Rocket Uniface Library 10.4

params...endparams

Declare the parameter block for an operation, function, or global ProcScript.

```
params
  DataType ParamName : Direction
  {DataType} Field.Entity{.Model} : Direction
  {DataType} $ComponentVariable$ : Direction
  {byRef} | {byVal} struct ParamName : Direction
  entity Entity{.Model} : Direction
  occurrence Entity{.Model} : Direction
  xmlstream [DTD:DTDName]ParamName : Direction
endparams
```

Parameters

Table: Parameters

Parameter	Data Type	Description
DataType	Literal	Uniface data type. For more information, see Uniface Data Types. If the data type is one of the following, additional qualifiers and or parameter naming rules apply: • `struct` • `entity` • `occurrence` • `xmlstream`
ParamName	Literal	String has a maximum length of 32 bytes, including letters (A-Z), digits (0-9), and underscores (_); the first character must be a letter.
Direction	Literal	Direction for the parameter with respect to the operation; one of `IN` (input only), `OUT` (output only), or `INOUT` (input and output). If a parameter is defined as `OUT`, its value at the start of the operation cannot be predicted.
$ComponentVariable$	Literal	Name of a component variable that is defined in the component.
`byRef`	Literal	The Struct data is passed by reference, so only a memory pointer is passed, not the actual data.
`byVal`	Literal	The Struct data is passed by Value, so it is copied before being passed.
Field	Literal	Name of a Component Field.
Entity	String	Name of a Component Entity (either database or non-database).

Parameter	Data Type	Description
		On both the `activate` statement for this operation and in the `params` block, the corresponding `entity` parameters must represent the same entity, or one or its subtypes.
Model	Literal	Name of the model to which the entity belongs.
DTDName	String	Name of the DTD that defines the structure of XML stream variables. *DTDName* has the format: *DTDName*{`.` *ModelName*} *DTDName* is the literal DTD name defined in *ModelName*.

Return Values

None

Use

Allowed in all component types.

Also allowed in global ProcScript modules. The maximum size of a parameter can be 10 MB.

Description

The `params` statement defines the formal parameters for an operation, an function (either a global or local ProcScript module), or a global ProcScript. A maximum of 64 parameters can be defined. When the operation or function is referenced in an `activate` or `call` statement, the arguments provided on that statement must match this list of parameters in number and in type.

The parameter block defined by `params` can occur in any of the following places:

- The first statement following an `operation` statement
- The first statement in the exec operation in all components except dynamic server pages
- The first statement following a `function` statement in a function module
- The first statement of a global ProcScript module

If no parameters are required, the `params` block is not required. If a `variables` block is present, it should immediately follow the `params` block, or the `operation` statement, if no parameters are defined.

Struct Parameters

The content of a `struct` parameter is not a Struct, but zero or more references to Struct nodes, just as it is for `struct` variables. This is in contrast to scalar data types such as `string`. References are memory pointers so it is only possible to pass them between ProcScript modules that share the same memory. This is always the case for partner operations and functions, but may not be the case for public operations.

For this reason, the default behavior is to pass Struct parameters by reference in partner operations and functions, and by value in public operations. When passed by reference, the parameter passes a copy of the reference to the called function or operation, so an additional reference to the same struct is created. When passed by value (the default for public operations), a copy of the Struct is made and this can have consequences when the data is returned.

It is possible to override the default behavior by explicitly defining how Struct parameters are passed using the `byRef` or `byVal` qualifiers. This can be useful if you know that the calling and receiving component instances are running in the same process.

If neither the `byRef` nor the `byVal` qualifiers are used, the default behavior depends on whether the parameter is declared in a public operation or not. However, because of the functional difference between the two, it is advisable to explicitly specify the intended qualifier.

If an operation with a `byref` struct parameter is activated in a component running in a different process, this will result in the following error:

```
-73 <ACTERR_REMOTE_NOT_SUPPORTED> "Operation with byref-Struct parameter cannot be activated across processes"
```

The way that Structs are passed has implications for the way in which the data is synchronized. For more information, see Passing Struct Parameters.

