PROCEDURE

PROCEDURE action

Declaration

```
 PROCEDURE ProcName [([IN] type1 paramName1[,paramName2, ...] [IN] type2
paramName3]...)]

 ;actions

 END ProcName
```

Procedure declaration for a remote call

```
 [IMPLEMENTATION] RPC PROCEDURE [ESLInterface^]ProcName [([IN] type1
paramName1[,paramName2, ...] [IN] type2 paramName3]...)]

 ;actions

  END ProcName
```

[IMPLEMENTATION] RPCX PROCEDURE [ESLInterface^]ProcName **[(([IN]** type1 paramName1**[,**paramName2**, ...] [IN]** type2 paramName3**]**...)**]**

; actions

END ProcName

Declaration for calling to UNIT

```
PUBLIC PROCEDURE ProcName [([IN] type1 paramName1[,paramName2, ...]   [IN]
type2 paramName3]...)]

 ; actions

 END ProcName
```

Parameters

procname	in	Procedure name (must meet the rules for object name).
type1, type2, ..., type10	in	Type of the first (second, third, ..., tenth) formal parameter.
paramName1, paramName2, ..., paramName10	in	Name of the first (second, third, ..., tenth) formal parameter.

Description

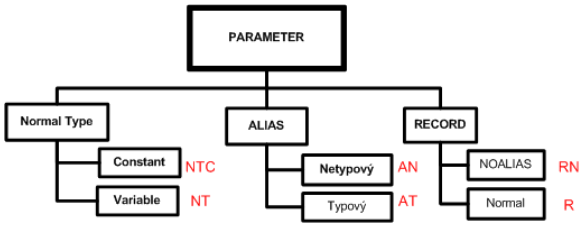
The action introduces the procedure (procedure header) with the name *ProcName*. Procedure name must be unique within the frame of the particular script. Procedure header may be placed before the script initialization part out of other procedure (embedded procedures are not supported).

Number of parameters in procedure is not limited.

Each parameter in the procedure body presents a local variable (the type and name are written in the declaration). If the key word **IN** is written before the type in the declaration of the parameter, the local variable is accepted as input variable (contingent changes of its value do not influence the value of parameter after the procedure termination). Parameters with no key word **IN** are input-output ones.

Procedure must be finished by the action **END** ProcName.

Possible types of procedure formal parameters:



Examples of individual types of formal parameters:

Formal parameter type	Parameter description	Example
NTC	Constant of INT, BOOL, ..., types is not admissible as a parameter.	
NT	Variable of INT, BOOL, REAL, TIME, TEXT type	INT _paramInt
AN	Untype ALIAS	ALIAS _an
AT	Type ALIAS	ALIAS (SD.RecordDef) _at
RN	Structure with no reference to objects	RECORD NOALIAS (SD.RecordDef) _rn
R	Structure	RECORD (SD.RecordDef) _rn

Using the action CALL for the calling of the procedure, a real parameter will be substituted into the formal parameter. For the formal parameter of NT type, the type conversion is executed among real > formal > real parameter. The following combinations of real parameters are admissible for individual types of formal parameters:

Parameter category	Formal parameter				
	NT	AN	AT	RN	R
IC_C	IN	-	-	-	-
IC_HBJ_EXPR	IN	IN	-	-	-
IC_L		-	-	-	-
IC_L_CONST	IN	-	-	-	-
IC_L_AN			-	-	-
IC_L_AT_RIA			-	-	-
IC_L_AT_RIN		-	-	-	-
IC_L_AT_R	-	-	-		
IC_L_AT	-	IN		-	-
IC_L_R_RIA			-	-	-
IC_L_R_RIN		-	-	-	-
IC_L_R_R	-	-	-		
IC_L_R	-	-	-		
IC_L_RNA_RIA		-	-	-	-
IC_L_RNA_RIN		-	-	-	-
IC_L_RNA_R	-	-	-		-
IC_L_RNA	-	-	-		-
IC_O		IN	-	-	-
IC_O_R_RIA			-	-	-
IC_O_R_RIN		-	-	-	-
IC_O_R_R	-	-	-		
IC_O_R	-	-	-		
IC_L_NR_R	-	-	-		
IC_L_NR	-	-	-		
IC_L_NRNA_R	-	-	-		-
IC_L_NRNA	-	-	-		-

- Green box represents admissible combination of formal and real parameter.
- Box with the text IN specifies the need to declare formal parameter as input one (key word IN).

Operation with the real and formal parameter for calling and return from the procedure:

Formal parameter type	Real parameter		Formal parameter
	Call	Return*	Call
NT	Value reading.	Value setting.	Setting the value according to the parameter.
AN	Reading the reference to object (not object value).	Setting the reference to object (not value).	Setting the reference to object according to the parameter, that is also the reference.
AT	Reading the reference to object (not object value).	Setting the reference to object (not value).	Setting the reference to object according to the parameter, that is also the reference.

