Calling Class Operations

Operations can be defined on classes and they are implemented by an activity. That activity needs to be assigned to the operation.
Operations can be called by a **call operation** or within **action script**.

On this Page:

Related Pages:

- Using the Suggestion Features of the Action Script Editor
- Local Variables

Example File (Builder project Basic Modeling/ClassOperation):

	<your example path>\Basic Modeling\ClassOperation\uml\callClassOperations.xml

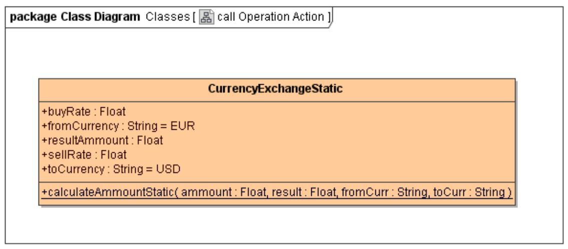
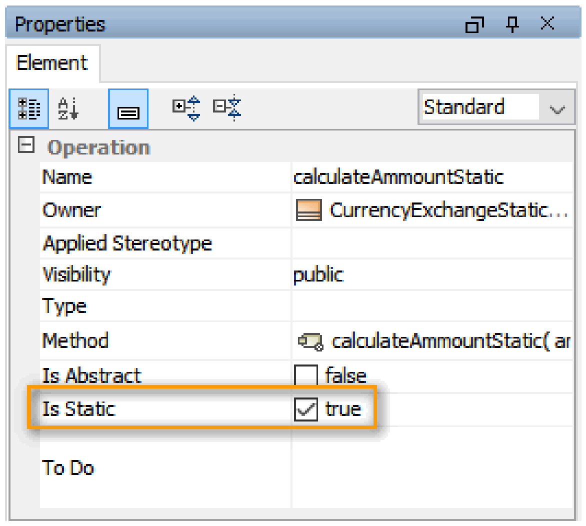
Call Operation Action

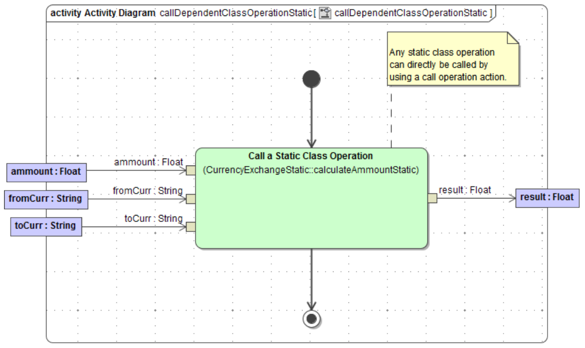
A class operation can be called by a call operation action in two ways:

- **The class operation is static.**
  The operation can be called without instantiating the class.
- **The class operation is not static.**
  A local instance of the class has to be created and the operations is called on that instance.

Static Call Operation Action

This example shows how to call a static class operation by call operation action.

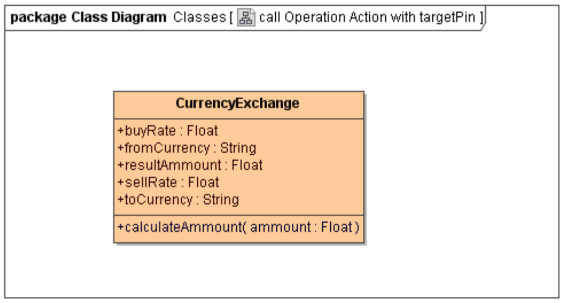
 package Class Diagram Classes [call Operation Action] CurrencyExchangeStatic +buyRate : Float +fromCurrency : String = EUR +resultAmmount : Float +sellRate : Float +toCurrency : String = USD +calculateAmmountStatic(ammount : Float, result : Float, fromCurr : String, toCurr : String)	Define a class with an operation.
 Operation Name: calculateAmmountStatic Owner: CurrencyExchangeStatic... Applied Stereotype: Visibility: public Type: Method: calculateAmmountStatic(ar Is Abstract: false Is Static: ✓ true To Do:	Make this operation static by ticking **Is Static** on the operation's specification.

