TYPO3.Fluid - Feature # 48355

Status:	New	**Priority:**	Could have
Author:	Dave no-lastname-given	**Category:**	
Created:	2013-05-17	**Assigned To:**	
Updated:	2013-07-06	**Due date:**	
Has patch:			
Subject:	Assign output of viewhelper to template variable for further processing.		

Description

It would be nice if it was possible to "assign" the output of a viewhelper to a template variable with an assign attribute for usage elsewhere in the template. When assign is set the output of the viewhelper is suppressed.

For example the template:

```
<f:format.date format="d.m.Y - H:i:s" assign="myDate">\@{your_timestamp}</f:format.date>
<p>Here is my date: {myDate}</p>
<p>Here is again my date: {myDate}</p>
```

Would output:

```
Here is my date: 1.1.2013 - 12:00:00
Here is again my date: 1.1.2013 - 12:00:00
```

I know <f:alias> exists but the kind of assignment I am suggesting would by faster to parse by not having to call the viewhelper multiple times to output the same value.

The "assign" should be a universal attribute to all Viewhelpers.

Then you can do things also like this:

```
<f:for each="{myCollection}" as="collectionItem" assign="myList">
  <f:render partial="Some/Item"  />
</f:for>

<div class="top list">{myList}</div>
<div class="content">{myOtherContent}</div>
<div class="bottom list">{myList}</div>
```

The advantages I think is clear when applied to looped elements. You don't have to process the same content twice or more.

History

#1 - 2013-05-17 18:46 - Philipp Gampe

This would need to go to upstream first:
http://forge.typo3.org/projects/package-typo3-fluid

#2 - 2013-05-17 23:11 - Dave no-lastname-given

- Project changed from Fluid to TYPO3.Fluid
- Category deleted (Fluid: Core)

Just upstreaming this issue from Typo3_CMS->Core->Fluid

#3 - 2013-05-19 14:28 - Alexander Berl

I don't quite get what would be the benefit over the f:alias viewhelper:

```
<f:alias map="{myDate: '{f:format.date(subject: \'\@{your_timestamp}\', format: \'d.m.Y - H:i:s\')}'}">
  <p>Here is my date: {myDate}</p>
  <p>Here is again my date: {myDate}</p>
</f:alias>
```

Depending on the content of the f:alias viewHelper it would anyway be recommended to use a partial instead - see the following.
Also, I'm not sure what your partial rendering example is supposed to achieve:

```
<f:for each="{myCollection}" as="collectionItem" assign="myList">
  <f:render partial="Some/Item"  />
</f:for>
```

This would not work, as you need to assign variables to the partial explicitly via the "arguments" attribute, or else the variable scope inside the partial would be empty.

However, the assign would do nothing of benefit in this case, as the following would achieve the same:

```
<f:for each="{myCollection}" as="collectionItem">
  <f:render partial="Some/Item" arguments="{myList: collectionItem}" />
</f:for>
```

Please provide more information on what is supposed to be achieved and what the use cases are.

#4 - 2013-05-20 10:19 - Dave no-lastname-given

In my mind the benefits are clear.

- Cleaner, simpler template markup.

- Faster to write templates.
- Less processing for the parser.

Alias is clumsy, and time consuming.
Inline templating is difficult to write and read when it gets complex.

It is not at first glance obvious what a map is doing when written inline.
Multiple mappings of different variables in the same alias complicate it even further.
It is not user friendly for someone trying to figure out someone elses code.

Trying mapping a complex "for loop" containing multiple if conditions in a alias map.

Off the top of my head, a use case could be with adding a paginator in multiparts of the same template.
That is a classic for-loop scenario. But there are countless uses for repeating a variable multiple times.

Alexander Berl wrote:

> *However, the assign would do nothing of benefit in this case, as the following would achieve the same:*

Actually you have edited my example snippet to push your argument
Maybe you have really mis-read the code, or have a misunderstanding of what it means to assign the output of a viewhelper to a template variable.

In this example I am not concerned in pushing "myList" into the "Some/Item" partial.
My intention is to repeat the processing of the for-loop without having to process it again.

The original snippet was this (see parent post to compare):

```
<f:for each="{myCollection}" as="collectionItem" assign="myList">
  <f:render partial="Some/Item"  />
</f:for>

<div class="top list">{myList}</div>
<div class="content">{myOtherContent}</div>
<div class="bottom list">{myList}</div>
```

If I was to write that snippet to achieve the same result in Fluid today it would look like this:

```
<div class="top list">
  <f:for each="{myCollection}" as="collectionItem">
    <f:render partial="Some/Item"  />
  </f:for>
</div>
<div class="content">{myOtherContent}</div>
<div class="bottom list">
  <f:for each="{myCollection}" as="collectionItem">
    <f:render partial="Some/Item"  />
  </f:for>
```

```
</div>
```

It increased the template code by 5 lines, and calls the "for" and "render" viewhelpers twice, when it should only be neccesary to process it once. You are making the server work harder than necessary.

Also assigning template variables in such a way has been used in Smarty templating, somebody coming from Smarty would be familiar with the concept. The concept of assigning variables in templates in such a way has been in Smarty for a long time. Fluid even uses the word "assign" for the template variable "setting" method in viewhelpers. So assigning template variables in such a way has a historical and even sub-conscience precedence. I think doing it the same way has it's benefits for somebody new to Fluid who has experience in Smarty.

In fact the logical step of this would be to additionally create a assign viewhelper.

```
<f:assign name="myVar" value="Hello World!" />
```

It would not require a closing tag block to be used like alias.

I am not suggesting something new, just something powerfully convenient for templating that has proven production value in other templating systems.

