Local Variables (Event Script Language (ESL))

Script local variables

Using local variables allows to effectively implement loops, subprograms and other control algorithms.

The identifier of a local variable is a sequence of characters, which can be used for the name of the D2000 system (rules for object name definition). It must begin with the character "_" (underscore). For example: _i, _i1, _locVar.1. Maximum: 64 characters. The value of a local variable is characterized by all value attributes as well as objects in the D2000 system (time, limits, process alarm etc.).

Available types of local variables:

- **INT** - integer variable within the range of -2147483647...2147483648
- **BOOL** - logical variable (Boolean)
- **REAL** - real variable within the range of -1.7E+308...`
- **TIME** - variable of Absolute time type
- **TEXT** - variable of Text type
- **ALIAS** - reference to a D2000 system object
- **RECORD** - structured variable

Note

The SD.RecordDef and SV.Structure objects are used for a demonstration of some features of local variables.

A local variable definition is denoted by its type (in the case of RECORD and ALIAS type, by the type of the structure), which is followed by the local variable identifiers of that type, separated with commas, e.g.:
INT _a, _b, _c,
or
RECORD NOALIAS (SD.RecordDef) _rec1, _rec2.

TIME represents also a keyword in the assignment.

Variables of **INT**, **BOOL**, **REAL**, **TIME** and **TEXT** types represent simple values of a given type.

Variables of **TEXT** type are internally encoded in UTF-8.

A variable of **ALIAS** type represents a reference to a D2000 system object. After running the script, the reference is empty and its value is not defined. It is not possible to use such a variable and the error *_ERR_NO_ASSIGNED_ALIAS*, which causes interruption of the script execution, will be generated if you attempt to do it. After initialization of such a variable (setting the reference to an object), it acquires the value of the linked object by means of a particular action (SET AS). It is possible to use such a variable in expressions or to assign a value to it in the same way as if a linked object was used.

Example:

```
ALIAS _obj
INT _i
_obj := 1     ; action will cause ERR_NO_ASSIGNED_ALIAS error
```

```
ALIAS _obj
INT _i
SET _obj AS U.Int
_obj := 1    ; assigns the value of 1 the user variable U.Int
WAIT         ; wait for execution of the assignment
_i := _obj   ; assigns the value of U.Int to the local variable _i
```

If an *ALIAS* type local variable is linked with an object of Value array type, it allows using indexing in the same way as for the object of Value array type:

Example:

```
ALIAS _obj
INT _i
SET _obj AS X.Array
_i := _obj[2]   ; assigns the value of the second item of X.array to local variable _i
```

There are two different types of *ALIAS* type local variables:

- *untyped ALIAS*. The type is mentioned above.
- *typed ALIAS*. The reference to an object of Structured variable type.

When you declare a typed ALIAS it is necessary to specify a type (in our case, it is an object of Structure definition type), which limits the set of possible objects (of Structured variable type), to which the local variable (typed ALIAS) may be linked. For a local variable declared in this way, it is able, in expressions (of course, after an association using SET AS action), to use indexing and to access items in the same way as for the object of Structured variable type.

Example:

```
ALIAS (SD.RecordDef) _recAlias
INT _i

_i := 2
ALIAS _recAlias AS SV.Structure
_recAlias[_i]^Int := 1
```

The variable of **RECORD** type allows creating a local variable of *Record* type. The structure type is given by the object of the Structure definition type, which is declared. If the RECORD variable is a formal parameter, its structure type - non-typed RECORD need not be set.

Example:

```
RECORD (SD.RecordDef) _recLocal
```

After running the script, the array size of such a local variable defined in this way is one (1) and the values of all the items are invalid.

Example: Assigning a value

```
RECORD (SD.RecordDef) _recLocal
_recLocal[1]^Text := "text value"
_recLocal[1]^Int := 8
```

Change of the array size of a local variable may be executed by using the REDIM action.

Example: Change the array size to ten (10) items

```
REDIM _recLocal [10]
```

The minimum array size is zero (0). An array size change doesn't lose values of items contained in the array before the action. Potential new items created are initialized to an invalid value.