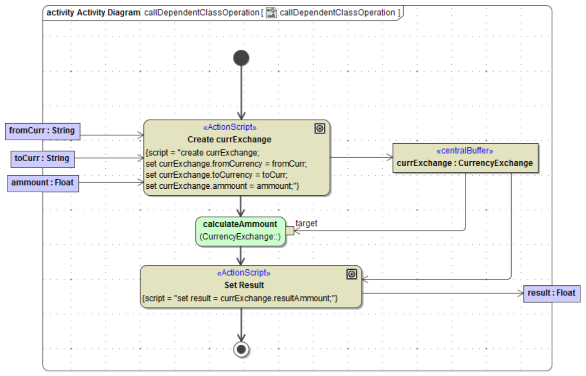
 activity Activity Diagram callDependentClassOperationStatic [callDependentClassOperationStatic] Any static class operation can directly be called by using a call operation action. Call a Static Class Operation (CurrencyExchangeStatic::calculateAmmountStatic) ammount : Float fromCurr : String toCurr : String result : Float	This class operation can easily be used in any activity diagram. Easy way is to drag and drop the operation from the containment tree directly into your activity diagram.A call operation action linking to the selected class operation will be created automatically - together with all needed pins for you to connect to the related objects via object flows.

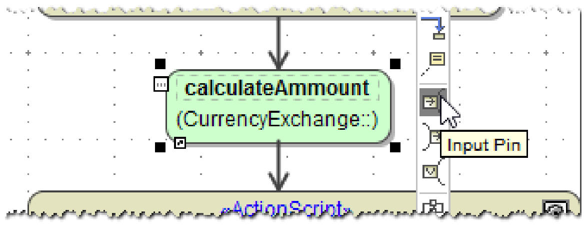
All static class operations can be called by call operation action without needing to create an instance of the class.

Important to know: With static operations, no attributes of the class itself can be accessed within a class operation. Any `self` context does not exist because no instance of the class is referenced.

Referenced Call Operation Action

 package Class Diagram Classes [call Operation Action with targetPin] CurrencyExchange +buyRate : Float +fromCurrency : String +resultAmmount : Float +sellRate : Float +toCurrency : String +calculateAmmount(ammount : Float)	To use data on the class, call a class operation by referencing an instance of this class by a target pin. In this case, you do not need to define the class operation as static.

 activity Activity Diagram callDependentClassOperation [callDependentClassOperation]
fromCurr : String
toCurr : String
ammount : Float
«ActionScript»
Create currExchange
{script = "create currExchange;
set currExchange.fromCurrency = fromCurr;
set currExchange.toCurrency = toCurr;
set currExchange.ammount = ammount;"}
«centralBuffer»
currExchange : CurrencyExchange
calculateAmmount
(CurrencyExchange::)
target
«ActionScript»
Set Result
{script = "set result = currExchange.resultAmmount;"}
result : Float
	In the activity diagram, create an instance of the class by action script. This central buffer object has to be connected to a target pin on the call operation action by an object flow.

 calculateAmmount (CurrencyExchange::) Input Pin «ActionScript»	To create this target pin choose a new input pin from the object short menu bar. Now you can draw the object flow from the central buffer of your class to the target pin.

These are the two easy ways to directly call a class operation within your activity diagram.