Entity and Occurrence Parameters

`entity` and `occurrence` parameters are known as constructed parameters. An entity parameter transfers all occurrences of the specified entity from one component to the other component. An occurrence parameter transfers the current occurrence of the specified entity from one component to the other component.

> **Note:** Only fields whose **Is External** property is `T` can be included in the entity parameter or occurrence parameter. By default, this is true for DBMS entities.

On each call to the operation, an implicit `creocc` is performed to set aside sufficient memory for the occurrence parameter. The call does not release the memory implicitly, but the memory can be cleared by clearing the data in the component entity that was created by the call.

> **Note:** Constructed parameters cannot be used in entries and functions (global or local ProcScript modules).

- If the entity parameter has direction `IN` or `INOUT`, when the operation starts, an implicit `clear/e` statement for that entity is performed in the component instance before the occurrences are filled with the data being transferred.
- If the entity parameter has direction `OUT` or `INOUT`, when the operation returns, an implicit `clear/e` statement for that entity is performed in the requesting component instance before the occurrences are filled with the data being returned. This is followed by a `release/e/mod` statement for the entity.
- If the occurrence parameter has direction `IN` or `INOUT`, when an operation starts, the current occurrence of the requesting component becomes the new occurrence in the requested component (such as an entity service).
- If the occurrence parameter has direction `OUT` or `INOUT`, when the operation returns, the current occurrence of the requested component becomes the new occurrence in the requesting component.

> **Note:** If a component is activated asynchronously (for example, with the `/async` switch on the `newinstance` statement), you should not activate an operation that contains a parameter with the direction `OUT` or `INOUT`.

XML Stream Parameters

`xmlstream` parameters contain XML data, which can be transferred between the parameter and the component's external data structure by ProcScript statements, using `xmlsave` and `xmlload`. For more information, see xmlsave and xmlload.

Scope of Parameters

An entity, occurrence, field, or component variable parameter is defined at the component level. The values of these parameters are component-wide in scope; that is, the values are available to all operations and modules in the component.

A named parameter exists only in the operation or module in which it is defined. Its scope is limited to that operation or module. It cannot be directly referenced from another operation or module in the component, or from a local or global ProcScript called by the operation or module. If a named parameter has the same name as a field defined in the component, the parameter takes precedence over the field; to access the field, you must use the qualified field name.

For example, consider a component that contains a field named DATE in the entity PO. The operation TODAY has a named parameter, DATE. To update the field DATE in the operation TODAY, the field must be referenced by its qualified name, including its entity:

```
operation TODAY
params
 date DATE : OUT
endparams
DATE = $date ;assign the current date to parameter DATE
DATE.PO = $date ;assign the current date to field DATE.PO
end ; operation TODAY
```

`xmlstream` parameters have the same scope as named parameters. However, the data in an XML stream is not accessible to Uniface until the data has been loaded into the component's external data structure, which is accessible by all ProcScript modules in the component.

`struct` parameters have the same scope as named parameters. Unlike other named parameters, which contain data, a `struct` parameter contains only a reference to a Struct. The Struct itself is unaffected by the scope of the `struct` parameter. Thus, when the operation or function completes, the reference to the Struct no longer exists, but the Struct itself remains (as long as any variable refers to it).

Example: Defining an Operation

The following example shows the operation `DISCOUNT` of a service component named SERV1:

```
operation DISCOUNT
params
  string CUSTID : IN
  numeric AMOUNT : INOUT
  numeric PERCENTAGE : OUT
```

```
endparams
; no discount till proven otherwise
; 20% discount for Uniface
; 15% discount for Acme
; adjust amount
PERCENTAGE = 0
if ( CUSTID == "ufbv" ) PERCENTAGE = 20
if ( CUSTID == "acme" ) PERCENTAGE = 15
AMOUNT = AMOUNT * ( 100 - PERCENTAGE) / 100
end
```

The operation `DISCOUNT` could be referenced from another component as follows:

```
activate "SERV1".DISCOUNT (ID.CUST, TOTAL.INVOICE, $DISCOUNT$)
```

Related concepts

- ProcScript: Data Types
- occurrence
- entity
- entry
- operation
- variables...endvariables
- $subsetreturn