TYPO3 Flow Base Distribution - Story # 12862

Status:	Resolved	**Priority:**	Should have
Author:	Sebastian Kurfuerst	**Category:**	
Created:	2011-02-08	**Assigned To:**	Sebastian Kurfuerst
Updated:	2011-05-06	**Due date:**	
Subject:	As Roger, I want a clean Property Mapper		

Description

{{>toc}}

Goal

The property mapper maps simple types to Objects. (Hence, we (for now) drop the Object to Object support, although this could be added later)

The Property Mapper maps all values present in $source to $target (or a subset thereof) (right now, it requires a list of properties to be mapped, and if $source[$propertyName] does not exist, an error is thrown)

- We try to strip down the Property Mapper, and make it extensible instead.

Wanted Features

- Support for Object Converters:
  - array / ArrayObject / SplObjectStorage
  - DateTime use case
  - File Upload (Resource objects)
- Partial Validation should be supported
- Property Merging should be supported (pass in existing object as base-object)
- Better error handling in case target property is not found (right now, NULL returned)
- Mapping should be configurable
- Secure by default

Basic Interface

```
1map(mixed $source, string $targetType, PropertyMappingConfigurationInterface $propertyMappingConfiguration = NULL)
```

- $source can be:
  - all simple values (integer, string, ...)
  - ARRAY
  - an Object implementing ArrayAccess
  - **We will drop support for arbitary objects through ObjectAccess::isPropertyGettable** (for now)
  - else EXCEPTION
- $target is the classname on which the value should be mapped to (or a simple type)

Pseudocode:

```
1map(...) {
2   if ($propertyMappingConfiguration === NULL) {
3       $propertyMappingConfiguration = $this->buildDefaultPropertyMappingConfiguration();
4   }
5   $propertyMappingConfiguration->setTargetType($targetType); // Just a convenience shorthand method
6   $this->doMapping(....);
7}
8
9protected function doMapping(mixed $source, $propertyMappingConfiguration) {
10   $typeConverter = $this->somehowDetermineTypeConverter(); // TODO, see below
11
12   $object = $typeConverter->convertFrom($source, $propertyMappingConfiguration);
13
14   foreach ($typeConverter->getPropertyNames($source) as $sourcePropertyName) {
```

```
15        $targetPropertyName = $propertyMappingConfiguration->getTargetPropertyName($sourcePropertyName);
16        if (!$propertyMappingConfiguration->shouldMap($targetPropertyName)) continue;
17
18        $mappedPropertyValue = $this->doMapping($source[$sourcePropertyName], $propertyMappingConfiguration
->getConfigurationFor($targetPropertyName));
19
20        $typeConverter->setResultInTarget($target, $targetPropertyName, $mappedPropertyValue);
21    }
22    return $object;
23}
```

- The property mapper only **delegates** the actual mapping to **Type Converters**.
- Type converters expose the following information:
  - Applicable source types
  - The target type the converter can convert to
  - A priority
- The used type converter is determined as follows:
  - If a TypeConverter is set inside the current propertyMappingConfiguration, this one is taken.
  - Else, we do the following:

```
1$typeConvertersWhichCanHandleSourceType = // first, we pick all TypeConverters
2        // which can handle $sourceType, and put it into an
3        // associative array where KEY is the target class name of the converter
4$classNamesInInheritanceHierarchy = // $targetType combined with all superclasses
5        // and interfaces, ordered from the inside out (so the $targetType
6        // is the first element in the list)
7
8foreach ($classNamesInInheritanceHierarchy as $singleTargetClassName) {
9    if (isset($typeConvertersWhichCanHandleSourceType[$singleTargetClassName])) {
10        return the type converter from $typeConvertersWhichCanHandleSourceType[$singleTargetClassName] with highest
priority
11    }
12}
13// Exception if no type converter found
```

- Type Converters have to adhere to the following interface:

```
1public function convertFrom($source, $propertyMappingConfiguration);
2public function getPropertyNames($source);
3public function setMappingResultInTarget($target, $propertyName, $mappingResult);
```

- In case a type converter does **not** want to handle a certain element (for whatever reason), he can throw an DontWantToHandleThisInputException inside convertFrom. Then, the next Type converter according to the above rules is taken.

