Concept - Property Transformer

- Introduction
  - Basic Concept
  - Configuration
  - Default behavior
    - Basic Field
    - Composite Fields
      - CompositeField
      - SwitchableField
    - Multi-Value Field
      - MultiField
    - i18n
    - Default Value
- Implemented
  - Property
  - Implemented Transformer
    - BasicTransformer
    - CompositeTransformer
      - NoOpCompositeTransformer
    - MultiTransformer
      - MultiValueTransformer
      - MultiValueJSONTransformer
      - MultiValueChildrenNodeTransformer
      - MultiValueSubChildrenNodePropertiesTransformer
      - MultiValueSubChildrenNodeTransformer

Introduction

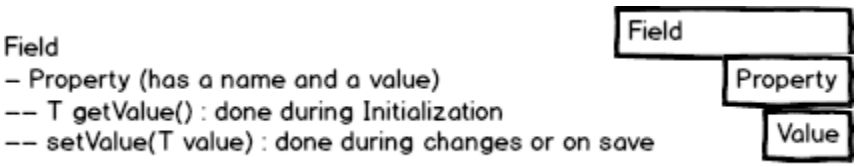
Magnolia 5.1 introduce a new way to handle properties bound to a dialog field. Until Magnolia 5.1 the only way to modify the default behavior (a field is bound to a simple property) was to override the `FieldBuilder.getOrCreateProperty(..` method.
Now by configuration we have the possibility to define custom way to read and write a value or values from a property linked to a field. This is us full for complex fields like `MultiValueField` or `CompositeField` that needs more than one simple property bound to them.

Basic Concept

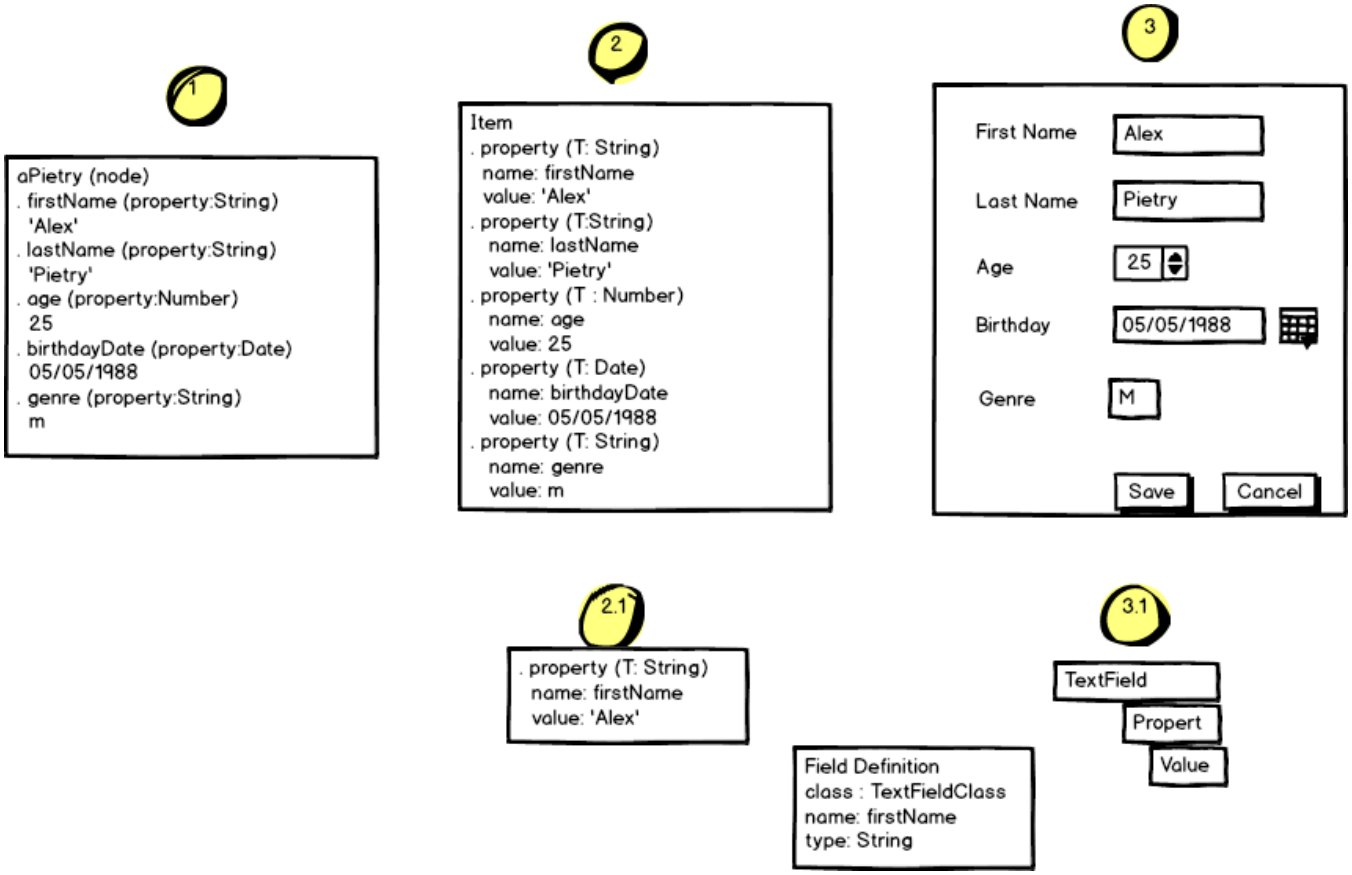
A `Field` is linked to a `Property` used to store the field value. This `Property` :