Call Operations by Action Script

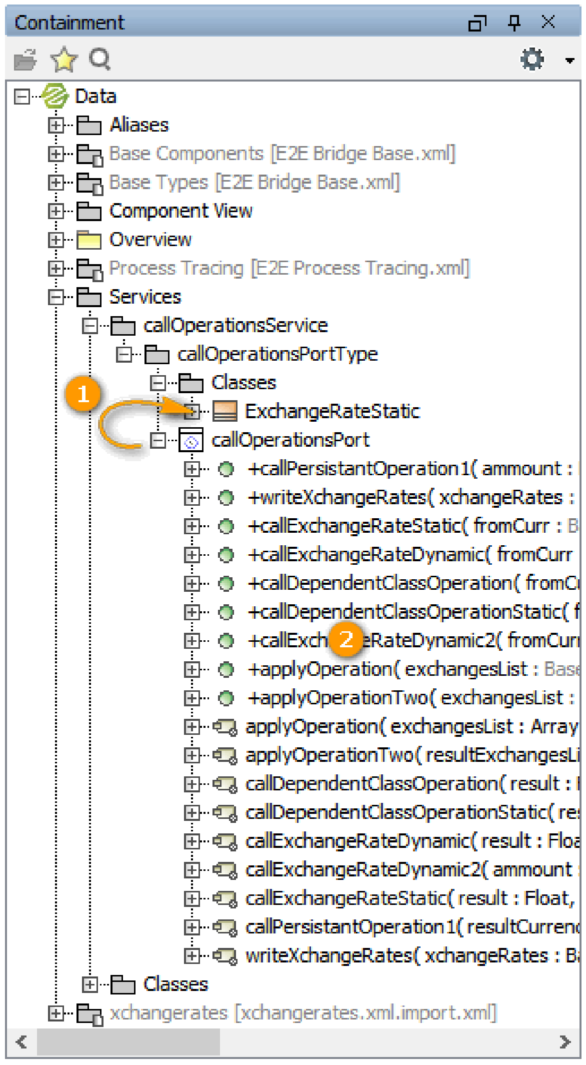
Example File (Builder project Basic Modeling/ClassOperation):

<your example path>\Basic Modeling\ClassOperation\uml\callClassOperations.xml

You can easily use self defined technical functions directly from your action script by implementing them as a class operation. Calling these class operations in action script is possible in two different ways:

- Either call a static class operation or ...
- call a class operation of a referenced class object.

In both cases, the class containing the operation you want to call has to be made available to action script usage via a <<use>> dependency.
The effect of the <<use>> dependency relates to the package structure of the Builder project:

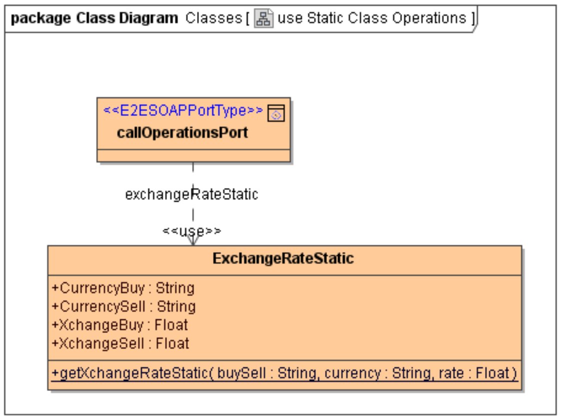


```
Data
  Aliases
  Base Components [E2E Bridge Base.xml]
  Base Types [E2E Bridge Base.xml]
  Component View
  Overview
  Process Tracing [E2E Process Tracing.xml]
  Services
    callOperationsService
      callOperationsPortType
        Classes
          ExchangeRateStatic
        callOperationsPort
          +callPersistantOperation1( ammount :
          +writeXchangeRates( xchangeRates :
          +callExchangeRateStatic( fromCurr : B
          +callExchangeRateDynamic( fromCurr
          +callDependentClassOperation( fromCu
          +callDependentClassOperationStatic( f
          +callExchangeRateDynamic2( fromCurr
          +applyOperation( exchangesList : Base
          +applyOperationTwo( exchangesList :
          applyOperation( exchangesList : Array
          applyOperationTwo( resultExchangesLi
          callDependentClassOperation( result :
          callDependentClassOperationStatic( re
          callExchangeRateDynamic( result : Floa
          callExchangeRateDynamic2( ammount
          callExchangeRateStatic( result : Float,
          callPersistantOperation1( resultCurrenc
          writeXchangeRates( xchangeRates : B
    Classes
  xchangerates [xchangerates.xml.import.xml]
```

Service operations implemented below **callOperationsPort** cannot access classes defined in package **Classes**, because they are not part of the same branch in the containment tree.

If a superior class (like **callOperationsPort** in our example) is having a <<use>> dependency to a another class 1 implementing a class operation (like **ExchangeRateStatic** in our example), all subordinated objects 2 of that class (**callOperationsPort** in our example) can instantiate the dependent class and use their operations via that <<use>> dependency.