- Furthermore, the Property Mapper **can handle renaming of properties**, see the method getTargetPropertyName in the PropertyMappingConfigurationInterface

PropertyMappingConfiguration

```
1interface PropertyMappingConfigurationInterface {
2
3    /**
4     * @return string
5     */
6    public function getTargetType();
7
8    /**
9     * @return TRUE if the given propertyName should be mapped
10     */
11    public function shouldMap($propertyName);
12
13    /**
14     * @return PropertyMappingConfigurationInterface the property mapping configuration for the given PropertyName.
```

```
    */
    public function getConfigurationFor($propertyName);

    /**
     * @return string property name of target; can be used to rename properties from source to target.
     */
    public function getTargetPropertyName($sourcePropertyName);

    /**
     * @return mixed configuration value for the specific $typeConverterClassName. Can be used by Type Converters to fetch converter-specific configuration
     */
    public function getConfigurationValue($typeConverterClassName, $key);
}
```

This interface will have an implementing class "PropertyMappingConfiguration", looking like the following:

```
class PropertyMappingConfiguration implements PropertyMappingConfigurationInterface {
    public function setTargetType($type);
    public function setMapping($sourcePropertyName, $targetPropertyName);
    public function setConfigurationValue($typeConverter, $key, $value);

    /**
     * Returns the default configuration for all sub-objects, ready to be modified
     */
    public function defaultSubConfiguration();

    /**
     * Returns the configuration for the specific property path, ready to be modified
     */
    public function at($propertyPath);

}
```

This API can then be used as follows:

- To globally disable the modification of entities by the property mapper:
  ```
  $propertyMappingConfiguration->setConfigurationValue('F3....\EntityConverter', 'modificationAllowed', FALSE);
  ```

- To disable the modification of sub-entities of the root element:
  ```
  $propertyMappingConfiguration->defaultSubConfiguration()->setConfigurationValue('F3....\EntityConverter', 'modificationAllowed', FALSE);
  ```

- To enable the creation of sub-objects inside a certain element:
  ```
  $propertyMappingConfiguration->at('subobject1.subsubobject2')->setConfigurationValue('F3....\EntityConverter', 'creationAllowed', TRUE);
  ```

Furthermore, there will be a subclass ControllerPropertyMappingConfiguration, which has some convenience methods implemented such that people can program against a type-safe API:

```
class ControllerPropertyMappingConfiguration extends PropertyMappingConfiguration {
    public function allowCreationOfSubObjectsAt($position) {
        $this->at($position)->setConfigurationValue('F3....\EntityConverter', 'creationAllowed', TRUE);
    }
    public function allowModificationOfSubObjectsAt($position) {
        $this->at($position)->setConfigurationValue('F3....\EntityConverter', 'modificationAllowed', TRUE);
    }
}
```

Default Property Mapping Configuration

- The default property mapping configuration for the **top level** should configure the Entity Type Converter to **modify** and **create new** objects
- The property mapping configuration further down the hierarchy should configure the Entity Type Converter to **not modify** and **not**

create objects.
- The above two things are safety precautions for the user.
- This boils down to the following code:

```
1 $propertyMappingConfiguration->setConfigurationValue('F3....\EntityConverter', 'modificationAllowed', TRUE);
2 $propertyMappingConfiguration->defaultSubConfiguration()->setConfigurationValue('F3....\EntityConverter', '
modificationAllowed', FALSE);
```

Type Converter examples

Here follow some examples of type converters, to see the concept more easily:

Entity Type Converter

- **Input Type**: String and Array
- **Output Type**: Object
- **Priority**: low

```
1public function convertFrom($source, $propertyMappingConfiguration) {
2    if (is_string($source)) {
3        if (/* is UUID */) {
4            // create and return object
5        } else {
6            // Exception
7        }
8    }
9
10   if (isset($source['__identity'])) {
11       $target = // fetch from persistence
12       if (count($source) > 1) {
13           if ($propertyMappingConfiguration->getConfigurationValue('F3\....\EntityConverter', 'modificationAllowed') !== TRUE)
{
14               // throw exception
15           }
16           $target = clone $target;
17       }
18   } else {
19       if ($propertyMappingConfiguration->getConfigurationValue('F3\....\EntityConverter', 'creationAllowed') !== TRUE) {
20           // throw exception
21       }
22       $target = // new
23   }
24
25   return $target;
26}
27
28public function getPropertyNames($source) {
29       if (is_string($source)) return array();
30       return all $source property names, *except* __identity;
31}
32
33public function setMappingResultInTarget($target, $propertyName, $mappingResult) {
34       // Set Property through Object Access
35}
```

Date Type Converter

- **Input Type**: String and Array
- **Output Type**: DateTime
- **Priority**: low

```
1public function convertFrom($source, $propertyMappingConfiguration) {
2    new DateTime($source); // or more advanced...
3};
4public function getPropertyNames() {
5    return array();
6}
```

```
7public function setMappingResultInTarget($target, $propertyName, $mappingResult) {
8    // empty, as it is never called
9}
```

SplObjectStorage/ArrayCollection Type Converter

- **Input Type**: Array
- **Output Type**: SplObjectStorage
- **Priority**: low

```
1public function convertFrom($source, $propertyMappingConfiguration) {
2    return new \SplObjectStorage();
3};
4public function getPropertyNames($source) {
5    return array_keys($source);
6}
7
8public function setMappingResultInTarget($target, $propertyName, $mappingResult) {
9    $target->attach($value);
10}
```

Error Handling

- Instead of outputting human-readable error messages, the property mapper will throw **exceptions** in case of errors. For things like mandatory properties, specific validators should be used.

