EXT: Hijax

Extension Key: extbase_hijax
Language: en
Keywords: extbase, hijax, forDevelopers, forAdvanced

Table of Contents

EXT: Hijax....1
Introduction....3
What does it do?....3
Screenshots....3
Warning....3
Developers manual....4
Installation/Configuration....4
AJAXifying Extbase forms....4
JavaScript-based IF Fluid viewhelper....5
Event-driven programming....5
Cache control....8
Known problems....10
To-Do list....11
ChangeLog....12

Introduction

What does it do?

Hijax is a term for the application of progressive enhancement to AJAX. It is also known as Simplified Ajax. Hijax is a technique for building web applications that degrade gracefully by only using AJAX techniques to 'hijack' form submissions and responses, updating the DOM to reflect the results of the request/response cycle. The goal of this extension is to enable easy integration of the hijax technique in TYPO3, namely in Extbase and Fluid.

In addition to plain hijaxing, extbase_hijax does a bit more that you could conclude by analyzing its name. The most notable features are:

- Ability to ajaxify any Fluid form,
- JavaScript-based IF Fluid viewhelper,
- Allows event-driven programming between any FE plugin instance,
- Automatic cache control.

Screenshots

No screenshots available as this extension is for developers use only.

Warning

This extension is still in alpha phase. It is used on a real projects and is stable, but there is actually no official API which will be safe to use. So you need to either stick to a tag, or follow the changes closely.

Developers manual

Installation/Configuration

Installation and configuration should be really simple:

- Install the extension using Extension Manager,
- Make sure that jQuery JS library is included on your website (extbase_hijax won't do that for you :D),
- Include the extbase_hijax TypoScript configuration in your TypoScript template:

```
<INCLUDE_TYPOSCRIPT: source="FILE: EXT:extbase_hijax/Configuration/TypoScript/setup.txt">
```

(There is no static TypoScript template registered for the use in the BE)

AJAXifying Extbase forms

You can ajaxify any Fluid form by switching it to extbase_hijax's form viewhelper. For example, lets look at the existing Fluid form (taken from the aoe_poll extension):

```
<f:form action="create" controller="Vote" object="{newVote}" name="newVote">
	<!--
		// form fields
	-->
</f:form>
```

In order to ajaxify it you just need to import the extbase_hijax namespace (differences are bolded):

```
{namespace h=Tx_ExtbaseHijax_ViewHelpers}

<h:form action="create" controller="Vote" object="{newVote}" name="newVote">
	<!--
		// form fields
	→
</h:form>
```

That's it – you form will now work with AJAX. If a client doesn't have JavaScript support turned on, the form will behave just like a regular Fluid form.

Styling the loader

If you tried the sample above on one of your existing forms you might notice that there's a default extbase_hijax loader with class **hijax-loading**. Every extbase_hijax element (and form is one of them, at least in the current implementation) has the following DOM structure:

```
<div class="hijax-element"...>
	<div class="hijax-content">
		<!-- actual element content -->
	</div>
	<div class="hijax-loading"><!-- --></div>
</div>
```

Please note that the elements used might not be DIVs (in case of form, the .**hijax-element** is in fact **form**).

Make sure that you are not using hijax-element and hijax-content classes in your CSS – use only the CSS file provided with the extbase_hijax extension.

You can however style the .**hijax-loading** element by override background-color, background-image and opacity attributes.

Setting the target for ajax results

By default the result received from AJAX call on form submit will be loaded in the place of the form element (warning – this will be for sure improved in near future). You can override that by using new resultTarget attribute of the form viewhelper (differences are bolded):

```
{namespace h=Tx_ExtbaseHijax_ViewHelpers}

<h:form action="create" controller="Vote" object="{newVote}" name="newVote"
resultTarget="jQuery(this).parent()">
	<!--
		// form fields
	→
</h:form>
```

In this example the result will be loaded in the form's parent element. This attribute has to be a valid JavaScript expression. If you need to check in your extension whether the action is called from AJAX or regular version (in order to provide possibly slightly different markup), you can use the following function call:

```
if ($this->objectManager->get('Tx_ExtbaseHijax_Utility_Ajax_Dispatcher')->getIsActive()) {
      ...
}
```

WARNING

It is planned to slightly change the default behavior of the form viewhelper which doesn't have the resultTarget attribute set in near future. This change will allow easier usage of the form viewhelper in situations when the plugin output includes a bit more markup around the ajaxified form.

