Script Actions

Script actions

Every ESL script consists of a sequence of actions executed when the script is activated. The actions are defined in the script editor environment.
Parts of action declarations enclosed in square brackets **[ ]** are optional. The symbol **|** (i.e. OR) expresses a possibility of an alternative notation.

Action types

The list of the actions types. The following list describes the basic set of actions, which are available in both script applications (object of Event type, Active picture).
Actions may be divided into the following basic categories (types):

- assignment actions
- action for accessing a database
- actions for handling error states
- control actions
- actions for communication with operator
- archive manipulation actions
- actions for synchronisation of script actions execution
- controlling alarms
- structure manipulation actions
- data container manipulation actions
- access right manipulation actions
- actions for manipulation with list of object
- other actions

Note: D2000 System allows to use JAVA language to write the applications. The equivalents of ESL actions are stated in the extra html files available in the D2000 System installation directory - subdirectory **Help**.

Assignment actions

Assignment actions allow to change values and references of :

1. objects in the system
2. local variables

- Assignment
- SET WITH
- SET AS [DIRECT]
- SET BIND

Actions for accessing a database

Groups of actions, which use various methods for accessing data, are intended for working with a table of a database.
If an error occurs while working with a database, the numerical code that describes it closely may be acquired by calling the function %GetLastExtErrorCode. More detailed information about the error may be acquired by %GetLastExtErrorMsg function.
The first method uses the existence of a key or a WHERE condition. According to the way of action notation, it allows to write / read one or several lines.

- DB_CONNECT
- DB_DELETE
- DB_DISCONNECT
- DB_INSERT
- DB_INSUPD
- DB_READ
- DB_READ_BLOB
- DB_SET_PROCESS_PARAMS
- DB_UPDATE
- DB_UPDATE_BLOB

Example: work with a database table (actions DB_...)

The second method uses paging, where the page size (line number in database table) is optional.

- PG_CONNECT
- PG_DISCONNECT
- PG_READ
- PG_INSERT
- PG_DELETE
- PG_UPDATE

Example: work with a database table (actions PG_ ...)

The third method is a modification of the first one. The modification is based on the fact, that work with a database table does not require the CONNECT and DISCONNECT actions.

- DBS_DELETE
- DBS_INSERT
- DBS_INSUPD
- DBS_READ
- DBS_READ_BLOB
- DBS_UPDATE
- DBS_UPDATE_BLOB

The last method allows to use the whole scale of **SQL** commands to work with a database.

- SQL_CONNECT
- SQL_DISCONNECT
- SQL_EXEC_DIRECT
- SQL_EXEC_PROC
- SQL_SELECT

Reading a database by means of the command **SELECT**

- SQL_PREPARE
- SQL_BINDIN
- SQL_FETCH
- SQL_FREE

Example: work with a database (actions SQL_ ...)

To control transaction command processing (in the sense of database transaction), you can use the following actions:

- DB_TRANS_OPEN
- DB_TRANS_COMMIT
- DB_TRANS_ROLLBACK
- DB_TRANS_CLOSE

Example: database transactions

To force refreshing displayed data in user's views in process D2000 HI (e.g. displayer of Browser type) use the action:

- DB_REFRESH_TABLE

To register a procedure which will be called after

- change of data in the table
- deleting of table
- switching from active to passive instance of the process DbManager
- refreshing of displayed data via action DB_REFRESH_TABLE

use the action:

- ON DB_CHANGE

Transfer of handle to database connection between the running ESL scripts

Conversion and representation of values in database

Writing

When writing the value of variable (in most of situations) of local structured variable array into database, it may be converted in two ways: if it is **valid**, it is written in a standard way. If **invalid**, NULL value is written into database. For **TEXT** type, this rule works in the same way, except for ORACLE, in which an empty text is represented by NULL value (as the invalid value).

Reading

After reading the value (which is different from NULL) from database, it will be converted to the required type, which is defined by a value type to which the result of reading is stored. If the conversion is done successfully, the resultant value is valid. When reading the NULL value, the resultant value is invalid. For **TEXT** type, NULL value in database is converted to a valid empty text string. The only exception is the reading by DB_READ/DBS_READ script action on ORACLE OCI platform, when NULL value is converted to **invalid** one.

The table below illustrates the result of writing and reading of text value depending on the database platform.

DBS_INSERT - writing a text value into database (D2Value -> DBValue).
PG_READ, BrowserRead - reading the text value from database by PG_READ or into the Browser displayer (data displayed through OnFetchDone) (DBValue -> D2Value).
DB_READ - reading the text value by DB_READ (DBValue -> D2Value).

Database	DBS_INSERT	PG_READ, BrowserRead	DB_READ
Sybase 12/PostgreSQL	"Text" -> "Text	"Text" -> "Text	"Text" -> "Text
dbmanager.exe	"" -> ""	"" -> ""	"" -> ""
	Invalid->NULL	NULL->""	NULL->""
ORACLE OCI	"Text" -> "Text	"Text" -> "Text	"Text" -> "Text
dbmanager_ora.exe	"" -> NULL		
	Invalid->NULL	NULL->""	NULL->Invalid
ORACLE ODBC	"Text" -> "Text	"Text" -> "Text	"Text" -> "Text
dbmanager.exe	"" -> NULL		
	Invalid->NULL	NULL->""	NULL->""

Actions for handling error states

- EXCEPTION_HANDLER
- ON ERROR
- RETRY
- RESUME

Control actions

Control actions are the actions, which manipulate the control flow (actions execution order) in the script.

