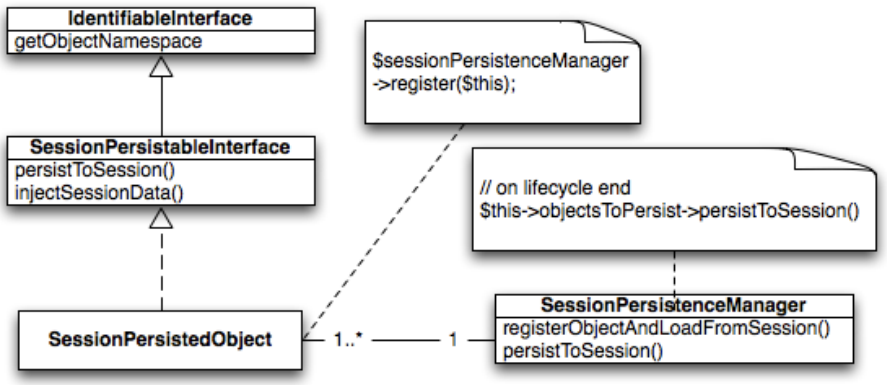

Figure 2.20. Session Persistence in pt_extlist

Please mind, that an object creates its own namespace within the session using the IdentifiableInterface. As we cannot automatically create a namespace for an object this method is implemented individually in each object that wants to use session persistence.

Another place where objects can gather state from are GET and POST parameters. Coming back to our filter example from above, a user could change a previously session-persisted filter so the data coming from the POST vars should overwrite the session settings.

Therefore, we introduced a GET/POST-Var adapter that works somehow similar as the session persistence manager, of course without being able to write back data to it as this would not make any sense.

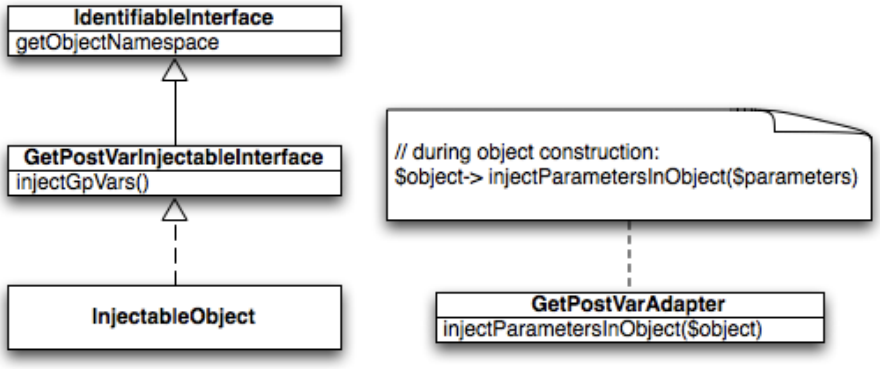

Figure 2.21. GET/POST Var Adapter

The GET/POST-Var adapter also handles the extension and instance namespace. Think about the following situation: you have two instances of pt_extlist plugin on the same page. Each instance comes with its own filters, pagers etc. and the should not influence each other. As an example think about a shopping page, where you have a list of articles on the left and a overview of your shopping cart as a second list on the right.

What the GP-Var adapter does is adding the list identifier (e.g. 'articleList' and 'shoppingCartList') to the namespace of the objects to differentiate between the two instances of the plugin.

4.6. The Data Backend

You can talk about the Data Backend as the heart of the pt_extlist extension. Somehow almost everything comes together here. Most of the pt_extlist components like filters, lists, pagers and breadcrumbs etc. influence the data backend or are themselves influenced or created by the data backend.

Besides being the "glue" holding all components of the extension together, the data backend has the taks of communicating with the "outer-world". This means, that whatever data is displayed within the extension is gathered by the backend from whatever datasource it is working on. The data backend also knows how to translate generic queries and constraints created from the components before they are passed to the data source.

###TODO### insert diagram for data backend

4.6.1. The Query Object

Former implementations of pt_extlist where bound to SQL-databases as a databackend. All queries created were SQL queries so you could directly create SQL snippets within all classes that manipulated the query. E.g. a filter class created some WHERE-clauses, a pager created a LIMIT-clause and a column header created an ORDER BY-clause for sorting.