This change should be seamless, but you are warned to test everything in detail upon upgrading. Extbase_hijax extension is still in the brainstorming (read alpha) phase meaning that big changes can be expected.

JavaScript-based IF Fluid viewhelper

This one is also really simple and very useful. Works the same way as the Fluid's built-in IF viewhelper with only one difference – the condition should be JavaScript expression as it is evaluated on client-side on DOM ready event. Sample (again from aoe_poll):

```
{namespace h=Tx_ExtbaseHijax_ViewHelpers}

<h:if condition="jQuery.cookie('{poll.cookieIdentifier}')!=1">
      <h:then>
            <f:render partial="Poll/Form" arguments="{poll: poll, newVote: newVote}"/>
      </h:then>
      <h:else>
            <f:render partial="Poll/Results" arguments="{poll: poll}"/>
      </h:else>
</h:if>
```

If a user does not have JS support enabled, he will always see the content from the THEN node.

Event-driven programming

Problem

Imagine the following common website (a portal) with info about the currently logged in user (such as username, avatar image...) displayed in the header. On this website, a user should have the option to edit his account details. This will be the most common situation:

- User data in the header is always rendered first,
- Plugin for edit account information is rendered afterwards.

If a user edits his account data, the controller action doing the DB change will be fired upon plugin rendering completion. In that moment, info for the header is rendered with old data. So this might end up with a situation where old info is displayed in the header and bellow that there's a message that the user info is successfully changed.

Sure, you can solve this with some additional redirect, but things can get a hell more complex (and havier for execution on the server) when there are more than 2 plugins which need to interact.

Solution

Extbase_hijax implements completely new way of communicating between FE plugin instances. It allows fully event-driven programming of Extbase based extensions. At first, you might think that this already exist in form of SignalSlots. Extbase_hijax events are however not replacement for SignalSlots, these two event dispatchers can and should be used together (for different purposes of course).

Both implement observer pattern. However, there is one fundamental difference:

- Tx_Extbase_SignalSlot_Dispatcher notifies the listeners immediately,
- Tx_ExtbaseHijax_Event_Dispatcher notifies the listeners **once per rendering cycle**.

If we return to our above-mentioned problem, sending a signal (with for example name *user-updated*) from the edit account plugin through SignalSlot_Dispatcher won't change the already rendered user info in the header. User info is at this point just a (*dead*) string in the $GLOBALS['TSFE']->content :)

So, the idea behind extbase_hijax events is to *revive* the plugin which rendered the user info in the header, make it run again

(with the event passed on to it), and replace the previously rendered content with the new one. In order to do this with easily understandable flow, events from one rendering cycle are passed to designated listeners upon rendering cycle.

TYPO3 by default has only one rendering cycle:

- Render cacheable elements,
- Render INT scripts (COA_INT, USER_INT...) if any,
- If there are new INT scripts return to previous step, otherwise return the content.

Extbase_hijax augments this with iteration of these rendering cycles:

- Render the page with standard TYPO3 rendering cycle,
- If there are events thrown during execution, revive the plugins which subscribed to them, render just their content again and replace it in the final page content,
- If there are new events thrown during execution of the previous step, revive the plugins again, otherwise return the content.

Each of this steps executed after initial page rendering cycle are additional page rendering cycles. There is a built-in prevention to disable dead-loops as those are easy to "implement" with this new event-driven approach.

Ok, so let's get down to the example. Edit account plugin from the above problem description, can dispatch the *user-updated* event in the following way:

```
if (t3lib_extMgm::isLoaded('extbase_hijax')) {
	$event = new Tx_ExtbaseHijax_Event_Event('user-updated', array('user'=>$user));
	$this->objectManager->get('Tx_ExtbaseHijax_Event_Dispatcher')->notify($event);
}
```

User info plugin in the header can connect to that event in the following way:

```
public function someAction() {
	...
	$this->objectManager->get('Tx_ExtbaseHijax_Event_Dispatcher')->connect('user-updated',
array($this, 'onUserUpdate'));
	...
}

/**
 * @var $event Tx_ExtbaseHijax_Event_Event
 */
public function onUserUpdate(Tx_ExtbaseHijax_Event_Event $event) {
		// sample handling of event data
	$params = $event->getParameters();
		/* @var $user Tx_Extbase_Domain_Model_FrontendUser */
	$user = $params['user'];
	...
}
```

The first parameter of the connect method is event name, while the second is a PHP callback. The second parameter is optional. Omitting it will still subscribe the plugin instance to the event, and the rerendering process will run as well – just the callback (onUserUpdate from the sample above) won't be called.