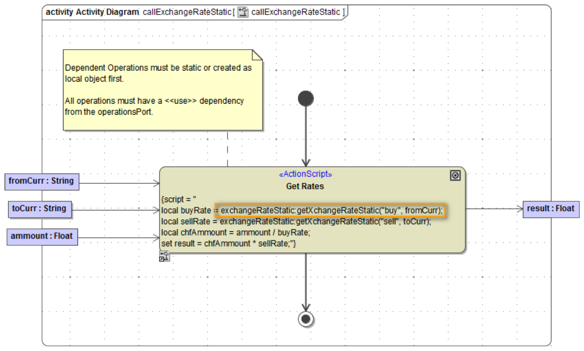
The example below shows how to make class **ExchangeRateStatic** and operation **getXchangeRateStatic** available for all other subordinated action scripts of the <<E2ESOAPPortType>> **callOperationsPort** by drawing a <<use>> dependency from the port type class to the implementing class.



Give the <<use>> dependency a name, like **exchangeRateStatic** in our example. Via this name the class and its operations can be accessed within action script.

Static Class Operations Within Action Script

Static class operations have to be made available to action script via a <<use>> dependency as mentioned above.


activity Activity Diagram callExchangeRateStatic [callExchangeRateStatic]
Dependent Operations must be static or created as local object first.
All operations must have a <<use>> dependency from the operationsPort.
«ActionScript»
Get Rates
fromCurr : String
toCurr : String
ammount : Float
result : Float

```
{script = "
local buyRate = exchangeRateStatic:getXchangeRateStatic("buy", fromCurr);
local sellRate = exchangeRateStatic:getXchangeRateStatic("sell", toCurr);
local chfAmmount = ammount / buyRate;
set result = chfAmmount * sellRate;"}
```

Type in the name of the <<use>> dependency or select it out of the selection offered by the Action Script Editor. With a colon you can select the available operations on the referenced class. Any parameters are given within the brackets. So finally an operation call should look like:

```
exchangeRateStatic:getXchangeRateStatic("buy", fromCurr);
```

Be aware that all the referenced parameters are available as parameters or central buffers associated to the action script or available as local variables.

If you want to call a static operations of the current class, you can use keyword **self** to identify the operation. For example:

```
self:myStaticOperation()
```

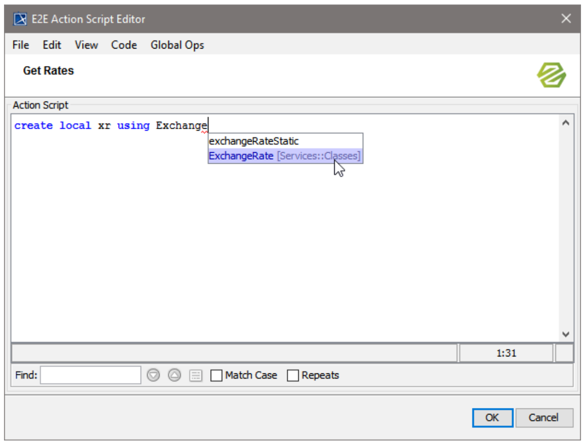
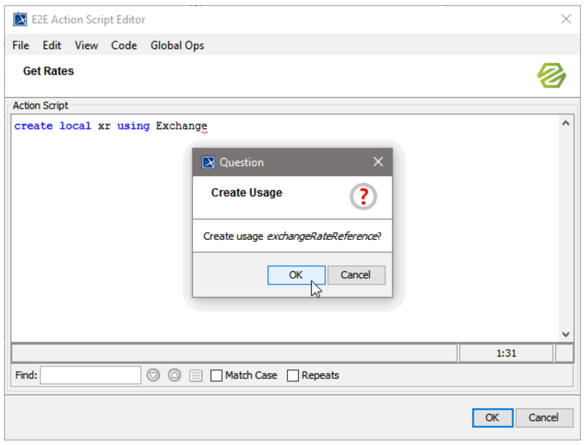
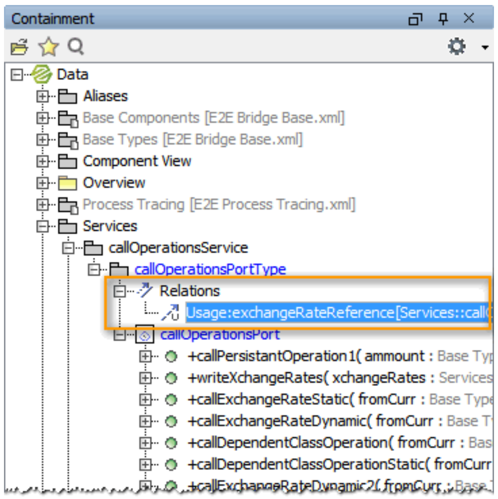
Referenced Class Operation Within Action Script