- BEGIN
- CALL - local procedure call
- CALL - public procedure call
- CALL - remote procedure call
- DELAY
- DO_LOOP, EXIT_LOOP, END_LOOP
- ENABLE
- END
- END procedure
- EVENT
- GOSUB
- GOTO
- IF GOTO
- IF THEN [ELSE] ENDIF
- IMPLEMENTATION
- ON DB_CHANGE
- ON GOTO
- ON CHANGE
- OnExternalEvent
- PRAGMA
- PROCEDURE
- RETURN
- Control functions (with a void return value)
- WAIT

Actions for communication with operator

The following actions allow to implement a dialog with operator, or insert pictures from operator console. It is advisable to use the predefined local variable `_FROM_HIP`. If the script is started from picture (graphic object connected to control), the local variable is automatically linked to process, where the script was started from. This allows to address work with this process:

- open, close pictures,
- send messages for operator,
- route QUERY action.

- MESSAGE
- QUERY
- OPEN
- OPENEVENT
- CLOSE

Actions for synchronisation of script actions execution

The GETACCESS ad RELEASEACCESS actions allow to mutually synchronize the execution of actions in

- various instances of events within one process D2000 Event Handler
- various scripts of active pictures within one process D2000 HI
- various scripts or globally in event instances or scripts of active pictures in the application (for D2000 Entis only !!!)

They provide a specified form of communication among scripts.

- GETACCESS
- RELEASEACCESS

Archive manipulation actions

- CALCARCHEXPR
  CALCARCHEXPR
- CALCARCHEXPRCALCONDEMANDSTAT
- CALCSTATFUNC
- CALCSTATFUNCARR
- DELETEARCHDATA
- GETARCHARR
- GETARCHARR_TO_CNT
- GETARCHCOL
- GETARCHROW
- GETARCHSTRUCT
- GETARCHVAL
- INSERTARCHARR
- UPDATEARCHVAL

Controlling alarms

Controlling of system or process alarms.

- BLOCK
- KVIT
- UNBLOCK
- UNBLOCK_ALL

Structure manipulation actions

Working a with large-scale local structures sometimes requires to classify a structure, insert or delete a row, or find a row. ESL permits such operations, but they require the iteration of a particular structure in a loop. It is a time consuming task. ESL therefore defines the following actions, which execute the described operations more effectively:

- COPYCOL
- COPYCOLIDX
- DELETE
- EXPORT_CSV
- FIND_TRUE
- GETCOLTIME
- GETROWDESC
- IMPORT_CSV
- INSERT
- SETCOLTIME
- SORT
- TRANSCOLTOROW
- TRANSROWTOCOL

Data container manipulation actions

These actions allow to work with data storage, so-called *data container* (internal data structure). The owner of a container is always one running instance of script. Data container can be shared between various scripts and processes.
Container has its unique identifier. Container is automatically terminated by terminating the script, to which it belongs, or by executing the action **CNT_DES TROY**.
The size of a container is not specified and is only limited by the operating memory size.

Each value included in container is uniquely determined by so-called *key*.
User can insert, find, read and delete values into/from container. The type of values to be inserted into container is optional (*Int*, *Bool*, *Text*, *Real*, *Time*) or structures (entire structured variable, row, ...). The value type of the key must be one of *Int*, *Bool*, *Text*, *Real* or *Time*, but all keys in container must be the same type.

The actions **CNT_GETNR**, **CNT_CNVTOARRAY** and **CNT_GETITEM** allow reading container values by index. The action **CNT_CNVTOARRAY** internally creates an array, that contains all container values sorted by key in ascending order. Index is a sequence number of value within the array.
Array is terminated after inserting or deleting a value into/from it.

Data container may be created by the actions **CNT_CREATE** (a spare container), or **GETARCHARR_TO_CNT**. The second type is filled by pages containing data read from archive.
The first access to the archive is more effective (memory consumption and partly speed) than using GETARCHARR action. Data container created by GETARCHARR_TO_CNT action can use only the actions CNT_GETNR, CNT_FIND and CNT_DESTROY (example).

- CNT_CNVTOARRAY
- CNT_CREATE
- CNT_DEBUG
- CNT_DELETE
- CNT_DESTROY
- CNT_FIND
- CNT_GETITEM
- CNT_GETKEY
- CNT_GETNR
- CNT_INSERT

Data container transfer between running ESL scripts

Access rights manipulation actions

The actions allow to define access rights when the system is running.

- REBUILD_ACC
- RES_GROUP_DELETE
- RES_GROUP_DELETE_ALL
- RES_GROUP_INSERT
- RES_GROUP_QUERY

Actions for manipulation with list of objects

Actions for manipulation with list of objects allow to:

- create the list of objects according to set criteria
- go on the first, previous, next or last page of the list
- go on the page on the basis of its page number
- recognize the quantity of objects in list (caution - it is not quantity of objects on page!)
- close the list

After the list has been created data are accessible on the first page immediately.
Each record in list of object represents the unique identifier, name, description, type of object and number of rows and columns.

- LST_CREATE
- LST_CLOSE
- LST_GO_NEXT
- LST_GO_PREV
- LST_GO_LAST
- LST_GO_FIRST
- LST_GO_PAGE
- LST_GETINFO

Example of manipulation with list of objects (actions LST_...).

Other actions

- COMMAND
- COPYOBJECT
- DELETEOBJECT
- FIND_FILES
- GETPOINTADR
- GETOLDVAL
- LOG
- LOGEX
- PLAY
- READLOG
- REDIM
- RUN
- RUNEX
- SETDT_LINEOBJ
- ON EDA_WARNING