Conclusion

Use Tx_Extbase_SignalSlot_Dispatcher in all occasions where output rendering does not need to be changed. SignalSlot mechanism is actually just a replacement for old TYPO3 hooks and doesn't enable what event-driven programming of web applications is all about.

Event-driven programming is widely used in graphical user interfaces because it has been adopted by most commercial widget toolkits as the model for interaction. By using Tx_ExtbaseHijax_Event_Dispatcher you can now use that in TYPO3 as well.

Events with ajaxifyed forms

Continuing to torture the previous user info and edit account plugins, lets assume that we have proper events implemented and that we ajaxifyed the edit account form plugin. You might assume that we can again get the situation where user info will be out of sync with the edit account plugin messages and data. One very cool feature of extbase_hijax is that you actually don't have to think about it – if you implemented server-side events properly, they will work on the client-side as well.

If you submit the ajaxifyed form on edit account plugin, the data about all event-listeners on your currently opened page will be internally sent together with actual form data. The AJAX dispatcher in extbase_hijax will know what other instances need to be revived and rerun (if some events are thrown during the processing of the ajaxifyed form). The AJAX dispatcher will

return the content of all affected plugins on the current page through only one HTTP call.

What do you need to know about this client-side event processing? At the moment, the whole markup of the affected elements is replaced upon receiving the result from the ajaxifyed form. That might break some of your functionalities, or it might affect some actions the user took in his browser (like filling up the input fields). This can be easily prevented by using two hooks in the JavaScript. (Almost) full sample follows:

```
var layout = {
	/**
	 * Main init function
	 */
	init: function() {
		...
		this.initTabElements();
		...
	},

	restoreState: function() {
		...
		this.initTabElements();
		if (LAYOUT_SETTINGS.topSearchInputVal) {
			jQuery('#topsearch-input').val(LAYOUT_SETTINGS.topSearchInputVal);
		}
		...
	},

	rememberState: function() {
		...
		LAYOUT_SETTINGS.topSearchInputVal = jQuery('#topsearch-input').val();
		..
	}
};

jQuery(document).ready(function(){
	layout.init();
});

	// testing for EXTBASE_HIJAX will prevent your code to throw exception in static markup
if (EXTBASE_HIJAX) {
	EXTBASE_HIJAX.beforeLoadElement.push('layout.rememberState()');
	EXTBASE_HIJAX.onLoadElement.push('layout.restoreState()');
}
```

Bolded is the code which is current API. Please note that this might change as well in future.

Performing redirects

Usually (if you properly use TYPO3 framework), you would perform the redirect by using t3lib_utility_Http::redirect function. That's not good for extensions which use extbase_hijax anymore, as calling TYPO3 core's redirect function would redirect the AJAX response to a HTML page (while the JavaScript callback is expecting JSON or JSONP).

In order to properly redirect from your extbase_hijax powered extension, please use Tx_ExtbaseHijax_Utility_HTTP::redirect instead. It is 100% compatible with t3lib_utility_Http::redirect and is aware of environment from which it is called (regular FE or AJAX call). In both cases, the execution is stopped and the user is being redirected.