To create an object of a class within action script, this class has to be made available to action script via a <<use>> dependency as mentioned above.

If the operation is not static, an instance of the class has to be created before calling the operation. This can be done in two ways:

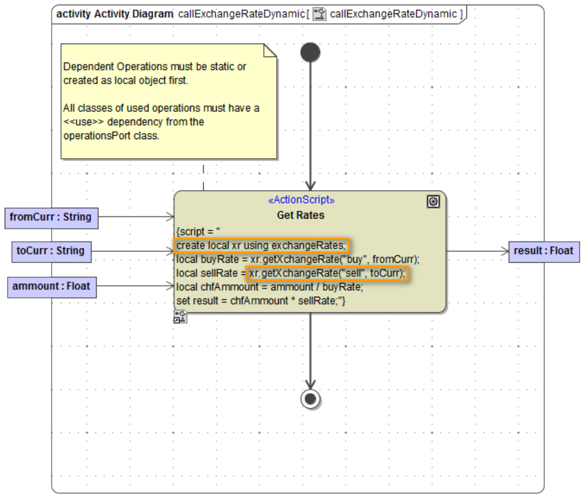
- by creating a local object within the action script
- by creating an instance of the class outside the action script and then referencing it via an object flow

In fact, when no <<use>> dependency has been created yet, MagicDraw will suggested to create one for you:

 `create local xr using Exchange` exchangeRateStatic ExchangeRate [Services::Classes]	Upon creating e.g. the local object, simply start typing the name of the class and press **Ctrl + Space** to force the suggestion list. Select the class you want to use an operation of and ...
 `create local xr using Exchange` Question Create Usage Create usage *exchangeRateReference*?	MagicDraw will suggest to create a <<use>>dependency automatically. The name of this dependency will be **<name of the related class in lower-case>Reference**. You can change this name in the containment tree later on, if you like.
 Containment Data Aliases Base Components [E2E Bridge Base.xml] Base Types [E2E Bridge Base.xml] Component View Overview Process Tracing [E2E Process Tracing.xml] Services callOperationsService callOperationsPortType Relations Usage:exchangeRateReference[Services::call callOperationsPort +callPersistantOperation1(ammount : Base Ty +writeXchangeRates(xchangeRates : Services +callExchangeRateStatic(fromCurr : Base Type +callExchangeRateDynamic(fromCurr : Base T +callDependentClassOperation(fromCurr : Bas +callDependentClassOperationStatic(fromCurr +callExchangeRateDynamic2(fromCurr : Base	You can find the new <<use>> dependency in a package **Relations** in the package your activity resides in.

Class Instance as a Local Object

Figure: Call reference class operation on local object


activity Activity Diagram callExchangeRateDynamic[callExchangeRateDynamic]
Dependent Operations must be static or created as local object first.
All classes of used operations must have a <<use>> dependency from the operationsPort class.
«ActionScript»
Get Rates
fromCurr : String
toCurr : String
ammount : Float
result : Float

```
{script = "
create local xr using exchangeRates;
local buyRate = xr.getXchangeRate("buy", fromCurr);
local sellRate = xr.getXchangeRate("sell", toCurr);
local chfAmmount = ammount / buyRate;
set result = chfAmmount * sellRate;"}
```

First, a local instance of the class has to be created:

```
create local xr using exchangeRates;
```