- is set by the `FieldBuilder` based on the passed `Item` .
- has a name and a value.
  - name of the property is generally the name used to store the property
  - value contains the user input.

In other words a field has a property, this property has a value.



The field property is normally simply linked to an Item property. Let's take a simple example. Assume that we have a Form displaying five fields. The following schema display the Jcr representation ***(1)***, his equivalent Item representation ***(2)*** (used in Magnolia UI and also to build the form) and the form view ***(3)***.

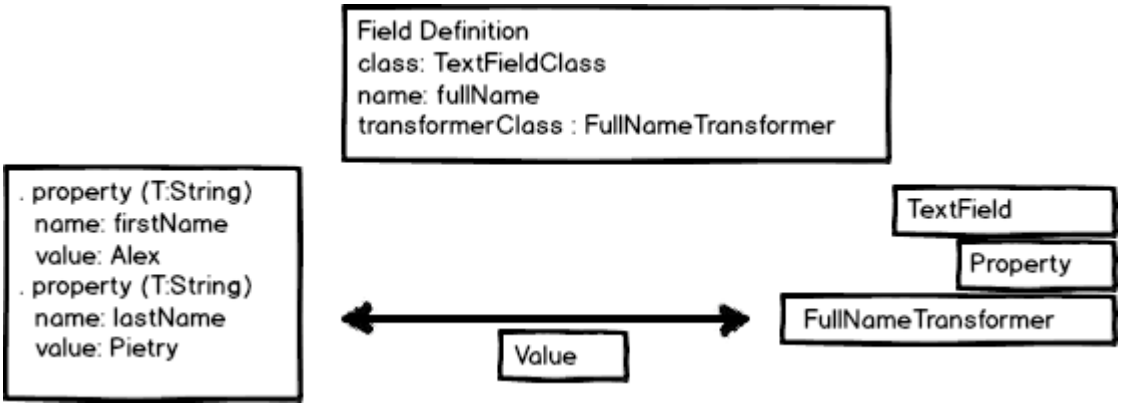


A form is build by the form builder using the Item ***(2)***. Each individual form field are build by the field builder. This builder used the field definition (that contains the field class, name,...) and the related item property to create an individual field.
In the previous example, ***(2.1)*** the field definition says that we have to create a field of type Text that has a value coming from a property called 'firstName'. The field builder, request from the Item **(2)** the property 'firstName'. It create a text field, a new property set as property datasource of the field, and associate the value of ***(2.1)*** to this property datasource ***(3.1)***.