RN	Value reading (row or the whole structure).	REDIM for the parameter as necessary (REDIM is executed if index = 0 and real parameter is the local variable) and value setting.	REDIM of the as necessary and value setting.
R	Value reading (row or the whole structure).	REDIM for the parameter as necessary REDIM is executed if index = 0 and real parameter is the local variable) and value setting.	REDIM of the as necessary and value setting.

* only in case of the parameter IN OUT

A procedure declared with the key word **RPC** may be called from other SL scripts using the remote procedure call. The types of formal parameters for RPC procedures are **NT** and **RN**.

Executing the server event (or script) is performed in one instance. Therefore simultaneous calling RPC procedure by several another events is serialized and requests are executed in sequence by means of a request queue. If a remote procedure is declared by the keyword **RPCX**, then all requests to execute the procedure are removed after occurring the new request. The result behaviour is, that there are not two current requests to execute the procedure declared by using the keyword **RPCX**. From the side of caller, request removing is the termination of the remote call with an error.

The key word IMPLEMENTATION is used when the procedure implements some procedure from object ESL Interface. The name of proper object is stated as prefix of procedure name which is separated by character ^.

The key word **PUBLIC** is used when the procedure is a part of UNIT event. This type of procedure may be called only from the script in which the given UNIT is defined (declared) (Public procedure call).

RPC procedures allow transferring data containers and handle to the database connections.

Formal parameter of procedure - NON-TYPED RECORD

When declaring the formal parameter of RECORD procedure, if you do not define the structure type (RECORD NOALIAS () _rec), there occurs the **non-typed record**. The real parameter that inputs to the procedure by CALL action when calling the procedure, defines the type to non-typed record.
The structure definition HBJ of non-typed record can be determined by ESL function % GetRecordStructHBJ (IN recordVal) (_hbj := % GetRecordStructHBJ(_rec\HBJ)).

Example

```
PROCEDURE Proc1(RECORD NOALIAS () _arr)
INT _iHbj

_iHbj := %GetRecordStructHBJ(_arr\HBJ)
; ....
; ....
; ....
REDIM _arr[ _arr\DIM + 1 ]
END

BEGIN
RECORD NOALIAS (SD.RecordDef) _arr1
; ....
; setting the value of local variable _arr1 according to the object value
; SV.Struktura
REDIM _arr1[SV.Struktura\DIM]
SET _arr1 WITH SV.Struktura
CALL Proc1(_arr1)

END
```

Procedure declaration and its calling with two input parameters and one output parameter:

```
PROCEDURE Adding(IN  INT _p1, IN  INT _p2, INT _v)
   _v := _p1 + _p2
```

```
 _p2:= _p2 + _p2
END Adding

BEGIN
  INT _a = 3
  INT _b = 2
  INT _c
  CALL Adding(_a, _b, _c)
END
```

The procedure add two input parameters of **INT** type *_p1* and *_p2* together. The result is saved into the input-output parameter *_v* of **INT** type. After the return from the procedure, the parameter *_cc* is set to the sum of values of the parameters *_a* and *_b*. The value of the parameter *_a* is not changed by the action (_p2 := _p2 + _p2), because *_p2* is the input parameter and its changes are not to be published.

Procedures may be called recursively (procedure from procedure).

The declaration of procedure with the parameter of Structure type:

```
PROCEDURE Proc1(RECORD  NOALIAS (SD.RecordDef) _arr)
 ; ....
 ; ....
 ; ....
 ; ....
   REDIM _arr[_arr+1]
 END Proc1

 BEGIN
  RECORD  NOALIAS (SD.RecordDef) _arr1
  ; ....
  ; Setting the value of the local variable arr1 according
  ; to the object value of  SV.Structure
  REDIM _arr1[SV.Structure\DIM]
  SET _arr1 WITH SV.Structure
  CALL Proc1(_arr1)
 END
```

In the example, the calling CALL is performed in the following steps:

1. The evaluation of the real parameter row index (in this case 0 => the whole object value).
2. Value reading of the real parameter _arr1.
3. Change of the formal parameter size as necessary: REDIM _arr[_arr1\DIM].
4. Saving into the formal parameter: SET _arr WITH _arr1.

Return from the procedure:

1. Value reading of the formal parameter _arr.
2. The evaluation of the real parameter row index (in this case 0 => the whole object value).
3. Change of the formal parameter size as necessary: REDIM _arr1[_arr\DIM].
4. Saving into the formal parameter: SET _arr1 WITH _arr.

For calling, if no index is written besides the real parameter (structured variable or object of Structure type), the whole value is proceeded. This is also valid, if index is stated that gets value = 0 (no access to item).
Exception is the real parameter of **IC_L_AT** type. This is, with no index and access to the item, accepted as the reference to object of Structured variable type, not as its value.

Note

Standard procedure termination is executed by the execution of the action **RETURN, END ProcedureName** or aborted by the action **END** or an error occurrence which handling is not defined in the current procedure (**ON ERROR**). Input-output parameters in the particular action CALL are updated during the standard procedure termination.

If procedure is called as the remote one and is not terminated in normal way, contingent input-output parameters may not be updated. Therefore if this procedure is called synchronously with the assignment, the result of the assignment is the error **ERR_MISSING_RETURN**.

Related pages:

Script actions