Implementing basic event support for TypoScript and 3rd party plugins

There is a designated FE plugin registered for displaying arbitrary TypoScript and adding it ability to listen to exbase_hijax's events. This plugin is available only for usage in TypoScript, you won't find it in the BE. Sample:

```
lib.navigation.meta = USER
lib.navigation.meta {
	extensionName = ExtbaseHijax
	pluginName = Pi1
	userFunc = tx_extbase_core_bootstrap->run
	switchableControllerActions {
		ContentElement {
			0 = user
		}
	}
	settings {
		listenOnEvents = user-loginFailure,user-loggedIn,user-loggedOut,user-updated
		loadContentFromTypoScript = lib.navigation.metaTS
	}
}

lib.navigation.metaTS = COA
```

```
lib.navigation.metaTS {
      …
}
```

In this example, lib.navigation.meta will just render lib.navigation.metaTS and make it revive on any of these 4 events: user-loginFailure, user-loggedIn, user-loggedOut, user-updated. TypoScript object of course won't be able to receive the actual event and data stored within it. But it can refresh the output, which is often the only thing needed.

Cache control

Another very neat feature of extbase_hijax extension actually doesn't have a lot relation to the hijax term itself.

TYPO3 has great caching, but setting up cache invalidation can sometimes become cumbersome, especially when using FE plugins which display records stored in system folder. Often used TCEMAIN.clearCacheCmd pageTS setting is not satisfactory for the following reasons:

- if editors add new plug-in instances, administrator needs to reconfigure the page uids in the clearCacheCmd list,
- if only one record is changed, all versions of a page with single view plugin are cleared from cache.

The idea for enhancement is to leave the caching in TYPO3 as is and to improve the cache invalidation logic. In order to further explain the idea, we need to introduce a couple of new terms:

- **Tracking manager** tracks what records are used on what page (respecting all possible versions of a page). This manager can track usage of records and/or complete repositories on a page. Tracking of records is suitable for single view plugins, while tracking of repositories is suitable for list plugins.
- **Cache invalidation** is a process whereby entries in a cache are deleted.
- **Cache invalidation level** determines when the entries in the cache are deleted. In situations where the cache for a page is not invalidated upon updating of relevant data, some users might get a stale version of the page. This actually does not have to be a bad thing for most of the usages (like social networks, news websites etc). In some areas, having a stale version is usually unacceptable (think stock exchanges or sports betting websites). Each of these use case scenarios can require different level of cache invalidation. The most important thing is to have consistent data across the pages (even it is stale) – all cache entries related to an object need to be invalidated at the same time.

Why it this a part of extbase_hijax at all? We will implement automatic tracking and cache invalidation in future. As this is all done by means of hijacking the extbase core, it has some relation to the term hijack :) Furthermore, the per-element cache will be implemented which will be in use by the extbase_hijax's AJAX dispatcher. So, it will become more and more related in future.

Tracking an object

Tracking of an object usage can be done either from the controller or from the view (which seems more logical). Sample (from aoe_poll) of tracking from the controller:

```
$this->objectManager->get('Tx_ExtbaseHijax_Tracking_Manager')->trackObjectOnPage($poll);
```

Here, $poll is instance of class which extends extbase's Tx_Extbase_DomainObject_AbstractDomainObject. This single call will register display of the $poll object on the given page, but only for the current hash. What does that mean? For example, on a single news page, there's a plugin which can display bunch of different news records, one at a time – or better say, one per page hash. These all records are shown on the same page, but with different page hashes. The function trackObjectOnPage has two more parameters which are optional:

- $type – can be 'hash' (default) or 'id'. Option 'hash' denotes that the object will be tracked only on the current page hash. Option 'id' denotes that the object will be tracked on all hashes for the current page.
- $hash – a bit confusing parameter name. For $type='hash', this is an actual hash for which the record will be tracked (defaults to current hash if omitted). For $type='id', this is an id of the page where the record should be tracked. You usually don't need to set this parameter (if you want to track the object for the current page).

You can track an object in Fluid template as well. Sample:

```
{namespace h=Tx_ExtbaseHijax_ViewHelpers}

<h:trackRecord object="{poll}" clearCacheOnAllHashesForCurrentPage="false">
      <f:render partial="Poll/Results" arguments="{poll: poll}"/>
</h:trackRecord>
```

Attribute clearCacheOnAllHashesForCurrentPage is related to the $type parameter in trackObjectOnPage function. If set to false, cache invalidation would clear the cache on current page hash only. Otherwise, it would clear the cache on all hashes for the current page.