One big change that comes with pt_extlist was being able to use whatever data source you like as a data backend. We therefore hat to generalize all queries that are created within our classes.

For this purpose, we introduced a so-called query object that can handle common SQL-query like functionality in an object oriented manner. Queries can take constraints (WHERE-clauses), limitations (LIMIT-clausses), sortings (ORDER BY-clauses) and some other stuff. But it's kept in a form that enables us to transform the query to whatever backend we want to use. At the moment there is a TYPO3 backend, using the actual TYPO3-database as a datasource and an Extbase backend that uses Repositories as data sources.

In order to send a query to the corresponding datasource, we have to translate it to a language that can be handled by the backend. We therefore introduced interpreters. Compared to the Extbase query object, those interpreters are independent of the query object itself, so that you can implement your own interpreters for whatever backend you want to support (e.g. XML via XPath or XQuery, SOAP, REST...).

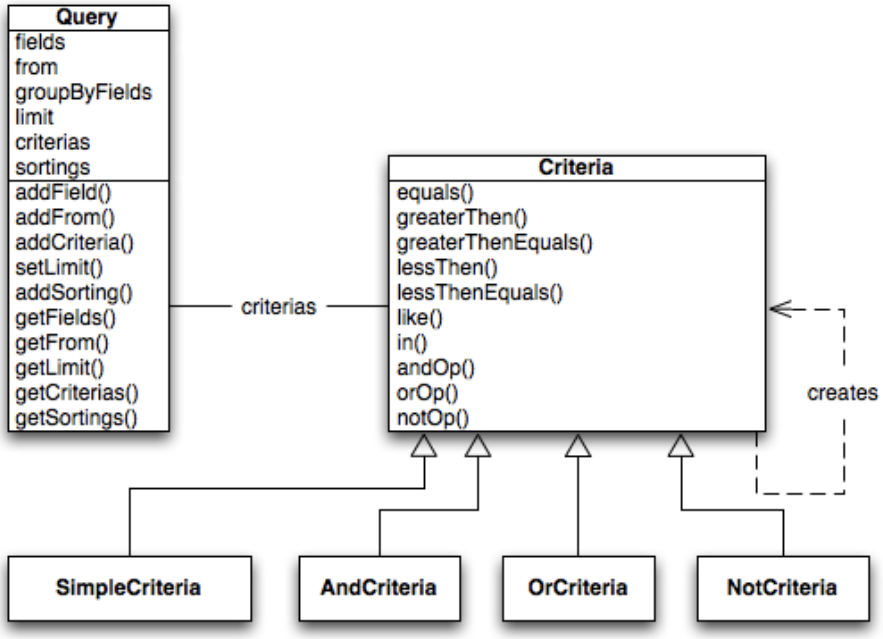

Figure 2.22. Query Object

The translation is handled by Interpreters shipping with a data-backend. Not every data-backend requires its own Interpreter, for example the MySQL backend and the TYPO3 backend share a common Interpreter as they both use SQL.

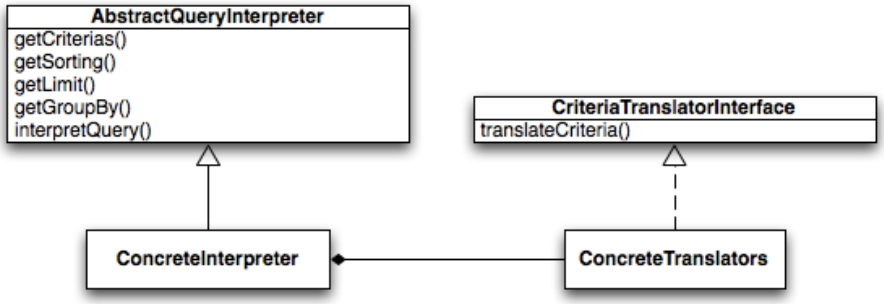

Figure 2.23.

4.6.2. Data Sources

4.6.3. Data Mappers

4.7. The domain model

4.7.1. The List object

4.7.2. Filters

4.7.3. Pager

4.7.4. BreadCrumbs

4.7.5. Bookmarks

4.8. List Rendering

4.8.1. The Renderer Chain

5. API

In this section you will learn how to use pt_extlist's API to extend the extension or use it within third-party extensions.

