TYPO3.Flow - Bug # 45290

Work Package # 45088 (Resolved): Improved REST support

Status:	Closed	**Priority:**	Should have
Author:	Bastian Waidelich	**Category:**	Http
Created:	2013-02-08	**Assigned To:**	Bastian Waidelich
Updated:	2013-09-23	**Due date:**	2013-04-13
PHP Version:			
Has patch:	No		
Complexity:			
Affected Flow version:	Git master		
Subject:	Body arguments should not be merged before property mapping takes place		

Description

Currently **body** and **POST** arguments are merged with **GET** arguments in Http\Request::__construct().
Instead the merging should happen after routing took place in order to know more about the affected arguments.

Now, if you invoke a PUT request to http://localhost/products/product-1 with a body of

```
1{
2  "product": {
3    "title": "new title"
4  }
5}
```

GET arguments:

```
1array('product' => '<uuid of product-1>')
```

Body arguments:

```
1array('product' => array('title' => 'new title'));
```

Obviously \Http\Request::buildUnifiedArguments() can't properly merge those and - as a result - ignores the body arguments.

Instead the property mapper should probably fetch body arguments separately before calling the action (probably in ActionController::mapRequestArgumentsToControllerArguments())

Related issues:

related to TYPO3.Flow - Task # 44712: Decouple Argument-Building in the HTTP-...	**Accepted**	**2013-01-22**

History

#1 - 2013-02-22 10:35 - Bastian Waidelich

- Parent task set to #45088

#2 - 2013-03-28 16:21 - Aske Ertmann

- Parent task deleted (#45088)

#3 - 2013-03-28 16:22 - Aske Ertmann

- Parent task set to #45088

#4 - 2013-04-02 13:17 - Bastian Waidelich

- Due date set to 2013-04-13

#5 - 2013-04-04 15:45 - Gerrit Code Review

- Status changed from New to Under Review

Patch set 1 for branch **master** has been pushed to the review server.
It is available at https://review.typo3.org/19556

#6 - 2013-04-08 12:36 - Gerrit Code Review

Patch set 2 for branch **master** has been pushed to the review server.
It is available at https://review.typo3.org/19556

#7 - 2013-04-08 12:37 - Bastian Waidelich

Here is a little test package that demonstrates the "bug": https://github.com/bwaidelich/Wwwision.ArgumentTest

#8 - 2013-04-24 10:55 - Bastian Waidelich

- % Done changed from 0 to 30

This one is a bit tricky, especially if we want to resolve it in a API-backwards-compatible (aka BC) way.
I wrote several spikes and prototypes but I always either broke the API or resulted with a overcomplex architecture..
Here are my findings:

Current request handling

1. The **RequestHandler** builds the **HttpRequest**[1] calling ***Request::createFromEnvironment()***
2. The **HttpRequest** merges **$_GET**, **$_POST**, **$_FILES** with the **bodyArguments**[2] and unifying them into **arguments**
3. The **Router** calls ***HttpRequest::createActionRequest()*** and passes all **arguments** to the created **ActionRequest**
4. After a route was found the **Router** then merges the arguments of the **ActionRequest** with the **$matchResults**[3]
5. The **AbstractController** maps ActionRequest arguments to the parameters of the action to be called (in ***AbstractController::mapRequestArgumentsToControllerArguments()***)

Problems

Here are a couple of "issues" we have with the current sequence:

- The **HttpRequest** always merges all arguments and parses the request body in its constructor even if they might not be needed (this can cost performance & memory)
- The parsing of bodyArguments is hard-coded and only supports basic XML, JSON & urlencoded strings.The **HttpRequest** is built very early (e.g. before DI is available), so we can't easily inject a service that could take over that job. Currently, if you use your custom mediatype (which is encouraged in REST services for example), you'll have to parse the request body yourself somehow and in the worst case the HttpRequest trips over your format when it tries to parse it in ***Request::decodeBodyArguments()***
- Currently there is no way to separately retrieve GET/POST and **bodyArguments** from the HttpRequest, resulting in the very problem described in

this ticket

Long term

Christopher came up with the idea to refactor the request handling into some kind of **chain**: Each component of the chain would take a **Request** and **Response** and could apply filters and such on them (visitor pattern).
This would also greatly decouple our MVC architecture from the application (you could define custom chains that end up somewhere else than in an ActionRequest, and you could replace single components – like the Router – with a simpler/better suited implementation for certain requests)

While I really like Christophers idea, I can't exactly imagine how that would work out in practice and whether it would kill the BC cow.. In any case we probably need an intermediate step really soon. So that would be my suggestion:

Short term

I tried to come up with some steps that would solve most of our current issues while keeping the API **kind of** backwards-compatible:

- In general I think the HttpRequest should be less smart and map "1:1" to what exists in the real HTTP request
- Keep ***Request::getArguments()*** (which is public API) but let it only return merged **_GET** and **_POST** arguments, not the parsed **bodyArguments** (I'm not sure about **_FILES**, they can be merged too I guess).
  - In addition the HttpRequest could store **_GET** and **_POST** separately so that the merging only needs to be done when needed (the first time) and that **_GET** and **_POST** variables can be accessed separately if needed
- ***HttpRequest::createActionRequest()*** only passes itself to the constructor of **ActionRequest** (this is already the case) and does not copy it's arguments to the **ActionRequest**
- The router passes the **$matchResults** to the **ActionRequest** (no merging, the ActionRequest only contains the routing values)
  - Maybe it would make sense to rename ***ActionRequest::arguments*** keeping the old getters/setters as deprecated for a while) to something that distinguish them from the HTTP arguments.
  - We could keep ***ActionRequest::getArguments()*** BC by returning a merged result of $this->arguments && $this->httpRequest->getArguments()
- The **ActionController** would do the heavy work and merge **required** arguments in ***AbstractController::mapRequestArgumentsToControllerArguments()***
  - This is also the time where the **bodyArguments** are parsed (preferably by some extensible service)