⚠ The transformation dome between ***(1)*** and ***(2)*** is performed by the Actions (Open Dialog action and Save Dialog action). **All changes done in the item *(2)* will be propagated to the Jcr structure once a user click to the save button** (adding/removing/changing a property, or adding a sub item, ...)

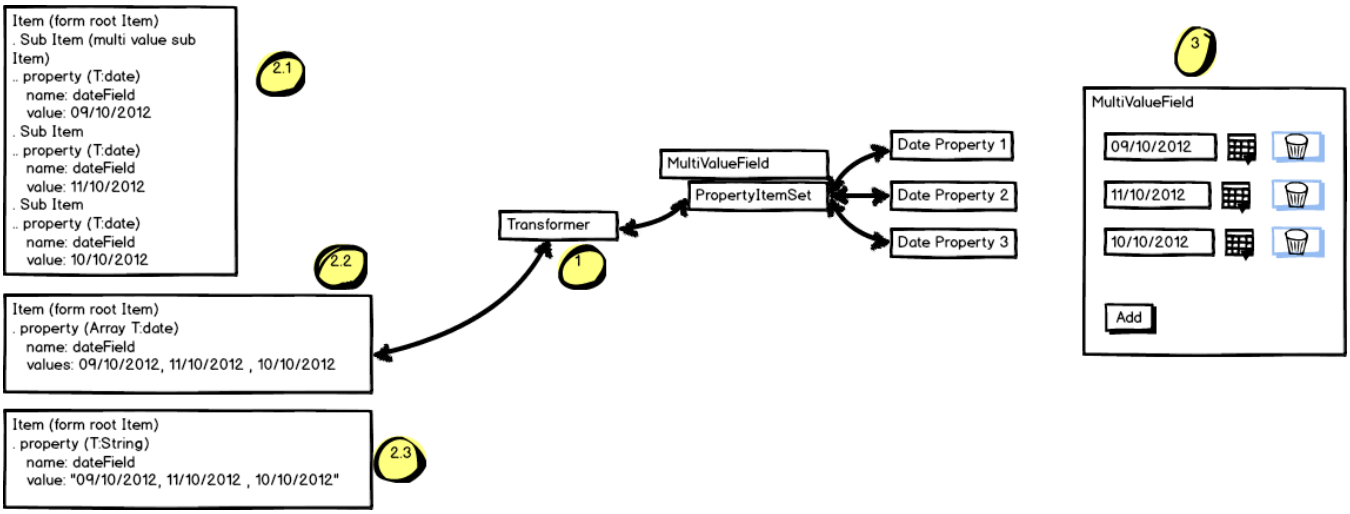
⚠ Until Magnolia 5.1, the transformation between ***(2)*** and ***(3)*** was a 1:1 translation, meaning that the individual fields where directly accessing the Item. Now Magnolia 5.1 introduce a new layer between ***(2)*** and ***(3)*** allowing to perform complexer mapping. This layer is called **Transformer**.

Assume that we have a new requirement regarding our form example: The Item ***(2)*** stay the same, but the form should only display one text field containing the full name (first and last name) . This is now possible by defining a `TransformerClass` property on the Field Definition:



In this case, the field builder will create a `TransformedProperty` as field property datasource. This `TransformedProperty` delegate to the `Transformer` the get and set value field calls.

Another good example for illustrating the `Transformer` behavior is the `MultiValueField`. The the related field `Property` can no more be a simple `Property` but rather a complex property containing several values. This property is a `PropertysetItemt` (a `Property` containing several `Property`).



The previous mockup display a Date multi value field ***(3)***. This field is associated to a `PropertysetItemt` that contains the individual Date `Property`. This property delegate the read and write to a `Transformer` ***(1)*** This `Transformer` knows how to read and write the individual Date `Property` from and to the related form item. In this example the `Transformer` is bound to an Item property of type list. This introduce another big advantage of the `Transformer`'s. For the same field it is possible to define several read/write strategy:
Store the field values into a single array, or into sub items, ...