5.1. Extending pt_extlist

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.

5.1.1. 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:

Tx_PtExtlist_Domain_Model_Filter_FilterInterface
- getErrorMessage()
- getFilterBoxIdentifier()
- getFilterBreadCrumb()
- getFilterConfig()
- getFilterIdentifier()
- getFilterQuery()
- getListIdentifier()
- init()
- injectDataBackend(Tx_PtExtlist_Domain_DataBackend_DataBackendInterface $dataBackend)
- injectFilterConfig(Tx_PtExtlist_Domain_Configuration_Filters_FilterConfig $filterConfig)
- injectGpVarAdapter(Tx_PtExtlist_Domain_StateAdapter_GetPostVarAdapter $gpVarAdapter)
- injectSessionPersistenceManager(Tx_PtExtlist_Domain_StateAdapter_SessionPersistenceManager $sessionPersistenceManager)
- isActive()
- reset()
- validate()

Figure 2.24. 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:

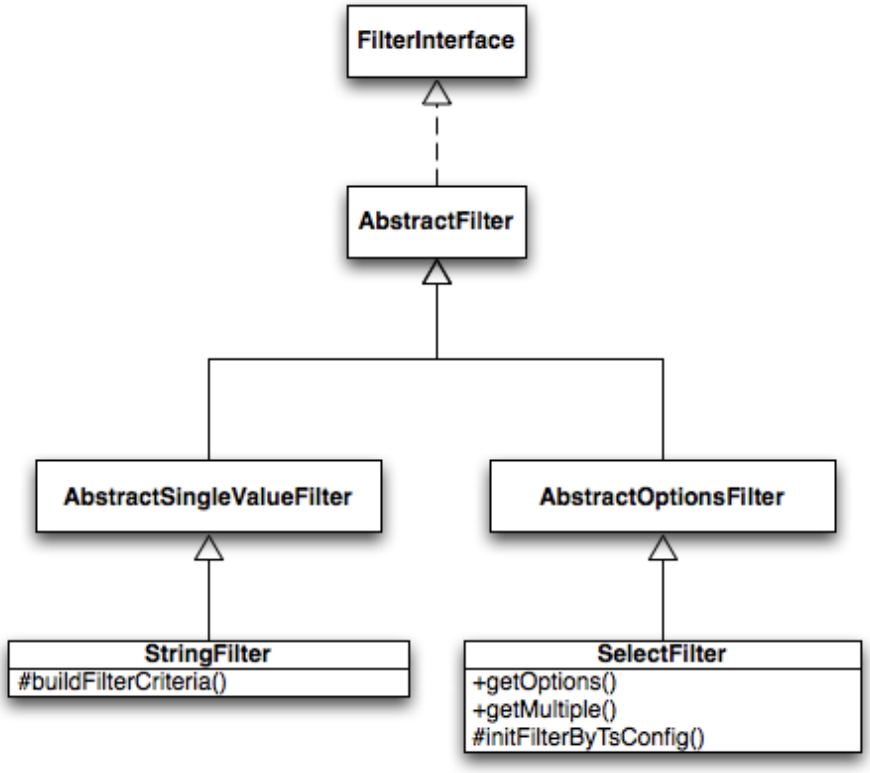

Figure 2.25. Abstract Filter Classes

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.

5.1.1.1. 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, take 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.

6. Coobook

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

7. pt_extlist

TypoSript Reference

plugin.tx_ptextlist.settings

plugin.tx_ptextlist.settings

Description

Main TS-key for all pt_extlist settings.
NO
StdWrap:

Child elements

listConfig,

listConfig

listConfig

Description

Holds configuration for all list identifiers configured by array key.
Datatype: Associative Array (listIdentifier => listConfiguration)
NO

Child elements

[yourListId],

[yourListId]

[yourListId]