An item of a local variable of *Record* type (or object of Structured variable type) may be *Object* type (see the topic Structure definition). Such an item is interpreted in the script as an *untyped* ALIAS. Consequence:

Assignment:

```
_recLocal[1]^Object := 1
```

may cause the error *_ERR_NO_ASSIGNED_ALIAS* if an object is not linked to an item. An assignment of an object to an item is identical with the assignment of variables of ALIAS type:

```
SET _recLocal[1]^Object AS U.Int
```

Note: Analogous to untyped ALIAS, the access to an item of a local variable that is not linked with will generate the error *_ERR_NO_ASSIGNED_ALIAS*.

To avoid this behaviour use the keyword NOALIAS when you declare the local variable.

```
RECORD NOALIAS (SD.RecordDef) _recLocal
```

For a local variable declared this way, all items of an *Object* type are interpreted as standard values of any type (INT, BOOL, REAL, TEXT, TIME). Therefore the following declaration is correct:

```
_recLocal[1]^Object := 1
_recLocal[1]^Object := "TEXT"
```

The item *Object* changes its own type together with individual assignments.

After assigning a value to an *Object* type item of a Structured variable type object, this assignment is automatically redirected to the linked object (likewise untyped ALIAS). If the item is not linked to an object, an error will occur only if the action WAIT follows.

Note

- The term *Structure* described below represents the value of a local variable of *Record* type or the value of a Structured variable type object.

ESL allows assigning either a whole structure row or a whole structure at once. This assignment is executed by SET WITH action.

Action syntax:

```
SET identDst_Rec WITH identSrc_Rec
```

where

- *identSrc_Rec* - represents the source of values,
- *identDst_Rec* - represents the destination of values.

Both identifiers must represent a structure row or a whole *structure*. Structure types must be the same (the error: *Incompatible types*). In the case of a whole structure, their number of rows must be identical (the error: *_ERR_RANGE_ERROR*). The action will assign all the values from the source (identSrc_Rec) to the destination (identDst_Rec).

!!! IT DOES NOT CHANGE THE ASSOCIATED (LINKED) OBJECTS IN INDIVIDUAL ITEMS !!!.

SET AS action permits to assign and change linked objects (for items of *Object* type) in a whole structure row, or in a whole structure together.

Action syntax:

SET identDst_Rec AS identSrc

The same limits as listed for the action SET WITH are also available for *identSrc* and *identDst*, but they may not be local variables of *Record* type declared as NOALIAS.

Permitted combinations of identSrc and identDst parameters for SET WITH a SET AS actions.

SET BIND action links a **row** of a local variable of *Record* type to a row of an object of Structured variable type.

Action syntax:

SET _recLocal_Row BIND struct_Row

where

- *_recLocal_Row* - represents the reference to a row of a local variable of *Record* type
- *struct_Row* - represents a row of objects of Structured variable type

Example:

SET _recLocal[2] BIND SV.Structure[1]

After the action execution, the values of all the items of row 2 of the local variable (_recLocal[2]) are the same as the values of row 1 of the object *SV.Structure*. For example, the expression _recLocal[2]^Int returns the same value as the expression SV.Structure[1]^Int.

The feature of row binding is also allowed for other actions:

Example: Assigning a value

_recLocal[2]^Int := 1

will assign the value of 1 to the item SV.Structure[1]^Int. Since the value of the item _recLocal[2]^Int copies the value of the item SV.Structure[1]^Int, any change of the local variable is executed after a change in SV... object (it is possible to use WAIT action). The behaviour of rows of a local variable *Record* type, which is not linked (SET BIND) to another row is unchanged (described above).

Each row of the local variable *_recLocal* may be linked to another (or the same) row of an object of Structured variable type. The only limitation is, that the local variables and objects must be of the same structured type (structure definition).

Unlinking a row (cancelling the action SET BIND) is allowed by the action:

SET _recLocal[2] BIND NONE

All values of items will acquire an undefined state.

Related pages:

Script actions