Syntax: **`create local`** `<objectName>` **`using`** `<nameOfUseDependency>`

The local object now is available within the action script and also shown in the suggestion list of available objects.
To call an operation the statement looks like:

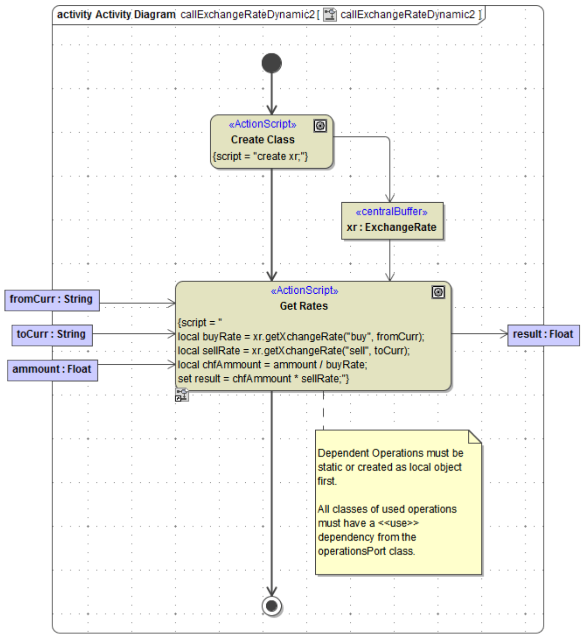
`<name of the referenced object>.<operation name>(Parameter, anotherParameter);`

```
set response = xr.getXchangeRate("sell", toCurr);
```

Class Instance Created Outside Action Script

In case an instance of the object is created outside the action script, it can be referenced by an object flow into the action script. Then the operation is available the same way like referenced by a local object.

Figure: Call referenced class operation on input object

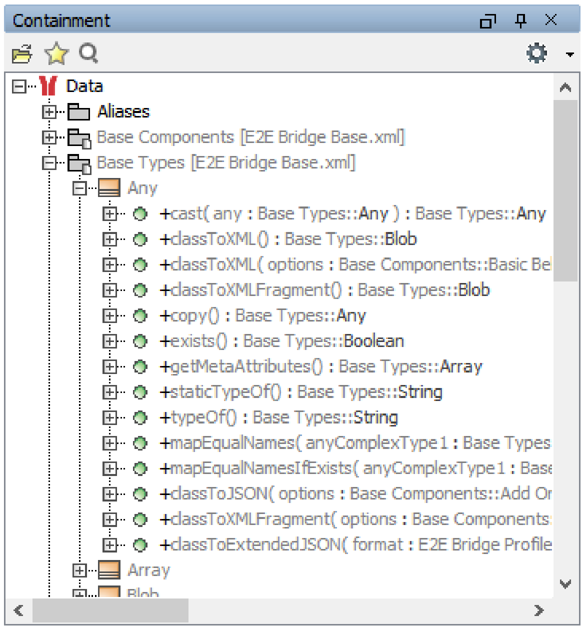

activity Activity Diagram callExchangeRateDynamic2[callExchangeRateDynamic2]
«ActionScript»
Create Class
{script = "create xr;"}
«centralBuffer»
xr : ExchangeRate
«ActionScript»
Get Rates
fromCurr : String
toCurr : String
ammount : Float
result : Float
Dependent Operations must be static or created as local object first.
All classes of used operations must have a <<use>> dependency from the operationsPort class.

```
{script = "
local buyRate = xr.getXchangeRate("buy", fromCurr);
local sellRate = xr.getXchangeRate("sell", toCurr);
local chfAmmount = ammount / buyRate;
set result = chfAmmount * sellRate;"}
```

Beside calling a class operation it is also possible to use class operations in context of **apply**, **reduce** or **s elect each from**. For additional information see example **callClassOperation**.