Description

Holds configuration for a single list identifier
NO

Example

```
here comes some sample code
```

Child elements

default, backendConfig, fields, columns, rendererChain, aggregateData, aggregateRows, filters, headerPartial, bodyPartial, agregateRowsPartial,

default

default

Description

List default values.
NO

Child elements

sortingColumn,

sortingColumn

sortingColumn

Description

The default sorting column while no other sortng ist set.
Datatype: String
Posible values: Any column identifier
NO

backendConfig

backendConfig

Description

Holds the configuration for the used data backend.
NO

Variants

backendConfig.mysql, backendConfig.typo3, backendConfig.extbase,

Variants of backendConfig:

backendConfig.mysql

backendConfig.mysql

Description

Typo3 Backend. Inherits the Features from the MySql Backend but the datasource is already configured. The Typo3 Backend handles enableFields automatically in where clauses.
NO
StdWrap: plugin.tx_ptextlist.prototype.backend.mysql

Child elements

datasource, baseFromClause, baseWhereClause, baseGroupByClause,

datasource

datasource

Description

NO

baseFromClause

baseFromClause

Description

Holds the FROM clause of a mysql query without the FROM keyword.
Datatype: String
YES

Example

```
static_countries
LEFT JOIN static_territories AS st_subcontinent ON (static_countries.cn_parent_tr_iso_nr
 = st_subcontinent.tr_iso_nr)
```

baseWhereClause

baseWhereClause

Description

Holds the WHERE clause of a mysql query without the WHERE keyword.

NO

Example

```
st_continent.tr_name_en [] ''
```

baseGroupByClause

baseGroupByClause

Description

Holds the GROUP BY clause of a mysql query without the GROUP BY keyword.
Datatype: String
YES

Example

```
st_continent
```

backendConfig.typo3

backendConfig.typo3

Description

Typo3 Backend. Inherits the Features from the MySql Backend but the datasource is already configured. The Typo3 Backend handles enableFields automatically in where clauses.
NO

Example

```
backendConfig < plugin.tx_ptextlist.prototype.backend.typo3
backendConfig {
 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 <> ''
 )
}
```

Child elements

tables,

tables

tables

Description

'Enable fields where clauses' are applied to all tables given.
Datatype: Boolean
Posible values: 0,1
NO

Example

```
static_countries,
static_territories st_continent
```

backendConfig.extbase

backendConfig.extbase

Description

Extbase Repository Backend.
NO
StdWrap: plugin.tx_ptextlist.prototype.backend.extbase

fields

fields

Description

Defines raw datasource fields, wich can than combined and processed in to table fields.
Datatype: Associative array
NO

Child elements

[yourFieldId],

[yourFieldId]

[yourFieldId]

Description

Named definition of a single data field.
Datatype: String
NO

Example

```
name_local {
```

```
 table = static_countries
 field = cn_short_local
 isSortable = 1
}
```

Child elements

table, field, special, isSortable, expandGroupRows,

table

table

Description

The table, this field belongs to.
Datatype: String
NO

field

field

Description

Name of the tables field.
NO

special

special

Description

Insert a individual SQL snippet.
NO

isSortable

isSortable

Description

NO

expandGroupRows

expandGroupRows

Description

NO

columns

columns

Description

Holds the tables column definitions.
Datatype: Array
NO

Example

```
10 {
 label = LLL:EXT:pt_extlist/Configuration/TypoScript/Demolist/
locallang.xml:cn_short_localColumn
 columnIdentifier =  cn_short_localColumn
 fieldIdentifier =  cn_short_local
 isSortable = 1
}
```

Child elements

10,20,30,

10,20,30

10,20,30

Description

NO

Child elements

fieldIdentifier, label, renderUserFunctions, renderTemplate, renderObj, sorting, sortingImageAsc, sortingImageDesc, sortingImageDefault, accessGroups, cellCSSClass,

fieldIdentifier

fieldIdentifier

Description

Identifier of a defined field.
NO

label

label

Description

A label for this element.
Datatype: String
YES

renderUserFunctions

renderUserFunctions

Description

A list of userfunctions to render the field value.

