Product category (CatalogElement)

The CatalogElement class defines the generic product and category model which is used within eZ Commerce. It inherits from the general class ValueObject which offers a convenient way of setting properties of instances via the constructor and makes these properties public readable (like $valueObject->name).

Inheriting from CatalogElement there are some sub classes worth mentioning:

- CatalogElementContainer
  - CatalogNode
- ProductNode
  - OrderableProductNode
  - ProductNodeContainer
    - GroupedProductNode
    - ComplexProductNode
- ProductType

Predefined properties for `CatalogElement`

Each `CatalogElement` has predefined properties. These methods are validated automated on constructor by the `validateProperties()` method.

Identifier	Type	Description
`name`	`string`	The name of the catalog
`text`	`string`	A short introduction text for the catalog
`image`	`ImageField (FieldInterface)`	An image for the catalog
`path`	`array`	The path of the catalog (array of identifier)
`url`	`string`	The internal URL of the catalog. This url should not be used for generating a link! Please use `seoUrl` instead
`seoUrl`	`string`	The human readable URL of the category
`permanentUrl`	`string`	The internal permanent URL
`parentElementIdentifier`	`string`	The unqique identifier of the parent catalog
`identifier`	`string`	The unqique identifier
`dataMap`	`FieldInterface[]`	A list of Field objects
`cacheIdentifier`	`int|string`	cache identifier of element to use as key in cache storage

> There are 4 public methods to set properties:
>
> - setImage,
> - setName,
> - setText,
> - setCacheIdentifier

Validators for `CatalogElement`

List of possible validators to used when attributes are set in CatalogElement

Name	Parameters	Description

`validateStringAttribute`	`$value, $attribute`	checks if the value is a valid string
`validateBooleanAttribute`	`$value, $attribute`	checks if the value is a valid boolean
`validateFloatAttribute`	`$value, $attribute`	checks if the value is a valid float
`validateIntegerAttribute`	`$value, $attribute`	checks if the value is a valid integer
`validateFieldAttribute`	`$value, $attribute, $fieldType`	checks if the value is of given Field Type
`validateArrayAttribute`	`$value, $attribute`	checks if the value is an array
`validateArrayOfAttribute`	`$value, $attribute, $class`	checks if the value is an array of concrete class (interface)

Concrete implementations of the `CatalogElement` class requires you to extend `validateProperties()`. This method is used to validate all given properties in the constructor of the class. As you do not want to overwrite the given implementation but extend it, you want to use `parent::validateProperties($properties)` in your implementation.

Example

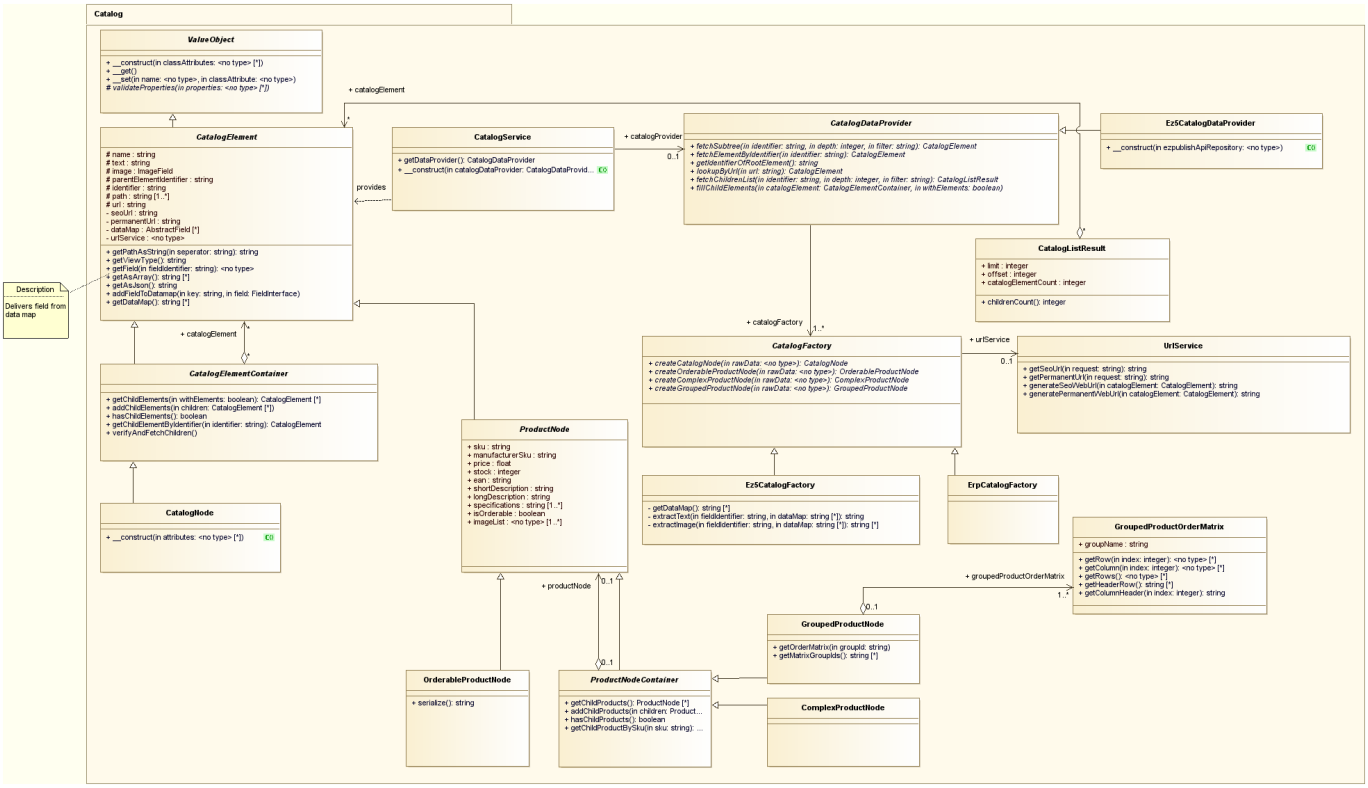
Example: Extending validateProperties()

```
/**
 * Simple class which extends OrderableProductNode (CatalogElement)
 */
class MyOrderableProductNode extends OrderableProductNode
{
    /**
     * An additional instance of a Field
     * @var FieldInterface
     */
    protected $myField;

    protected function validateProperties(array $properties =
array())
    {
        // call validateProperties() from super class to validate
default properties
        parent::validateProperties($properties);

        // ... now validate my specialized property "myField"
        if (
            isset($properties['myField'])
            && !($properties['myField'] instanceof FieldInterface)
        ) {
            $message = 'Attribute "myField" has wrong data type: '
                    . gettype($value)
                    . '. Instance of class FieldInterface expected.';
            throw new \InvalidArgumentException($message);
        }
    }
}
```

Class diagram



Sub classes

- CatalogElementContainer & CatalogNode
- ProductNode & OrderableProductNode
- ProductType