Data container

Data container (internal data structure) is an internal data structure that enables to save both simple and structured values according to a key.
The owner of a container is always one running instance of the script. It can be shared between various scripts and processes.

Data container may be created by the actions CNT_CREATE (a spare container), or GETARCHARR_TO_CNT. The second type is filled with pages containing data read from the 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).

The container has a unique identifier. The container is automatically terminated by terminating the script, to which it belongs, or by executing the **CNT_DESTROY** action.
The size of a container is not specified and is only limited by the operating memory size.

Each value included in a container is uniquely determined by the so-called *key*.
Users can insert, find, read and delete values into/from the container. The type of values to be inserted into the 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 the container must be the same type.

The actions that ensure work with the data container are stated on page Script actions.

Transferring data container between running ESL scripts

A container transfer can be done by RPC procedures. In the declaration for the RPC procedure, you have to tag the parameter that represents the handle to the container by **CNT_HANDLE**. An algorithm is contingent on the existence of data containers. If the handle to the data container is an invalid value or points to a non-existent data container, the algorithm ends with an error
The value of **CNT_HANDLE** local variables is an integer (INT).

RPC procedure declaration:

```
 RPC PROCEDURE ProcName [([IN]  CNT_HANDLE paramName1[,paramName2, ...] [IN]  CNT_HANDLE paramName3]...)]

;actions

END ProcName
```

Notes:

- The owner of the data container can be only one ESL script, which also ensures the canceling of the data container.
- Data containers can be transferred also between ESL scripts that belong to different processes. In this method, all data that are placed in the container are transferred.
- When calling the RPC procedure, if you use the value that is not the handle on the place for a formal parameter of CNT_HANDLE type, the ESL script will search the data container according to an input value.
    1. If a data container, tagged by input value, **exists**, the script **transfers** it.
    2. If a data container, tagged by input value, **does not** exist, the script **displays** an error - container with handle = x not found (there are not any data 22).

    ```
    INT _cnt_handle

    __cnt_handle := 5

    CALL [objIdent] CNT (_cnt_handle) ON procIdent
    ```

    3. If the input value, representing handle, is invalid, the calling Esl script ends with RunTime error.
- If the calling RPC procedure is **asynchronous**, the container is terminated in this script. Then ESL script that has been called becomes the owner of this container:

```
******************

; ESL script that is called
RPC PROCEDURE InsertToContainer(CNT_HANDLE _handle)
.....
END InsertToContainer

***************


 ; ESL script that is calling
INT _cnt_handle

CALL[...] InsertToContainer(_cnt_handle) ASYNC ON
....
; the container is terminated in this script, the owner is the called ESL script

******************
```

- If the calling RPC procedure is **synchronous**, there are two options:
  1. If the formal parameter, which represents CNT_HANDLE, is tagged by the keyword IN, when calling the RPC procedure, the ESL script containing the declaration of called RPC procedure will be **permanently** the owner of the data container.

     ```
     ******************
     ; ESL script that is called
     RPC PROCEDURE InsertToContainer(IN CNT_HANDLE _handle)
     .....
     END InsertToContainer
     ***************


     ; ESL script that is calling
     INT _cnt_handle

     CALL[...] InsertToContainer(_cnt_handle) ON ....
     ; after this calling the data container is terminated, the owner is the called ESL script
     ***************
     ```

  2. If the formal parameter, which represents CNT_HANDLE, is not tagged by the keyword IN, when calling RPC procedure, ESL script containing the declaration of called RPC procedure will be the **temporal** owner of data container. After finishing the called RPC procedure, the script, from which the RPC procedure has been called, will become the owner.

     ```
     *******************
     ; ESL script that is called
     RPC PROCEDURE InsertToContainer(CNT_HANDLE _handle)
     .....
     END InsertToContainer
     ***************


     ; ESL script that is calling
     INT _cnt_handle

     CALL[...] InsertToContainer(_cnt_handle) ON ....
     ; after this calling, the owner of data container is still ESL script that is calling
     ***************
     ```

Example 1:

```
INT _INTER_HANDLE

RPC PROCEDURE MakeWithCNT_IN (IN CNT_HANDLE _CNT_Handle, BOOL _bOk)

INT _iKey
INT _value
BOOL _bFound

 _iKey := 1
 _INTER_HANDLE := _CNT_Handle
 CNT_FIND _INTER_HANDLE, _iKey, _value, _bFound

 IF _bFound THEN
 _value := 3
 CNT_INSERT _INTER_HANDLE, _iKey, _value
ENDIF

END MakeWithCNT_IN

RPC PROCEDURE MakeWithCNT_IN_OUT (CNT_HANDLE _CNT_Handle, BOOL _bOk)

INT _iKey
INT _value
BOOL _bFound

 _iKey := 1
 _INTER_HANDLE := _CNT_Handle
 CNT_FIND _INTER_HANDLE, _iKey, _value, _bFound

 IF _bFound THEN
 _value := 3
 CNT_INSERT _INTER_HANDLE, _iKey, _value
 ENDIF

END MakeWithCNT_IN_OUT
```

Example 2:

```
INT _Handle

PROCEDURE Interny_call(BOOL _bOk)
INT _iKey
INT _value
BOOL _bFound

_bOk := @TRUE
_iKey := 1

;*****************************************************************************
;** it transfers cnt into other script and changes value 1 to 3 in the key
;**
;*****************************************************************************

CALL [E.1] MakeWithCNT_IN_OUT(_Handle,_bOk) ON SELF.EVH

IF !_bOk THEN
RETURN
 ENDIF

CNT_FIND _Handle, _iKey, _value, _bFind

_bOk := _bFind & _value = 3

;*****************************************************************************
;** it transfers cnt into other script, this script discards the cnt
;**
;*****************************************************************************

CALL [E.1] MakeWithCNT_IN(_Handle, _bOk) ON SELF.EVH
END Interny_call

BEGIN

INT _iKey
INT _value

 _iKey := 1
_value := 2
CNT_CREATE _Handle
CNT_INSERT _Handle, _iKey, _value

END
```

Related pages:

Script actions