NO

Example

```
renderUserFunctions {
 10 = EXT:pt_extlist/Resources/Private/UserFunctions/
class.tx_ptextlist_demolist_renderer.php:tx_ptextlist_demolist_renderer-
>iso2CodeRenderer
}
```

renderTemplate

renderTemplate

Description

NO

renderObj

renderObj

Description

NO

sorting

sorting

Description

NO

sortingImageAsc

sortingImageAsc

Description

NO

sortingImageDesc

sortingImageDesc

Description

NO

sortingImageDefault

sortingImageDefault

Description

NO

accessGroups

accessGroups

Description

NO

cellCSSClass

cellCSSClass

Description

NO

rendererChain

rendererChain

Description

Holds the renderer configuration.
NO
StdWrap: plugin.tx_ptextlist.prototype.rendererChain

Child elements

enabled, rendererConfigs,

enabled

enabled

Description

Datatype: Boolean
Posible values: 0,1
Default: 1
NO

rendererConfigs

rendererConfigs

Description

A list of chained renderer classes that work on list data structures. The default renderer class uses the column configuration, to render the list of field data in a list of rows and columns. By default this renderer iscalled at position 100. All defined renderer before 100 work on a field data list, while renderer after 100 work on a column list data structure.
Datatype: Array (10,20,30)

NO

Child elements
[yourNumericRendererId],

[yourNumericRendererId]

[yourNumericRendererId]

Description

Configuration of a single renderer.
Posible values: 10,20,30
NO

Child elements
renderClassName,

renderClassName

renderClassName

Description

The class name of the renderers php class.
Datatype: String
NO

aggregateData

aggregateData

Description

Defines aggregates of data fields.
Datatype: Associative array
NO

Child elements
[yourAggregateFieldId],

[yourAggregateFieldId]

[yourAggregateFieldId]

Description

Named definition of a single data field.
Datatype: Associative array
NO

Example

```
sumPhone {
 fieldIdentifier = phone
 method = sum
 scope = query
}
```

Child elements

method, scope,

method

method

Description

Defined aggregate methods.
Datatype: String
Posible values: min,max,sum,avg
NO

scope

scope

Description

The scope for the aggregation can be either set to the current page or to the whole query. Aggregates for the current page are calculated internally without an additonal database query.
Datatype: String
Posible values: page,query
NO

aggregateRows

aggregateRows

Description

Holds the aggregates columns definitions.
Datatype: Associative array
Posible values: All columnIdentifiers
NO

Child elements

[yourColumnId],

[yourColumnId]

[yourColumnId]

Description

NO

Child elements
aggregateDataIdentifier,

aggregateDataIdentifier

aggregateDataIdentifier

Description

Datatype: String
NO

filters

filters

Description

Holds all filterbox configurations.
Datatype: Associative array
NO

Child elements
[yourFilterBoxId],

[yourFilterBoxId]

[yourFilterBoxId]

Description

NO
StdWrap: plugin.tx_ptextlist.prototype.filterBox

Child elements
showReset, showSubmit, filterConfigs,

showReset

showReset

Description

Show a reset link for all filters of this filterBox.
Datatype: Boolean
Posible values: 0,1
Default: 1
NO

showSubmit

showSubmit

Description

Show a submit button for this filterBox.
Datatype: Boolean
Posible values: 0,1
Default: 1
NO

filterConfigs

filterConfigs

Description

Holds the configuration of the filters of this filter box.
Datatype: Array
Posible values: 10,20,30...
NO

Child elements

[yourNumericFilterId],

[yourNumericFilterId]

[yourNumericFilterId]

Description

NO

Variants

String, Select, Checkbox, radiobutton, firstletter, Proxy,

Variants of [yourNumericFilterId]:

String

String

Description

Shows a string filter.
NO
StdWrap: plugin.tx_ptextlist.prototype.filter.string

Example

```
10  < plugin.tx_ptextlist.prototype.filter.string
10 {
 filterIdentifier = firstNameSearch
 label = Firstname
 fieldIdentifier = firstName
}
```

Child elements

filterIdentifier, label, defaultValue, accessGroups, filterClassName, partialPath, invert, invertable,

filterIdentifier

filterIdentifier

Description

The unique identifier of this filter.
Datatype: String
NO

label

label

Description

The label wich is displayed beside the filter.
Datatype: String
NO

defaultValue

defaultValue

Description

The default value which is shown or selected by default.
Datatype: String
NO

accessGroups

accessGroups

Description

Comma separated list of user groups wich have access to this filter.
Datatype: Comma eparated list.
Posible values: Typo3 Group Ids
NO

filterClassName

filterClassName

Description

Name of the PHP Class of this filter.
Datatype: String
NO

Example

```
Tx_PtExtlist_Domain_Model_Filter_StringFilter
```

partialPath

partialPath

Description

NO

invert

invert

Description

Invert the constraint of this filter.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

invertable

invertable

Description

Show a controle to invert this filter.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