Tracking a repository

In some occasions, you would like to track the entire repository instead of just single records. Such example would be list view – you probably need to clear the cache of that page when the new record is created (and which couldn't be tracked before due to it not existing before creation :D). This can be done at the moment only from the controller. Sample:

```
$this->objectManager->get('Tx_ExtbaseHijax_Tracking_Manager')->trackRepositoryOnPage($this->pollRepository, 'hash');
```

The first parameter can be instance of a repository which implements Tx_Extbase_Persistence_RepositoryInterface. But it can be also an object extending Tx_Extbase_DomainObject_AbstractDomainObject. The function will know how to deduct the proper repository for an object (if object reference is passed). There are also two optional parameters, $type and $hash, which have the same purpose as in trackObjectOnPage function.

There is no viewhelper for this functionality at the moment.

Cache invalidation level

Tracks of records/repositories usages is used by the cache invalidation logic. It allows for configure-less cache invalidation (say goodbye to TCEMAIN.clearCacheCmd). Instead, the administrator has to concentrate on setting the proper cache invalidation level.

There are two levels currently implemented:

- consistent – This is the default value. The cache will be invalidated as soon as the record change is performed.
- noinvalidation – This completely disables the cache invalidation performed by extbase_hijax. Might be useful for high traffic websites where cache invalidation is done by some scheduler task.

The setting can be changed via TypoScript:

```
config.tx_extbase {
	settings {
		cacheInvalidationLevel = consistent
	}
}
```

Please note that there will be new levels implemented in future which will target mid-traffic websites. However, for most of the TYPO3 websites it is recommended to keep the default "consistent" level.

Cache invalidation

No matter what the cache invalidation level is set by the administrator, the developers should implement proper cache invalidation in their extensions. If you **update** a record, you have to explicitly invalidate cache for all related pages by calling:

```
$this->objectManager->get('Tx_ExtbaseHijax_Tracking_Manager')->clearPageCacheForObject($poll);
```

If you **add** or **delete** a record you need to invalidate cache for the whole repository. Sample code:

```
$this->trackingManager = $this->objectManager->get('Tx_ExtbaseHijax_Tracking_Manager');
...
$objectIdentifier = $this->trackingManager->getObjectIdentifierForRepository($table, $pid);
$this->trackingManager->clearPageCacheForObjectIdentifier($objectIdentifier);
```

You would need to invalidate cache for a repository in the event of updating an object only if that change can cause the list view to get change (for example hiding a record, or changing archive date...). Invalidating a cache for an object won't automatically invalidate the cache for its repository!

Automatic cache invalidation in the BE

If you have properly implemented tracking of records and repositories in your FE plugin, the BE changes will automatically fire cache invalidation for the related pages. Meaning you don't have to use TCEMAIN.clearCacheCmd to configure automatic FE cache clearing (at least not for your extensions :D).

Warning

We are currently evaluating whether it will be good (and how we can implement at all) automatic tracking of records/repositories and cache invalidation calls. You are again warned to stick to a tag when basing your extensions on extbase_hijax as bunch of stuff might change.

Known problems

- Nested event listeners might fire unneeded amount of calls (one for each listener). If a listener and its child are listening to the same event which got fired both will get revived and rerun. The child listener of course won't be displayed, as the parent might render it again... As these event listeners should be used mostly for rendering, the only flaw is that some small portions of code will be run without any need. This will be optimized in future.
- Ajaxifyed form will by default load the results in place of the form itself, not the whole plugin output. As noted before, this behavior will change. For now, you should probably just render forms without surrounding markup in your extensions, or use resultTarget attribute. You can of course put layout and bunch of markup in the form itself :)

To-Do list

- Fix known issues,
- Data-binding Fluid viewhelper (for all of you who worked with Adobe Flex, you probably already know what you can expect: MXML's {object.property} will be possible in Fluid with even more convenient features :)),
- Automatic object/repository usage tracking,
- Fully automatic cache invalidation in the FE,
- Allow customizing of cache invalidation levels/algorithms,
- Add support for hijaxing all f:link viewhelpers,
- Add support for hijaxing HMENU TypoScript objects,
- Implement cache management for per-plugin-instance elements (for usage in AJAX dispatcher),
- Implement scheduler task for queuing cache repopulation.

ChangeLog

Version	Changes:
0.1.4	Added documentation of the currently implemented features