Configuration

This configurations are done in the common fields Properties.

Property	Description	Default value	Valid values
`transformerClass`	Concrete implementation of `info.magnolia.ui.form.field.transformer.Transformer<T>`. Optional. If not defined, `info.magnolia.ui.form.field.transformer.basic.BasicTransformer<T>` is used. ⚠️ ComplexeFieldDefinition may define default `Transformer` in their constructor.		

Default behavior

Basic Field

By default, basic fields (Textm Date, Checkbox,...) uses `BasicTransformer`.

The field is created based on the related form `Item`. `BasicTransformer` will:

- Retrieve the `Item.property` if this `property` already exist on the `Item`.
  `property` is search based on the Field name defined as name property on the field definition.
- Create the `Item.property` if this `property` do not yet exist on the `Item`.
  `property` is created based on the following field definition:
  - `type`: `property` will get the desired `type`
  - `defaultValue`: if define, the string representation of the default field value is converted to a new `typed value`.

Composite Fields

Composite `Transformer` (`CompositeTransformer SwitchableTransformer`) are used by the following fields:

- `CompositeField`
- `SwitchableField`

`CompositeField`

This field use by default the `CompositeTransformer` . This `Transformer` will store each single field part of the `CompositeField` as single suffixed property.
Assume that your `CompositeField` is called '*composite*' and contains two fields: a text field called '*simpleText*' and a date field called '*simpleDate*'.
The values will be stored as following:

Node name	Value
compositesimpleText	some text value
compositesimpleDate	2006-05-01T21:47:58.230+02:00

`SwitchableField`

This field used by default the `SwitchableTransformer`. This `Transformer` will store each single field part of the `SwitchableField` as single suffixed property.
Assume that your `SwitchableField` is called '*switchable*' and contains two fields: a text field called '*simpleText*' and a date field called '*simpleDate*'.
The values will be stored as following:

Node name	Value
switchable	text (last tab selected)
switchablesimpleText	some text value
switchablesimpleDate	2006-05-01T21:47:58.230+02:00

Multi-Value Field

Multi-Value `Transformer` are used by the following field:

- `MultiField`

`MultiField`

This field is by default bound with `MultiValueTransformer` . This `Transformer` will store each single field part of the `MultiField` as a multiValue property (Basically a JCR multiValue property represented a a Typed List property).

i18n

All default `Transformer` implementation support the i18n definition.
If for example you have two language defined ('en', 'de') with 'en' set as default language:

Node name	Value
formNode	
simpleText	Simple English Text
simpleText_de	Einfache deutsche Text
multiValueText	English1,English2,English3
multiValueText_de	Deutsche1,Deutsche2

If you want to create your own implementation of `Transformer` that support i18n, your implementation will need to:

- return `true` for `Transformer.hasI18NSupport()`
- implement a compatible Magnolia i18n logic.

Default Value

`ConfiguredFieldDefinition.defaultValue` contains the String representation of the default value.