Select

Select

Description

NO
StdWrap: plugin.tx_ptextlist.prototype.filter.select

Child elements

filterIdentifier, fieldIdentifier, label, partialPath, filterClassName, defaultValue, accessGroups, invert, invertable, displayFields, multiple, excludeFilters, showRowCount, submitOnChange, inactiveOption, inactiveValue,

filterIdentifier

filterIdentifier

Description

The unique identifier of this filter.
Datatype: String
NO

fieldIdentifier

fieldIdentifier

Description

Identifier of a defined data field. The filter affects on this data field.
Datatype: String
NO

label

label

Description

The label wich is displayed beside the filter.
Datatype: String
NO

partialPath

partialPath

Description

NO

filterClassName

filterClassName

Description

Name of the PHP Class of this filter.
Datatype: String
NO

Example

```
Tx_PtExtlist_Domain_Model_Filter_StringFilter
```

defaultValue

defaultValue

Description

The default value which is shown or selected by default.

Datatype: String
NO

accessGroups

accessGroups

Description

Comma separated list of user groups wich have access to this filter.
Datatype: Comma eparated list.
Posible values: Typo3 Group Ids
NO

invert

invert

Description

Invert the constraint of this filter.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

invertable

invertable

Description

Show a controle to invert this filter.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

displayFields

displayFields

Description

One or multiple identifiers of previously defined data fields.
Datatype: String / Comma separated list
Posible values: Previously defined field indentifier.
Default: If not set, the field defined in fieldIdentifier is used.
NO

multiple

multiple

Description

Datatype: Boolean
Posible values: 0,1
Default: 0
NO

excludeFilters

excludeFilters

Description

List of filters that are not considered for the group query of this filter.
Datatype: String / Comma separated list of filterIdentifiers.
NO

showRowCount

showRowCount

Description

Show the rowcount if the select filter is filled with grouped data.
Datatype: Boolean
Posible values: 0,1
Default: 1
NO

submitOnChange

submitOnChange

Description

Instant submit filter if a value is selected.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

inactiveOption

inactiveOption

Description

Label of an option that is added to the select list. If this option is selected, the filter is inactive.
Datatype: String
NO

inactiveValue

inactiveValue

Description

The submitted value for the inactiveOption described above.
Datatype: String
NO

Checkbox

Checkbox

Description

NO
StdWrap: plugin.tx_ptextlist.prototype.filter.checkbox

Child elements

filterIdentifier, fieldIdentifier, label, partialPath, filterClassName, defaultValue, accessGroups, invert, invertable, displayFields, multiple, excludeFilters, showRowCount, submitOnChange, inactiveOption, inactiveValue,

filterIdentifier

filterIdentifier

Description

The unique identifier of this filter.
Datatype: String
NO

fieldIdentifier

fieldIdentifier

Description

Identifier of a defined data field. The filter affects on this data field.
Datatype: String
NO

label

label

Description

The label wich is displayed beside the filter.
Datatype: String
NO

partialPath

partialPath

Description

NO

filterClassName

filterClassName

Description

Name of the PHP Class of this filter.
Datatype: String
NO

Example

```
Tx_PtExtlist_Domain_Model_Filter_StringFilter
```

defaultValue

defaultValue

Description

The default value which is shown or selected by default.
Datatype: String
NO

accessGroups

accessGroups

Description

Comma separated list of user groups wich have access to this filter.
Datatype: Comma eparated list.
Posible values: Typo3 Group Ids
NO

invert

invert

Description

Invert the constraint of this filter.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

invertable

invertable

Description

Show a controle to invert this filter.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

displayFields

displayFields

Description

One or multiple identifiers of previously defined data fields.
Datatype: String / Comma separated list
Posible values: Previously defined field indentifier.
Default: If not set, the field defined in fieldIdentifier is used.
NO

multiple

multiple

Description

Datatype: Boolean
Posible values: 0,1
Default: 0
NO

excludeFilters

excludeFilters

Description

List of filters that are not considered for the group query of this filter.
Datatype: String / Comma separated list of filterIdentifiers.
NO

showRowCount

showRowCount

Description

Show the rowcount if the select filter is filled with grouped data.
Datatype: Boolean