Validation / Partial Validation

- The property mapper will **not** execute any validators for the given objects.
- However, it should return a list of mapped properties. This list can be used later, such that the ObjectValidator only needs to validate changed properties.

Naming

Object Converters will be superseded by this concept, and thrown away.

We suggest to change the naming in the following way:

- PropertyMapper -> TypeMapper / ObjectMapper?
- Object Converter -> Type Converter

Further work

- This concept could be extended lateron to work with **arbitrary** input objects, not just simple types. However, one would then need a way to deal with object inheritance hierarchies on the source side as well.
- Extend the concept such that Half-ready objects as $target are also supported; but we would use an explicit new API function for this:

```
1merge(mixed $source, object $targetObject, PropertyMappingConfigurationInterface $propertyMappingConfiguration = NULL)
```

Subtasks:

Task # 13943: Restructured MVC Error Handling	**Resolved**
Task # 13942: Restructured Property Mapper	**Resolved**
Task # 13161: We also need some better Error / Warning / Notice handling	**Resolved**

History

#1 - 2011-02-08 09:11 - Sebastian Kurfuerst

- Position deleted (1)

- Position set to 15

#2 - 2011-02-08 10:07 - Sebastian Kurfuerst

- Assigned To set to Sebastian Kurfuerst

#3 - 2011-02-08 21:12 - Peter Niederlag

Date Type Converter

- As Mr. Glue I want it to call the constructor of my Extension class if I have a property that has a type that extends DateTime()

Float Type Converter

- As Mr. Glue I want to influence what format the input must have

#4 - 2011-02-10 12:45 - Sebastian Kurfuerst

Hey Peter,

thanks for your input; both cases would be possible with the new concept:

> ### *Date Type Converter*
>
> - *As Mr. Glue I want it to call the constructor of my Extension class if I have a property that has a type that extends DateTime()*

This would be done differently with the new concept: You'd create your custom TypeConverter, which can convert to your extended DateTime object. This converter is then automatically used.

> ### *Float Type Converter*
>
> - *As Mr. Glue I want to influence what format the input must have*

A Float Type Converter could be configurable through the PropertyMappingConfiguration, so this should also be possible.

#5 - 2011-02-10 14:36 - Sebastian Kurfuerst

I just updated the concept after discussions with Andi and Christian.

#6 - 2011-02-14 11:04 - Karsten Dambekalns

Some comments: - seeing object-to-object go makes me uneasy, even if it's only "for now".

- map() without a target object is actually convert()
  - make map() accept a target object, allow NULL to indicate "convert" mode as a special case
  - or call it convert() and add merge() as suggested
- the pseudo code to determine the matching type converter ignores the priority
- there seems to be no way to configure the mapping in a way like "allow mapping for all properties except ..."
- what does PropertyMappingConfiguration::setMapping() do?
- setMappingResultInTarget() would use a domain model instance as a transfer object for infrastructure metadata...
  - no, it is actually badly named and would put the conversion result in the new object, aha...
  - the SplObjectStorage-example is bad then, because it only ever attaches one object.

On the naming of PropertyMapper - if anything, it now is an ObjectCreator, eyh? If merge() is added, it suddenly becomes a TypeConverter though, thus we have a name clash already. In the end, though, that's what it is: it converts stuff from array to SomeObject according to some configuration...

#7 - 2011-02-22 14:53 - Bastian Waidelich

- Improve error messages
- unschön: dass man für new/edit action auch eine initialize*Action braucht
- Idee mit Robert diskutiert:
  - Parameter bei "newAction" weglassen (können)
  - Momentaner Ablauf
    Validierungsfehler / Property Mapping Fehler
    errorAction
      - > packt argument errors in request
      - > forward zur letzten Action, ABER mit Argumenten der aktuellen Action (HACK) -- dabei bleiben Errors erhalten (HACK!)
    letzte Action (die das Formular anzeigt) aus referrer
      - > Argumente werden **nochmal** gemappt, und "invalides" Domänen-Objekt gebaut
      - > ABER dontvalidate verhindert, dass Property Validatoren gebaut werden (UNSCHÖN, greift nicht für Property Mapping errors)
      - > "invalides" domänenobjekt geht zum View -- > Basis für Formular
      - > Form ViewHelper geht in **REQUEST**- >getErrors(), um herauszufinden dass es Fehler gab (HACK)
  - Vorgeschlagener Ablauf
    Validierungsfehler / Property Mapping Fehler
    errorAction
      - > Fehler die es gab müssen **irgendwo** gespeichert werden (Error\Result) (im AbstractRequest); außerdem muss der "Original"-Request gespeichert werden
      - > forward zur letzten Action, **mit Argumenten der letzten Action** (die auch im Referrer gespeichert sein müssen)
    letzte Action (die das Formular anzeigt) aus referrer
      - > Genau gleicher Kontrollfluss wie im "nicht-fehler-Fall"
      - > View wird "normal" aufgerufen
      - > Form ViewHelper nutzt Request->originalRequest->arguments zum Vor-Ausfüllen des Formulares (mit Fallback zum übergebenen Objekt)
      - > Form ViewHelper nutzt Request->originalRequest->errors um herauszufinden, welche Formularfelder fehlerhaft sind (und kann die dann umrahmen / Fehler darstellen, ...)

/// VORALLEM WICHTIG FÜR EXTBASE:
1protectedfunctionmapRequestArgumentsToControllerArguments() { 2foreach ($this->arguments as$argument) { 3$argumentName = $argument ->getName(); 4 5// in EDITACTION request -- "originale" Request der Edit-Action (unmodifizierte Argumente) 6// originalRequest -- Request der "update"-Action 7if (compatible with extbase 1.3AND$this->request->originalRequest->hasArgument($argumentName) { 8$argument->setValue($this ->request->getArgument($argumentName)); 9 } elseif ($this->request->hasArgument($argumentName)) {**10**$argument->setValue($this ->request->getArgument($argumentName));11 } elseif ($argument->isRequired()) {12thrownew \F3\FLOW3\MVC\Exception\ RequiredArgumentMissingException('Required argument '" . $argumentName . '" is not set.', 1298012500);13 }14 }15 }

TODO: dontvalidate würde NICHT mehr im ValidatorResolver aufgelöst werden, sondern im ActionController::callActionMethod passieren (wo zwischen ErrorAction und normaler Action unterschieden wird) - > TODO: annotation umbenennen? @ignoreValidationErrors

TODO Berlin: discuss Namespaces // Annotation Processing // Schema
@validation.ignoreErrors

TODO: OriginalRequest in AbstractRequest hinzufügen

TODO: Error\Result sollte für **alle** Arguments gelten
-- > Arguments::haveErrors kann weggehauen werden; und Arguments::getValidationErrors umbennenen in Arguments::getMappingResult sollte Error\Result OBJEKT zurückgeben (und keinen Array) -- > Arguments::haveErrors() - > Arguments::getMappingResult()->hasErrors()

TODO: Abwärtskompatibilität -- wenn newAction erwartet Post, dann
  - > In Extbase Compatibilitätsflg einführen:
    - > tx_myext.extbaseCompatibilityVersion = 1.3
    - > PropertyMapper & Validierung wie bisher

"neuer" property mapper in Proprety\PropertyMapper

"alter" PM Property\DeprecatedPM

```
Property\Mapper {
    map($1, $2, $3, ...) {
        if (Extbase 1.3) {
          - > alter PM->map()
        } else {
            - > neuer PM->map()
        }
}
```

TODO: SubRequest

```
 1publicfunctionnew(Post$post = NULL) { 2if ($post != NULL) { 3$post->setBla('foo'); 4    } 5$this->view->assign('post', $post); 6} 7 8publicfunction
create(...Post$post) { 9}10 11publicfunctionedit(Post);1213publicfunctionupdate(Post) {14}
```

#8 - 2011-05-05 13:50 - Robert Lemke

- Project changed from Core Team to Base Distribution

- Target version deleted (788)

#9 - 2011-05-06 08:14 - Robert Lemke

- Target version set to 1228

- Position deleted (17837)

- Position set to 4

#10 - 2011-05-06 08:17 - Robert Lemke

- Project changed from Base Distribution to TYPO3 Flow Base Distribution

- Target version deleted (1228)

#11 - 2011-05-06 08:17 - Robert Lemke

- Target version set to 1.0 beta 1

- Position deleted (4)

- Position set to 3

#12 - 2011-05-06 09:29 - Sebastian Kurfuerst

- Status changed from New to Resolved

done.