Functions and function calls

What is it for?

Functions are used to allow common operations to take place in separate, easily reusable modules.

A function is a special relationship between inputs and outputs. The function transforms the input into the desired output. In Blueriq, functions do exactly that, and they can make use of the rule engine and flows. As the inputs and outputs of a function are clearly defined, it is clear what information is expected by the function, and which attributes have an updated value after the function call. The internal workings of a function are hidden, to reduce the long lists in Blueriq Encore and the mental load of the Business Engineer. Both these points lead to a high maintainability of your model.

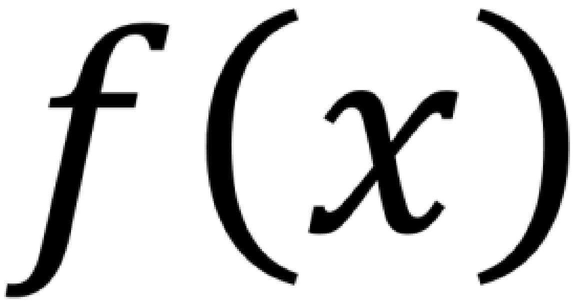


Table of contents

Intended use

You may think of a function as a computation or a service which is provided to you as a caller of the function.

Examples of functions are:

- Calculating the interest that you need to pay for a loan over the next years
  - Inputs: Loan amount, Number of years, Interest rate, Monthly payment
  - Output: Total amount of money owed after given number of years
- Deciding if you are eligible for a subsidy for solar panels
  - Inputs: Type of roof, Size of roof in square meters, Number of solar panels, Costs of solar panels, Number of inhabitants
  - Outputs: Decision, Subsidy amount, Reason for denial

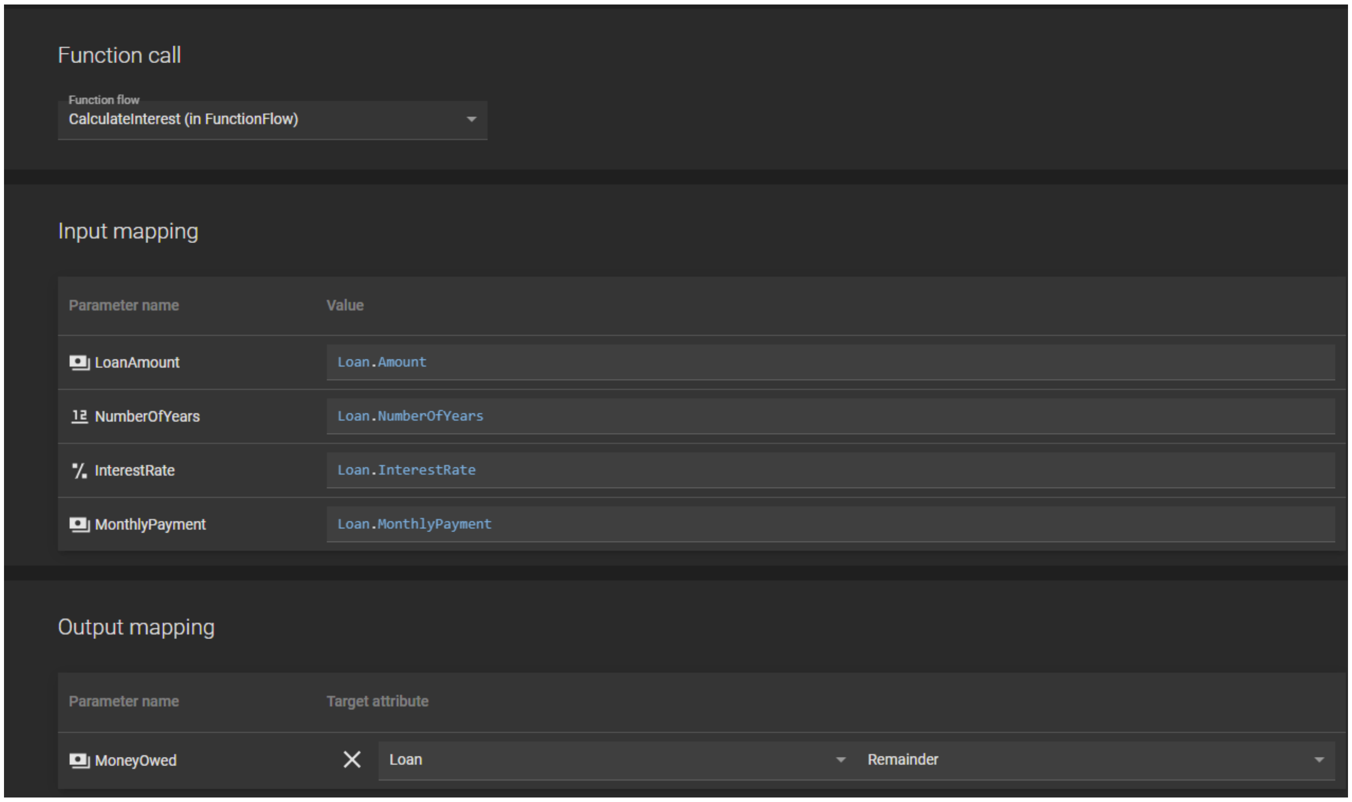
Blueriq functions are intended for decoupling smaller calculations. This is what they are good at and should be used for.

There are other scenarios, in which you might consider functions, for which a function may not be the best choice. Below there are some scenarios discussed:

- A function which has side effects. For example when a function stores data in an aggregate or writes data to an external system. This would mean that the outcome of a function can differ after calling it multiple times. This may be confusing to users which expect the outcome to be identical.
- A function which interacts with the process engine. In basis this is the same as above, as the state of the process engine would change, and the result of the function may be different on the second call. In fact using the process engine in a function currently does not work at all.
- A function with complex parameters. When you need to send a complex structure with instances which have relations to other instances and so on, then a web service is a much better option. Functions intentionally have simple parameters.
- A recursive function. You could model a function which calls another function. There is however no mechanism in place to prevent infinite looping and you have to be really carefull on what you do. Therefore this is discouraged.

Modeling

A function call can be compared to a service call in your flow. It behaves more or less the same, except that a different part of the model is used for execution. To use a given function, you create a function call by filling in the input parameters, and mapping the output parameters. Here is an example:


Function call

Function flow: CalculateInterest (in FunctionFlow)

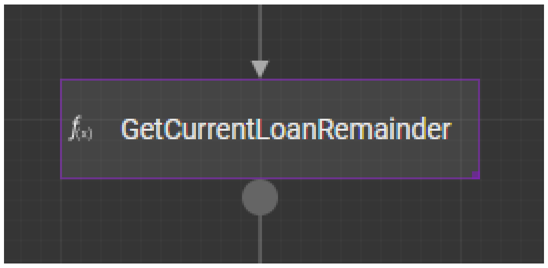
Input mapping

Parameter name	Value
LoanAmount	`Loan.Amount`
NumberOfYears	`Loan.NumberOfYears`
InterestRate	`Loan.InterestRate`
MonthlyPayment	`Loan.MonthlyPayment`

Output mapping

Parameter name	Target attribute	
MoneyOwed	Loan	Remainder

Now, you can place this function call in a flow from which you want to consume the function:



You can use the exception exit in case the function fails to execute.

Functions in library

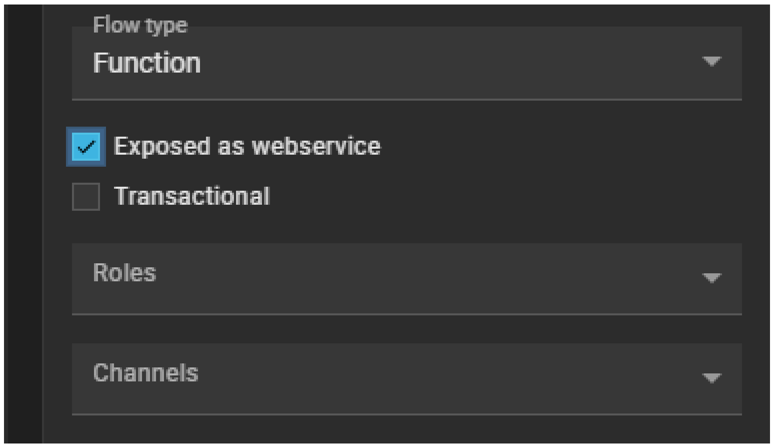
Of course functions can be modeled in a project structure with multiple stacked modules, when doing this you have to be aware of this fact: when calling the function, by default it will select the function definition from the lowest implementation module possible. It is possible to call the function if specified at a higher (implementation) module, but you have to explicitly point to that definition.