Posible values: 0,1
Default: 1
NO

submitOnChange

submitOnChange

Description

Instant submit filter if a value is selected.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

inactiveOption

inactiveOption

Description

Label of an option that is added to the select list. If this option is selected, the filter is inactive.
Datatype: String
NO

inactiveValue

inactiveValue

Description

The submitted value for the inactiveOption described above.
Datatype: String
NO

radiobutton

radiobutton

Description

NO
StdWrap: plugin.tx_ptextlist.prototype.filter.checkbox

Child elements

filterIdentifier, fieldIdentifier, label, partialPath, filterClassName, defaultValue, accessGroups, invert, invertable, displayFields, multiple, excludeFilters, showRowCount, submitOnChange, inactiveOption, inactiveValue,

filterIdentifier

filterIdentifier

Description

The unique identifier of this filter.
Datatype: String
NO

fieldIdentifier

fieldIdentifier

Description

Identifier of a defined data field. The filter affects on this data field.
Datatype: String
NO

label

label

Description

The label wich is displayed beside the filter.
Datatype: String
NO

partialPath

partialPath

Description

NO

filterClassName

filterClassName

Description

Name of the PHP Class of this filter.
Datatype: String
NO

Example

```
Tx_PtExtlist_Domain_Model_Filter_StringFilter
```

defaultValue

defaultValue

Description

The default value which is shown or selected by default.

Datatype: String
NO

accessGroups

accessGroups

Description

Comma separated list of user groups wich have access to this filter.
Datatype: Comma eparated list.
Posible values: Typo3 Group Ids
NO

invert

invert

Description

Invert the constraint of this filter.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

invertable

invertable

Description

Show a controle to invert this filter.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

displayFields

displayFields

Description

One or multiple identifiers of previously defined data fields.
Datatype: String / Comma separated list
Posible values: Previously defined field indentifier.
Default: If not set, the field defined in fieldIdentifier is used.
NO

multiple

multiple

Description

Datatype: Boolean
Posible values: 0,1
Default: 0
NO

excludeFilters

excludeFilters

Description

List of filters that are not considered for the group query of this filter.
Datatype: String / Comma separated list of filterIdentifiers.
NO

showRowCount

showRowCount

Description

Show the rowcount if the select filter is filled with grouped data.
Datatype: Boolean
Posible values: 0,1
Default: 1
NO

submitOnChange

submitOnChange

Description

Instant submit filter if a value is selected.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

inactiveOption

inactiveOption

Description

Label of an option that is added to the select list. If this option is selected, the filter is inactive.
Datatype: String
NO

inactiveValue

inactiveValue

Description

The submitted value for the inactiveOption described above.
Datatype: String
NO

firstletter

firstletter

Description

NO
StdWrap: plugin.tx_ptextlist.prototype.filter.firstletter

Child elements

filterIdentifier, fieldIdentifier, label, partialPath, filterClassName, defaultValue, accessGroups, invert, invertable, displayFields, multiple, excludeFilters, showRowCount, submitOnChange, inactiveOption, inactiveValue,

filterIdentifier

filterIdentifier

Description

The unique identifier of this filter.
Datatype: String
NO

fieldIdentifier

fieldIdentifier

Description

Identifier of a defined data field. The filter affects on this data field.
Datatype: String
NO

label

label

Description

The label wich is displayed beside the filter.
Datatype: String
NO

partialPath

partialPath

Description

NO

filterClassName

filterClassName

Description

Name of the PHP Class of this filter.
Datatype: String
NO

Example

```
Tx_PtExtlist_Domain_Model_Filter_StringFilter
```

defaultValue

defaultValue

Description

The default value which is shown or selected by default.
Datatype: String
NO

accessGroups

accessGroups

Description

Comma separated list of user groups wich have access to this filter.
Datatype: Comma eparated list.
Posible values: Typo3 Group Ids
NO