Breaking changes

The above path would result in a couple of breaking changes, but only for rare cases as far as I can see:

- arguments returned from ***Request::getArguments()*** would **not** contain the **bodyArguments** (you could use the above mentioned service to parse those)
- If you override ***AbstractController::mapRequestArgumentsToControllerArguments()*** you probably have to adjust your implementation slightly

Footnotes

[1] The class of the HTTP request is actually called *Request* – I call it *HttpRequest* here to avoid confusion with the *ActionRequest*

[2] With **bodyArguments** we refer to everything that is part of the request body and has not been mapped to _POST or _FILES by PHP – e.g. the payload in a REST request.

[3]**matchResults** are the results of a matched route which is an array of all **defaults** and the **uriPattern** values of the route

#9 - 2013-04-29 03:09 - Alexander Berl

Hi Bastian,

I like the general idea and here is some input from my side:

- Why does the **HttpRequest** need to know how to create an **ActionRequest**? This is IMO *bad coupling* and the creation of an **ActionRequest** should be moved to the **Router**.
- The **Router** is already capable of full DI, so it would be easy to inject argument mapping services in there, which uses the routing matches and **HttpRequest** arguments + body as sources
- The **ActionRequest** in turn can then just get the raw arguments injected
- Rest can stay as is, ie. **AbstractController:mapRequestArgumentsToControllerArguments()** would stay unchanged and really only do what it says - map raw request arguments to full blown Controller arguments (objects), without caring where they come from

My point is that it would be even *more BC* as really only **HttpRequest***becomes a little bit dumber*, it *keeps the concerns better seperated* (decouple **Http** from **Mvc**, keep argument merging in **Routing**), plus it is quite handy to have a central place outside the current Controller where you can just

access all raw action arguments without having to care if they came from routing, GET/POST/FILES, request body or whatever. A good example is creating caching identifiers from the arguments, where you don't want to deal with **Entity Objects** and the likes. The **ActionRequest** is IMO the best point to have those available.

#10 - 2013-05-06 12:09 - Bastian Waidelich

Alexander Berl wrote:

Hi Alexander,

thanks for your feedback, it's appreciated!

> - *Why does the **HttpRequest** need to know how to create an **ActionRequest**? This is IMO bad coupling* and the creation of an **ActionRequest** should be moved to the **Router**.

Correct! We already agreed to remove **HttpRequest::createActionRequest()** – luckily that's not part of the public API

> - *The **Router** is already capable of full DI, so it would be easy to inject argument mapping services in there, which uses the routing matches and **HttpRequest** arguments + body as sources*

Mh. I'm not sure that it's the routers job to do extended argument mapping but I'll give it a try

> - *The **ActionRequest** in turn can then just get the raw arguments injected*

Right. It could also fetch the arguments from the HttpRequest when **accessed the first time**.

> - *Rest can stay as is, ie. **AbstractController:mapRequestArgumentsToControllerArguments()** would stay unchanged and really only do what it says - map*
> *raw request arguments to full blown Controller arguments (objects), without caring where they come from*

The thing is: The only way (I can think of) to solve the problem described in this issue is to check the type of an action parameter in order to turn '<UUID>' into array('__identifier' => '<UUID>'). **AbstractController:mapRequestArgumentsToControllerArguments()** is currently the place where this could be done. But I agree that this doesn't feel right as it bloats the controller even more.

> *My point is that it would be even more BC* as really only **HttpRequest***becomes a little bit dumber...*

Exactly my point.

#11 - 2013-05-08 14:01 - Alexander Berl

Ok, I think I get it now. The problem is that you also need the Controller Arguments information to know that a parameter is supposed to be an entity, since the routing might not contain that information (*like in the example app where the route part is only a plain dynamic one*).

The easy solution would be to use the IdentityRoutePart for this parameter and supply the objectType explicitly.

Maybe the problem can be solved differently: Basically, only the information from the controller parameter types is missing in the routing configuration - now that information is normally static for as long as the application code doesn't change (see ActionController::getActionMethodParameters, it is even

statically compiled). What if some kind of *MvcControllerArgumentMappingService* would collect all parameter type information during compile time and add this information to the routing configuration for all parameters that are missing the objectType configuration?

Only thing that I see that would not work are dynamic controller arguments, but those are normally no entities as the type information is missing unless you do $this->arguments->addNewArgument manually somewhere in your controller.

#12 - 2013-05-21 13:28 - Robert Lemke

- Target version deleted (2.1)

#13 - 2013-09-19 16:02 - Bastian Waidelich

- % Done changed from 30 to 100

Will be resolved with https://review.typo3.org/21134/

#14 - 2013-09-23 14:01 - Bastian Waidelich

- Status changed from Under Review to Closed

See #45293