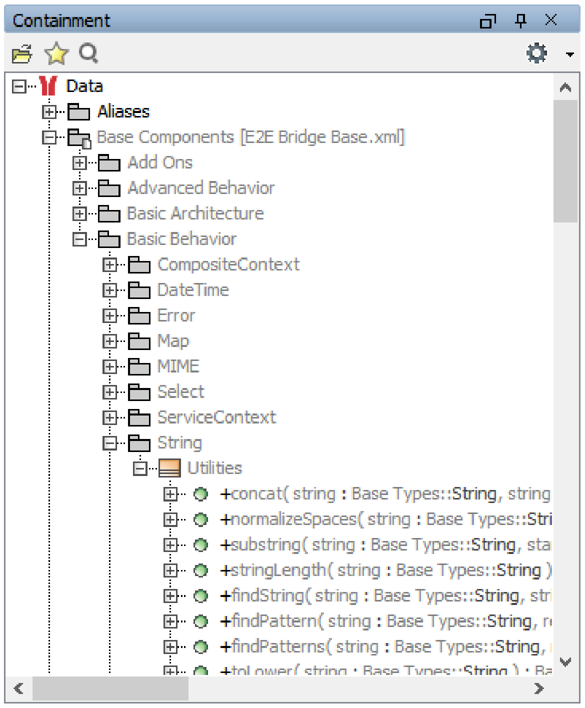
Calling Base Type Operations

All of the above also refers to **Base Type** operations. The xUML **Base Types** are implemented to the Builder/xUML Runtime as classes with related operations.



```
Containment
Data
  Aliases
  Base Components [E2E Bridge Base.xml]
  Base Types [E2E Bridge Base.xml]
    Any
      +cast( any : Base Types::Any ) : Base Types::Any
      +classToXML() : Base Types::Blob
      +classToXML( options : Base Components::Basic Be
      +classToXMLFragment() : Base Types::Blob
      +copy() : Base Types::Any
      +exists() : Base Types::Boolean
      +getMetaAttributes() : Base Types::Array
      +staticTypeOf() : Base Types::String
      +typeOf() : Base Types::String
      +mapEqualNames( anyComplexType1 : Base Types
      +mapEqualNamesIfExists( anyComplexType1 : Base
      +classToJSON( options : Base Components::Add Or
      +classToXMLFragment( options : Base Components
      +classToExtendedJSON( format : E2E Bridge Profile
    Array
    Blob
```

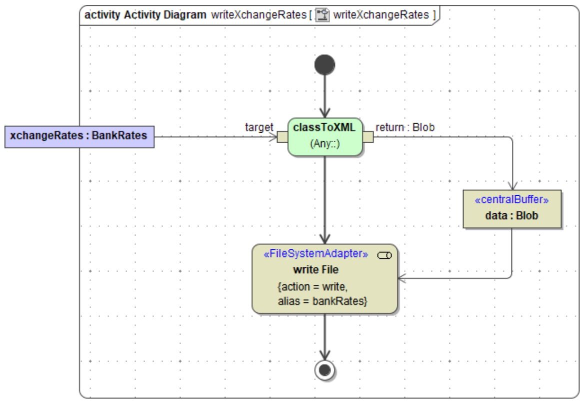
In package **Base Types** in the Builder, you can find the base types and their related operations.



```
Containment
Data
  Aliases
  Base Components [E2E Bridge Base.xml]
    Add Ons
    Advanced Behavior
    Basic Architecture
    Basic Behavior
      CompositeContext
      DateTime
      Error
      Map
      MIME
      Select
      ServiceContext
      String
        Utilities
          +concat( string : Base Types::String, string
          +normalizeSpaces( string : Base Types::Stri
          +substring( string : Base Types::String, sta
          +stringLength( string : Base Types::String )
          +findString( string : Base Types::String, str
          +findPattern( string : Base Types::String, r
          +findPatterns( string : Base Types::String,
          +toLower( string : Base Types::String ) : Ba
```

Package **Base Components** contains some more type related, **s tatic** operations that can be used likewise.

You can add operations that are defined on base types or on base type utilities to an activity diagram like any other class operation.



Exceptions

For technical reasons, this does not work for

- macros
- operations that can have a variable count of parameters like the concat() Operation: concat(string1:String, string2:String, string3:String, ...)
- operations where return type determines the behavior, like e.g. the xmlToClass() Operation

In most cases, however, you will use such operations in action script.