invert

invert

Description

Invert the constraint of this filter.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

invertable

invertable

Description

Show a controle to invert this filter.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

displayFields

displayFields

Description

One or multiple identifiers of previously defined data fields.
Datatype: String / Comma separated list
Posible values: Previously defined field indentifier.
Default: If not set, the field defined in fieldIdentifier is used.
NO

multiple

multiple

Description

Datatype: Boolean
Posible values: 0,1
Default: 0
NO

excludeFilters

excludeFilters

Description

List of filters that are not considered for the group query of this filter.
Datatype: String / Comma separated list of filterIdentifiers.
NO

showRowCount

showRowCount

Description

Show the rowcount if the select filter is filled with grouped data.
Datatype: Boolean

Posible values: 0,1
Default: 1
NO

submitOnChange

submitOnChange

Description

Instant submit filter if a value is selected.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

inactiveOption

inactiveOption

Description

Label of an option that is added to the select list. If this option is selected, the filter is inactive.
Datatype: String
NO

inactiveValue

inactiveValue

Description

The submitted value for the inactiveOption described above.
Datatype: String
NO

Proxy

Proxy

Description

A proxy filter is not displayed. It uses a filter from an other list defined by the proxy path and sets the result constraint to the given field.
NO
StdWrap: plugin.tx_ptextlist.prototype.filter.proxy

Child elements

proxyPath,

proxyPath

proxyPath

Description

Path to a filter from an other list. The path has the format [listId].[filterBoxId].[filterId]
Datatype: String
NO

Child elements

filterIdentifier, defaultValue, label, accessGroups, filterClassName, partialPath, invert,

filterIdentifier

filterIdentifier

Description

The unique identifier of this filter.
Datatype: String
NO

defaultValue

defaultValue

Description

The default value which is shown or selected by default.
Datatype: String
NO

label

label

Description

The label wich is displayed beside the filter.
Datatype: String
NO

accessGroups

accessGroups

Description

Comma separated list of user groups wich have access to this filter.
Datatype: Comma eparated list.
Posible values: Typo3 Group Ids
NO

filterClassName

filterClassName

Description

Name of the PHP Class of this filter.
Datatype: String
NO

Example

```
Tx_PtExtlist_Domain_Model_Filter_StringFilter
```

partialPath

partialPath

Description

NO

invert

invert

Description

Invert the constraint of this filter.
Datatype: Boolean
Posible values: 0,1
Default: 0
NO

headerPartial

headerPartial

Description

Path to the header partial.
Datatype: String
NO

bodyPartial

bodyPartial

Description

Path to the body partial.
Datatype: String
NO

agregateRowsPartial

agregateRowsPartial

Description

Path to the aggregate row partial.
Datatype: String
NO

8. Extending pt_extlist

This chapter is mainly for developing and extending pt_extlist.

8.1. RenderChain

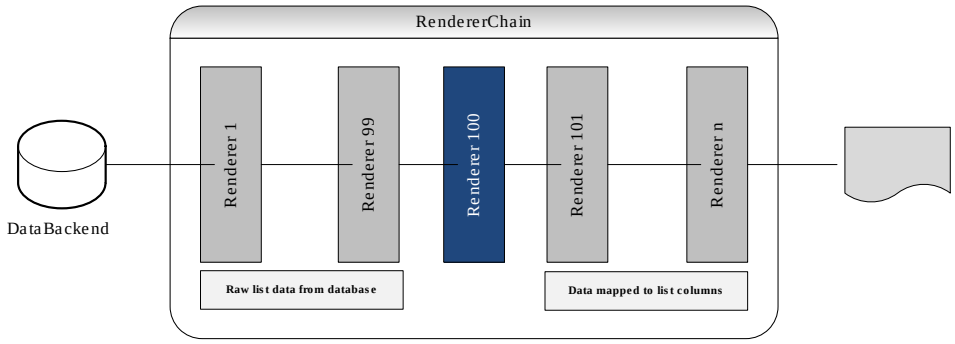

Figure 2.26. RenderChain

9. Glossary

Glossary

E

Extended Markup Language — A set of rules for encoding documents in machine-readable form. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards.