

# ESL Unit Event


**ESL UNIT** is an object of [EVENT](#) type. It enables to configure the universal scripts with the procedures and variables and use them in ESL scripts (as the libraries). There are available only those procedures which are defined by the enumerated word PUBLIC. You may use all available ESL [actions](#) and [functions](#) in UNIT. The only difference from the [ESL script](#) is that there cannot be defined the RPC procedures and [ESL Interface](#) in UNIT.

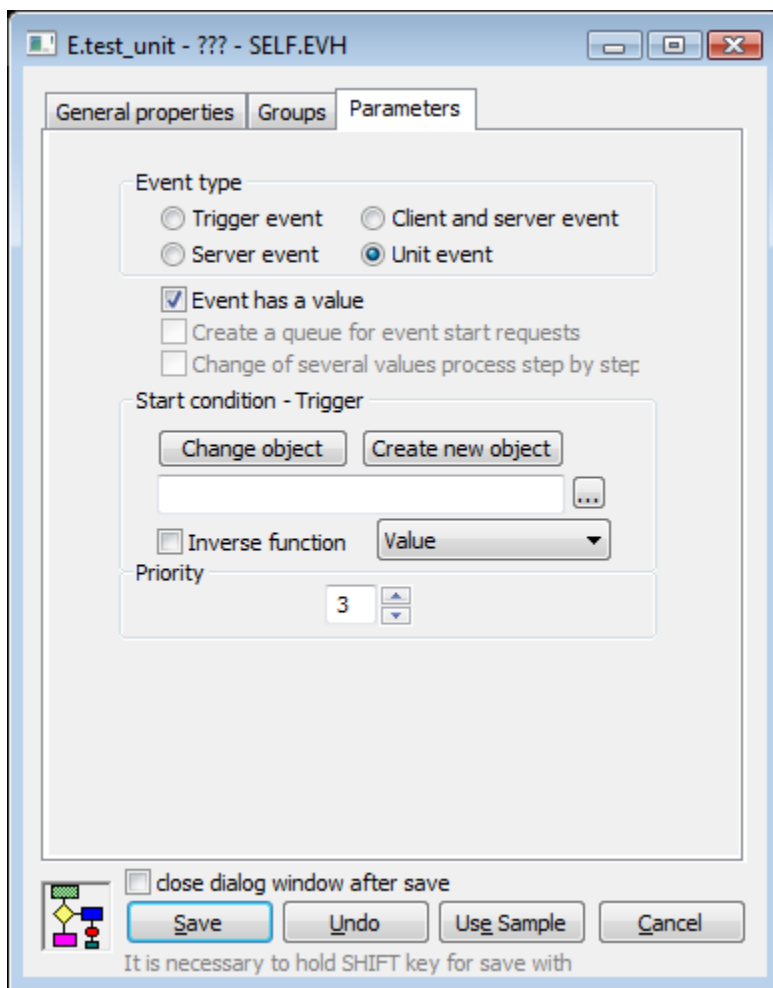
Each UNIT declaration in the script causes that there is created the new object with own local variables and procedures in the system. This is the main difference from `#include` in C, where `#include` inserts whole text of the library, which contains the declaration of functions, to the script. This feature enables to define more UNIT of the same type in the script. Each UNIT may have the different values of its local variables (each UNIT has its own life cycle).

The life cycle of UNIT is limited by the duration of life cycle of script, which is declared and used by the given UNIT, in the system. The life cycle of UNIT begins its declaration in the script and ends when the script, in which UNIT had been declared, is terminated in the system.

The addressing of Public procedure of UNIT means the calling of the procedure of particular UNIT by the local variable, which represents it. The procedures can be called only from the script that declares the given UNIT.

## Configuration

The UNIT is configured in the [configuration dialog box](#) in **D2000 CNF**. Create UNIT as a new object of EVENT type. When creating, [the script editor](#) is opened automatically. There, click the  **Parameters** button to open the configuration dialog box. Select the option **Unit event** on the Parameters tab.



A parent of UNIT can be any process of \*.EVH. UNIT event is not activated when starting the process. The configuration of script in ESL UNIT is the same as in other types of events. There are used all properties of ESL language.

Exceptions:

- It contains the key word PUBLIC.
- The RPC procedures and ESL Interface may not be defined.

The initial part of the script (between BEGIN - END) is always executed. The actions in the UNIT script are executed in the context of an instance of ESL script, in which the UNIT is declared. UNIT event does not allow a debugging. The debugging procedure is described in the section [Debugging the UNIT scripts](#).

Example:

```
;*****
; DESCRIPT: Unit1 - Counter
;
;
; AUTHOR: Programmer
; LAST CHANGE:
;*****

INT _iLocal
;
PUBLIC PROCEDURE Make
    _iLocal := _iLocal + 1
END Make

PUBLIC PROCEDURE GetValue(INT _iValue)
    _iValue := _iLocal
END GetValue

BEGIN
    _iLocal := 0
END
```

Declaration and usage of UNIT

```
UNIT (Name_of_unit_event) _unit1
```

UNIT	Type of local variable.
(Name_of_unit_event)	Type of UNIT event, for example E.unitCounter.
_unit1	Name of declared local variable that represents the given UNIT.

The declaration of a local variable of UNIT type must always be at the beginning of ESL script. When starting the script with UNIT declarations, the initial parts of these UNITS are executed in the order as they are declared. If the declaration of the local variables contains more UNITS of the same type, the initial part of all ones will be executed. There is supported the declaration of UNIT in UNIT, which uses the features of UNIT in other UNIT. For this feature, there is supported a detection of UNIT deadlock.

The calling of Public procedures is done by [CALL](#) action. After this action, there must be an addressee (\_unit1), which contains the given PUBLIC procedure.

```
CALL [_unit1] Make
CALL [_unit1] GetValue(_value)
```

CALL	CALL action.
[_unit1]	The addressing of required UNIT.
GetValue(_value)	Name of procedure + parameters.

The operation is done by the switching the context. Then the called procedure is done and the context is switched back (the executing of the original script).

**Note:** The addressee [\_unit1] must always be the name of local variable that has been used in UNIT declaration.

Example:

```

;*****
; DESCRIPT: Unit Caller
;
;
; AUTHOR: Programmer
; LAST CHANGE:
;*****

UNIT (E.Unit1) _unit1
UNIT (E.Unit1) _unit12
UNIT (E.Unit2) _unit2

RPC PROCEDURE CheckValue(BOOL _bOk)
INT _iValue
INT _iValue2

CALL [_unit1] GetValue(_iValue)
CALL [_unit12] GetValue(_iValue2)

_bOk := _iValue # _iValue2

END CheckValue

BEGIN

CALL [_unit1] Make
CALL [_unit12] Make
CALL [_unit12] Make

END

```

As each declared UNIT has own life cycle, this ensures that you may declare 1 up to n UNITS of the same type in one script (e.g. *\_unit1* and *\_unit12* are of the same type *E.Unit1*). It means, the UNITS of the same type may have the different values of the local variables after the different operations.

## Debugging the UNIT scripts

ESL Unit is debugged together with the script in which it is declared.

If you use the debugging [Step Into](#) (F8 - Stepping with the nesting), the tab with UNIT script is automatically opened in ESL editor and the current executed line is set. When debugging the script that contains the declarations of UNITS of the same type, ESL editor opens as many tabs with UNIT scripts as necessary.

After finishing the UNIT script, the tab is switched back to the script, which called the UNIT. After debugging the script, ESL editor automatically closes all the UNIT tabs, which were used.



### Related pages:

- [Events](#)
- [Start conditions](#)
- [Events - configuration dialog box](#)