castValue() Function

Syntax	`aSimpleChildClassObject = castValue(aSimpleParentClassObject)`
Semantics	Assigns a simple valued object of base class type to an object of its sub-class type. Supported simple value types are: **String**, **Integer**, **Float**, **Boolean**, and **DateTime**. If the generalization between the base- and sub-class type is of stereotype <<E2EValueGeneralization>>, all constraints given as tagged values of this stereotype are validated while doing the assignment. Allowed constraints are:

Constraint	Description
Pattern	**String** values may be constrained by a regular expression.
Max Length	Max length of **String** values.
Min Length	Min length of **String** values.
Length	Exact length of **String** values.
Enumeration	A set of allowed values for **Strings** or **Integers**.
Max Value	Max value of **Integer** or **Float**.
Min Value	Min value of **Integer** or **Float**.
Fraction Digits	Number of digits after the decimal separator of **Float**.

All constraints can be combined. An error is thrown if a constraint is violated. If the classes are imported from WSDL or XML schema files, all supported schema constraints are mapped to tagged values of the <<E2EValueGeneralization>>.

Error Codes	Error domain `FUMSM/FUMSMCV`. Find the related error codes on page System Errors.

On this Page:

Related Pages:

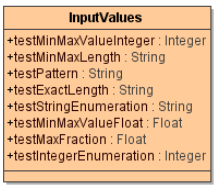
- cast() Operation
- System Errors

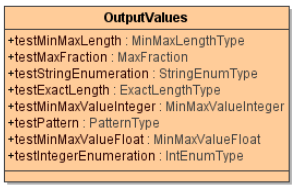
Example

Example File (Builder project Basic Modeling/Data):

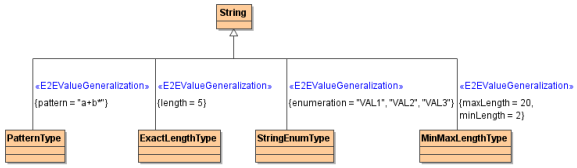
<your example path>\Basic Modeling\Data\uml\castValue.xml

The typical usage scenario of this function is mapping from generic values like **Strings**, **Integers**, and **Floats** to more specific types. For example, the following **InputValues** class has to be mapped to the **Output Values** class:

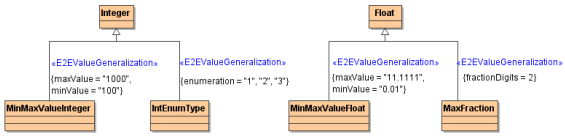




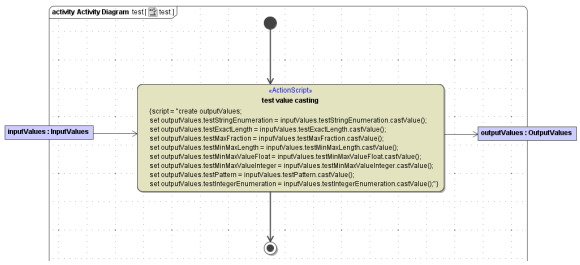
Frequently, classes such as **OuputValues** is imported from XML Schema files and their simple types are not plain **Strings** or **Integers** but more specific types. For example, the **String** related types in the `castValue()` example look like



and the numeric types are defined as



If you want to assign for example a **String** to a more specific type such as **StringEnumType**, then you have to downcast the values as shown in the example action below. But in contrast to the standard `cast()` function, `castValue()` will check the enumeration constraint, i.e. whether the **String** takes only the allowed values `VAL1`, `VAL2`, or `VAL3`. This happens at runtime.
If the constraint is violated, an exception is thrown at the very place the mis-assignment takes place. Thus, it is recommended to use `castValue()` wherever possible. The error codes are listed below.



Hierarchies of Constraints

It is possible that classes inherit from already constraint types. The following diagram shows the class **MinMaxLengthType** that specializes the class **PatternType**. This means that **MinMaxLengthType** inherits all constraints of **PatternType** whereas the constraints of the child class override the parent constraints. For example, **MinMaxLengthType** has the constraint **maxLength = 10** which overrides the constraint **maxLength = 20** of **PatternType**.

The same applies to enumerations. This means effectively that the enumeration values are *merged*. For example, the allowed values of **AdditionalEnumeration** objects are **SUBVAL1**, **SUBVAL2**, **VAL1**, **VAL2**, **VAL3**.