Input validation

When modeling a function, it is possible to check the validity of your input parameters. The input type is always checked, but other validations can be added to the parameters. Please note that when a multi-valued attribute as target attribute, your parameter also becomes multivalued. The validations can be added by adding a validation rule or validation type to the input attribute. This validation will then be triggered when calling the function. When a validation is triggered there will be an error message, on screen for the user when the function is called from within Blueriq or in the JSON result when exposed and called as a webservice.

Exposing the function as a REST webservice

Blueriq offers the option to expose a function as a REST webservice. This is done by selecting the Exposed as webservice checkbox in the properties tab.


Flow type
Function
☑ Exposed as webservice
☐ Transactional
Roles
Channels

When you reload projects in the runtime after saving the flow with this option checked, the function can be approached as a webservice.

The URL of the webservice will be:

```
http://<environment>:<port>/Runtime/server/api/v2/function?
project=<project>&function=<flowname>&version=<branch>&module=<module>
```

In which <environment>, <port>, <project>, <flowname> & <branch> can be found by selecting the flow and right clicking on the start button on the runtime overview and then copying the link address.

The call to the webservice is a JSON message containing the parameters defined in the function. The HTTP method being called is POST.

When calling the webservice, the HTTP headers Content-Type and `Accept` have to be set with `application/json`

For the example used to calculate the loan remainder this would look like:

```
{
        "LoanAmount":100000,
        "NumberOfYears":5,
        "InterestRate":3.5,
        "MonthlyPayment":500
}
```

The webservice will also return a JSON message containing the output parameters of the function.

For the example it would look like:

```
{
        "MoneyOwed":25000
}
```

Changing contract

When a function flow is exposed as a web service, changing the parameters may break contract with the using party. Be careful when changing the parameters while the exposed web service is already in use.

Shortcut to exposed function flow

It is possible to make a shortcut to an exposed function flow.

To do this you must add a shortcut in the application.properties with the following content:

```
blueriq.function-shortcut.{shortcutname}.project=
blueriq.function-shortcut.{shortcutname}.version=
blueriq.function-shortcut.{shortcutname}.module=
blueriq.function-shortcut.{shortcutname}.function=
```

The url to call the shortcut will be: /Runtime/server/api/v2/function/shortcut/{shortcutname}/

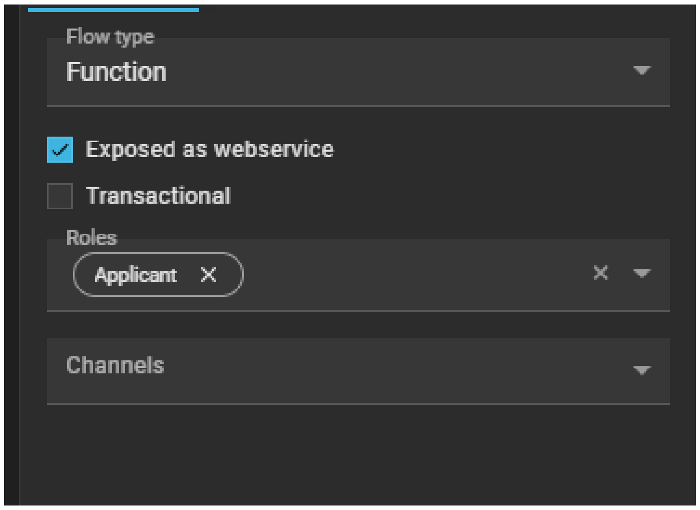
If the property `blueriq.production.shortcutsOnly` is set and the runtime is running in production mode, only shortcuts can be used to start a function.

OpenAPI feed for exposed function flows

An OpenAPI feed of Function Flows exposed as a webservice is provided by the Runtime, see OpenAPI for more information.

Securing an exposed function flow

By default, an exposed function flow can be called without authentication. It is necessary to add at least one role to make sure that the endpoint is secured and the user is authorized. If you add one or more roles to the function flow, the runtime will require credentials.


Flow type: Function
☑ Exposed as webservice
☐ Transactional
Roles: Applicant

Further Reading

- Flow type: function