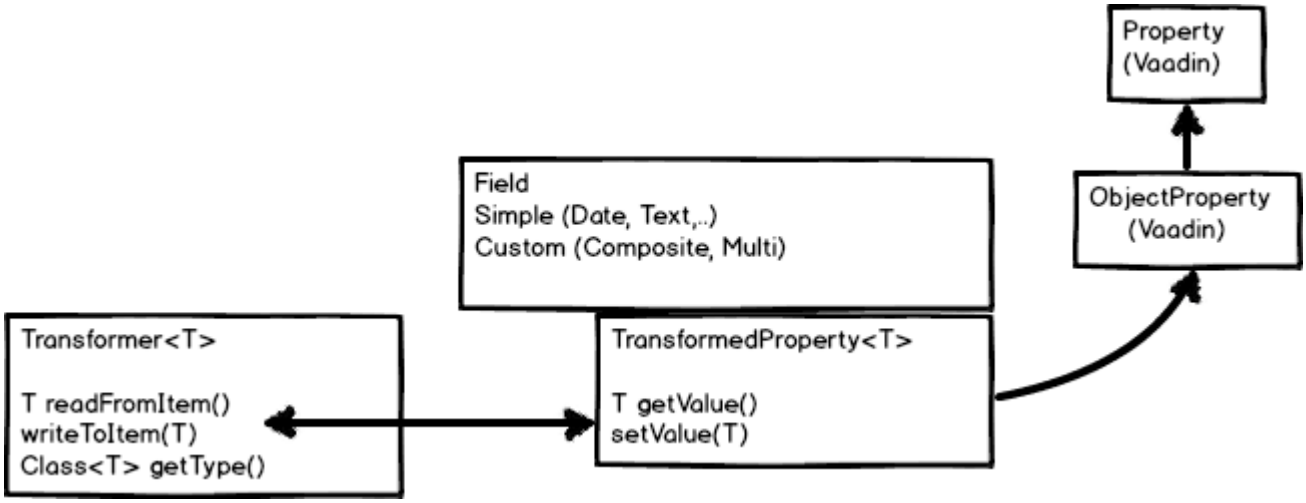
The default value is only showed the first time the related form is displayed.

This behavior is only supported by basic field .

Implemented

Property

A field needs to be linked to a property as data source. The value of the property is used to store the value entered/selected by the user on the field.
In Magnolia 5.1, every field are bound to a property that support `Transformer` called `TransformedProperty`.
This property is initialized with the configured `Transformer` and set to the field as datasource by the `FieldFactory`.



A ConfiguredField is linked to a `TransformedProperty<T>`.

- `TransformedProperty<T>` is initialized with a `Transformer`.

`TransformedProperty<T>` extend `ObjectProperty<T>`.

- value T : `Transformer.readFromItem()`
- T : `Transformer.getType()`

The `Transformer` has the responsibility to retrieve the initial value and to set the property class type.

`TransformedProperty<T>` may be of any type. For a TextField configured to handle Long (`ConfiguredFieldDefinition.type = Long`), the `<T>` will be of type `Long`.
For complexer Field (Multi, Composite) the `<T>` is of type `PropertysetItem` . This let the complex field easily handle multi property.

Implemented Transformer

`BasicTransformer`

The `property` linked to a field is retrieved and stored based on the Field's property `name` defined in the field definition.

A new `property` is created in case id does not yet exist.

- As `property` are typed, the created `property` will be of the type defined by the property named `type` coming from the field definition. (`type=Date` the property will be a `Date Object`). Default type is `String`
- If the property `defaultValue` is defined in the field definition, this value will be converted to the appropriate `type` and assigned to the newly created `property`. Otherwise the property will have a `null` value.

If the related field support `i18n` a language suffix is added to the `property` name:
For example, a field is called '*simpleText*' and has support for *'en, de, fr'*. Default language is '*en*'

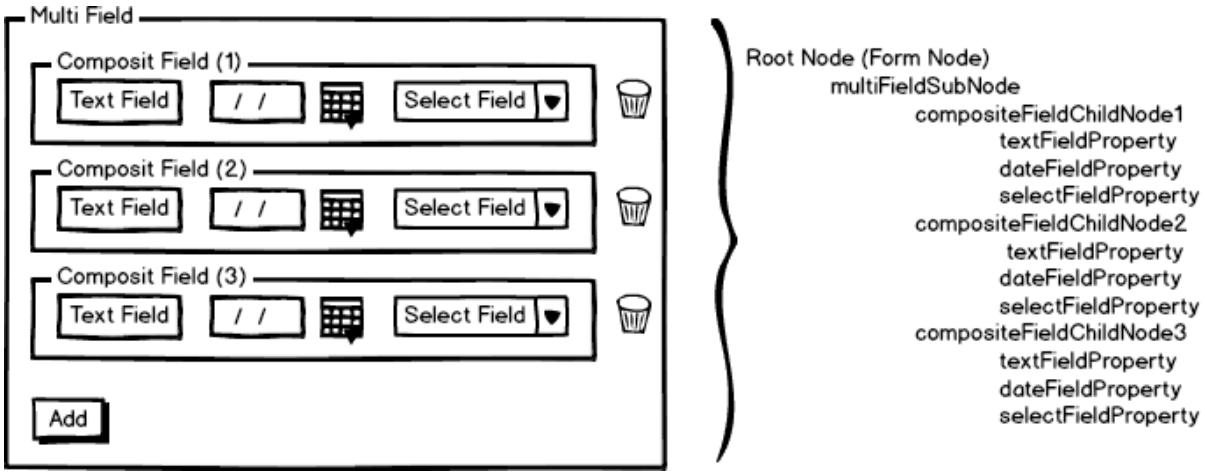
Node name	Value
simpleText	Simple English Text
simpleText_de	Einfache deutsche Text
simpleText_fr	Simple text en francais