#5 - 2013-05-20 11:28 - Dave no-lastname-given

BTW if you also intend to add a assign viewhelper, please make sure it has a **cast** attribute with type "string" being default:

Examples:

```
<f:assign name="myName" value="Dave" />
<f:assign name="myVar" value="true" />
<f:assign name="myVar2" value="true" cast="boolean" />
<f:assign name="myVar3" value="1.5" cast="float" />
<f:assign name="myVar4" value="cat,dog,mouse" cast="array" />
<f:assign name="myVar5" value="{myClassInitParams}" cast="\My\Cool\CustomClass" />

{myName}<br />
{myVar}<br />
{myVar2}<br />
{myVar3}<br />
{myVar4}<br />
{myVar5->publicMethodDoSomethingAwesome({myName})}
```

This would output:

```
Dave
true
1
```

```
1.5
Array()
Dave was 'ere!
```

Assigning variables is **primitive** in templating. It is all about that.
This would have been the first viewhelper I would have written when creating a new templating system.
I am quite amazed you don't have it this far in.

If fluid could do this it would be awesome. It is not too late.

As a second thought imagine this, "dynamic" ViewHelpers.
Just for the abstraction value:

```
<f:assign name="myViewHelper" value="{myClassInitParams}" cast="\TYPO3\CMS\Fluid\Core\ViewHelper\ForViewHelper" />
<f:myViewHelper each="{foo} as="bar">{bar}</f:myViewHelper>
```

That would be in my book really damn amazing if that was possible.

#6 - 2013-05-20 12:40 - Dave no-lastname-given

Sorry just thinking about possibilities....
You could even cast variables as native PHP functions.

```
<f:assign name="microtime" cast="\PHP\microtime" />
<f:microtime />

<f:assign name="upperCase" cast="\PHP\strtoupper" />
<f:upperCase>lower case text</f:upperCase>
```

The power it would bring to Fluid is the entire PHP API at your fingertips if you need it.
If some viewhelper functionality is missing in the Fluid Core or extension it shouldn't be a problem to instantly solve the problem using a assigned variable with a cast.

IMHO That would make Fluid incredibly powerful.
Better than Smarty.

Anyway that is my 2c..

#7 - 2013-07-04 21:24 - Alexander Berl

I see your point now. What I don't like about it, is that the assigned variable will be "global" scope, which might cause undesired variable name clashes and just contradicts the whole Fluid philosophy of scoped contexts. Also I'm still unsure if it brings that much benefit to assign rendered content to a

variable instead of just wrapping it in a(nother) partial which will get cached. I guess that would need some benchmarking, but in the end this could easily be achieved by a custom viewhelper as follows:

```
class AssignViewHelper extends AbstractViewHelper {
   ...
   public function render($name, $value = NULL) {
      if ($value === NULL) {
         $value = $this->renderChildren();
      }
      $this->viewHelperVariableContainer->add($name, $value);
   }
}
```

You might want to add checks for existence of a variable with that name though and handle the conflict case appropriately plus add your casting option. I doubt something like this will go into Fluid though, but that's not my decision, so I'll leave further discussion up to the core team.

#8 - 2013-07-06 18:05 - Dave no-lastname-given

Alexander Berl wrote:

> *I see your point now. What I don't like about it, is that the assigned variable will be "global" scope, which might cause undesired variable name clashes and just contradicts the whole Fluid philosophy of scoped contexts.*

I am sure if you want your variables scoped you can just add a scope attribute to the viewhelper abstract.
That is a small issue that can be easily fixed with a little thought. Though undesired variable name clashes is a human construct problem and not something you should massively worry about. If I use common reserved keywords for my variable names I should understand the potential pitfalls and consequences, this is common sense in all mark-up/languages. To explain this best I like to use the "douche" example: If I take a douche (http://en.wikipedia.org/wiki/Douche) in the UK, it is not the same thing as if I take a douche in France or Germany. You only make that mistake once. :-)

BTW is there a Fluid philosophy written somewhere I can read? I haven't found one.

This is my philosophy: Fluid should be fluid.
It should flow off my fingers like water off a ducks back, at the moment it is very convoluted and constrained in functionality. I always have to dig around in viewhelper classes to see if something has been implemented (in most cases generally not and in some cases duplicated under different names). This breaks the flow of template writing. With every extension I write or modify, I don't want to write custom viewhelper classes to replicate some native PHP functionality that is not declared in a core viewhelper file. I tell you.. my duck keeps drowning everytime I do this. Poor ducky. :-(

That IMHO is NOT a fluid process to template writing. It is just plain silly. It is like reinventing the wheel over and over.

By assigning variables and implementing casting to access the PHP API directly, it would eliminate most of the core viewhelpers that need to be written. It would save the project a lot of time and effort, as all the fuctionality already exists in native PHP. The only time you should need to write a custom viewhelper class is when you are dealing with some complex business logic that requires more than a few single worded PHP function calls to get the desired output. Basically anything that PHP doesn't do in a one liner.

BTW PHP is a "Hypertext Preprocessor" which are just fancy words for templating language.
That makes Fluid a templating language coded in a templating language.

Hold that thought for a second. Whoah dude! What are we smoking? That is deep. ;-)

So maybe we should keep to the PHP metal as close as possible just for the sake of performance and to keep our options open.

We could even have a native PHP fallback. If I write <f:strtoupper>foo</f:strtoupper> in my template and the viewhelper class does not exist, check if a native PHP function exists with the same name and if it does use that as the preprocessor. Surely this would not take so much to implement as compared to writing multitudes of redundant viewhelpers.

At the moment Fluid has a kind of very chunky custard consistency about it and it is clogging the drains.
I hope to see Fluid become more "fluid", assigning variables and "in-template" casting would make it flow better.

my 2c.
BTW I hope the core team reads this. Think of the ducks. :-)