`CompositeTransformer`

In addition to the `CompositeField` and `SwitchableField` default `Transformer`, we provide an additional `Transformer`:

`NoOpCompositeTransformer`

`NoOpCompositeTransformer` is useful if you want to combine a Multi filed storing individual field value into sub nodes with composite field as Multi field component.



In this case, the Multi field `Transformer` will read/write the Item structure and pass a properties as `PropertysetItem` to the `NoOpCompositeTransformer`. This `Transformer` just act as a property container.

`MultiTransformer`

`MultiValueTransformer`

This is the default `Transformer` set for Multi value fields. The fields values are stored in a `LinkedList<T>` . This `LinkedList<T>` is then automatically convert to a JCR multi-value-property once it is persisted.
⚠️ This will only work for simple fields like text/date/radio...

`MultiValueJSONTransformer`

`Transformer` storing the fields values as a `String` with ',' as separator.
⚠️ This will only work for simple fields like text/date/radio... and values are stored as `String`.

`MultiValueChildrenNodeTransformer`

`Transformer` storing the fields values in sub item property: (Equivalent Jcr Structure of the form Item)

Node name	Element name	Value
formNode		
00	Incremental child node name	
multi	Multi field name	Typed value in the First field

01	Incremental child node name	
multi	Multi field name	Typed value in the Second field
02	Incremental child node name	
multi	Multi field name	Typed value in the Third field

`MultiValueSubChildrenNodePropertiesTransformer`

`Transformer` storing each field values into a sub Item. Equivalent to `MultiValueChildrenNodeTransformer` but this `Transformer` is able to handle multiple values. Based on the previous `NoOpCompositeTransformer` example:

Node name	Element name	Value
formNode		
multi	Multi field name	
00	Incremental child node name	
text	Text field name composing the Composite field	Typed value in the Text field
date	Date field name composing the Composite field	Selected date
select	Select field name composing the Composite field	Selected selection
01	Incremental child node name	
text	Text field name composing the Composite field	Typed value in the Text field
date	Date field name composing the Composite field	Selected date
select	Select field name composing the Composite field	Selected selection
02	Incremental child node name	
text	Text field name composing the Composite field	Typed value in the Text field
date	Date field name composing the Composite field	Selected date
select	Select field name composing the Composite field	Selected selection

`MultiValueSubChildrenNodeTransformer`

`Transformer` creating first a child node (named as the multi field) and storing the fields values in sub node property (equivalent to `MultiValueChildrenNodeTransformer`) :

Node name	Element name	Value

formNode		
multi	Multi field name	
038a2c75-2638-48e6-a	Incremental child node name	
multi	Multi field name	038a2c75-2638-48e6-a6ba-9bd2a9fe6c78
72e2ef55-6c11-4b0e-8	Incremental child node name	
multi	Multi field name	72e2ef55-6c11-4b0e-8e02-d47c4ad41083
11bbf78b-4ecf-4e9f-a	Incremental child node name	
multi	Multi field name	11bbf78b-4ecf-4e9f-a06e-6181ef56d98c