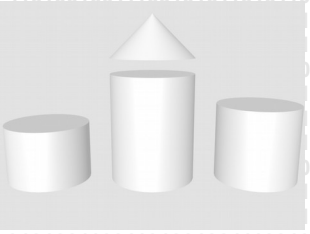

ODABA NG

OShell Command Reference

run Software-Werkstatt GmbH
Winckelmannstrasse 61
12487 Berlin

Tel: +49 (30) 609 853 44
e-mail: run@run-software.com
web: www.run-software.com

Berlin, September 2018

Table of Contents

1 Introduction

OShell is a command line utility that allows running most of the ODABA access functions from a command line. In contrast to **OSI**, **OShell** is not a query tool, but a way to navigate through a database similar to navigating through the directory structure on a disk. OShell commands are not case sensitive. For documentation purposes upper case letters are used in this chapter. Most of commands have got an abbreviation. Details for calling OShell are described in **3.1_DatabaseUtilities**.

Typically, **OShell** is called as

```
OShell ini_file [script_file]
```

A script file may be passed in order to run a predefined request. When the script file does not contain a quit command, OShell changes into command line input mode, which allows entering further commands.

Typically, one has do select a data source and a data collection at the beginning:

```
CD Sample
CC Companies
```

In order to enable certain options (e.g. for debugging), one may use the SET command. The following topics contain a short description for available commands.

When parameters passed to a command contain spaces or other special characters, parameters have to be passed within apostrophes (') or quotes ("). In order to insert comment lines, those have to begin with double slash (`//`).

Additional options may be passed to many commands, which are indicated b '-' (e.g. -D1). Options are always optional, but not always explicitly marked as such.

All OShell commands are case insensitive. Most command may be called with a short command name.

2 Process flow control commands

In order to support conditional processing or loops, a number of process flow commands are supported. A list of sub-commands for a process flow command (e.g. IF or WHILE) may be embedded in a DO/BEGIN-END block.

2.1 Run command block and reset state (BEGIN)

The command introduces a block of commands, which is terminated by an END command. Command lines must be entered completely before the block is executed. In contrast to DO, the state of the process is saved at beginning and restored at the end, i.e. commands in the BEGIN block may change the current collections and instance selections without affecting processing after the END statement.

```
Syntax:
  begin
```

2.2 Run command block and keep state (DO)

The command introduces a block of commands, which is terminated by an END command. Command lines must be entered completely before the block is executed. In contrast to BEGIN, DO returns he current state of selected collections and instances after terminating the DO block.

```
Syntax:
  do
```

2.3 Terminate command block (END)

The command terminates a command block, which has been introduced by a DO or BEGIN command. With the end command processing of statements within the block begins. This includes also syntax and semantic checks of the commands in the block.

```
Syntax:
  end
```

2.4 Leave block (LB)

The LB or LEAVEBLOCK statement allows leaving a BEGIN/DO-END block before processing all statements. This is important especially for leaving while blocks.

The command will leave the current block and continue with the next statement after the END statement.

```
Syntax:
  leaveblock|lb
```

2.5 Return from procedure (RETURN)

When calling a procedure defined in a command file, the defined procedure should be terminated by a RETURN statement. The command will leave the current procedure and continue with the next statement after the procedure call. When no RETURN statement has been defined, the command continues until the end of the command file. This may also include commands of a subsequent procedure in the command file.

```
Syntax:
  return
```

2.6 Quit application (QUIT)

The command will close all collections and data sources and terminate the **OShell** program. When running **OShell** from a command-line console with a command file, that does not contain a QUIT command, **OShell** changes to command input via console.

When debugging GUI applications or context functions, quit may not work properly. In this case, one may also call exit.

```
Syntax:
  quit|q
```

2.7 Exit application (EXIT)

The command will exit the **OShell** immediately. This is an emergency function, which might not close databases properly.

```
Syntax:
  exit
```

2.8 Pause process (PAUSE)

The command stops the execution until any key is pressed.

```
Syntax:
  pause
```

2.9 Conditional execution (IF)

The command allows running a command or a sequence of commands under certain conditions. The `command` is executed when the `expression` is **true**. One may also run a sequence of commands enclosed in a `BEGIN`/`DO`-`END` block. The complete `IF` statement including `command` or `BEGIN`/`DO` has to be defined on a single line.

Parameters:

- `expression` - The OSI expression defines the condition to be checked before running the command(s). The expression must be a valid OSI expression. When the expression contains special characters it must be put into quotes or apostrophes.
- `command` - The command to be executed when the condition ist **true**. For processing a block of statements one may use CALL for running a pre-defined procedure or a `BEGIN`/`DO`-`END` block.
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  if expression command -Dn - Cn

Examples:
  if "age > 20" p name  // Show value for 'name' when age > 20
  if "age > 20" do      // Show value for 'name' and 'age' when age > 20
    p name
    p age
  end
```

2.10 Do while condition is true

The command allows running a `command` or a sequence of commands in a `BEGIN/DO-END` block as long as the condition passed in `expression` is true. In contrast to `FA`, the function does not implicitly change the selection for the collection. Instead of a single command one may pass a sequence of commands enclosed in a begin/do-end block. The complete `WHILE` statement including `command` or `BEGIN/DO` has to be defined on a single line.

Parameters:

- `expression` - The OSI expression defines the condition to be checked before running the command(s). The expression must be a valid OSI expression. When the expression contains special characters it must be put into quotes or apostrophes.
- `command` - The command to be executed when the condition ist **true**. For processing a block of statements one may use `CALL` for running a pre-defined procedure or a `BEGIN/DO-END` block.
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.
- `-In` - limit number of iterations to `n`\n");

```
Syntax:
  while expression command -Dn -Cn -In

Examples:
  while "++age < 20" p age // print age as long as age < 20
  while next begin         // Print name for all persons with age > 20
    if 'age > 29' p name
  end
```

2.11 Do for all instances in current collection (FA)

The statement allows running `command` for all instances in a collection. `FA` starts with the currently selected instance or with the first one when no instance is selected in the collection. The complete `FA` statement including `command` or `BEGIN`/`DO` has to be defined on a single line.

Parameters:

- `command` - The command to be executed in each iteration of the loop. For processing a block of statements one may use `CALL` for running a pre-defined procedure or a `BEGIN`/`DO`-`END` block.
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.
- `-In` - limit number of iterations to `n`.

```
Syntax:
  fa command -Dn -Cn -In

Examples:
  fa lav          // List attribute values for instances in a collection
  fa call proc1   // run procedure proc1 for instances in the collection
  fa lav -D2 -C1 //show attributes for instances in collection 1 of data
                  // source 2
  fa do           // run the list of subsequent commands for all instances
in
  ...             // current collection beginning at selected instance
  end             // ('end' requested at end of commandlist)
```

2.12 Call procedure file (CALL)

The command runs commands from the file. Procedure files should not contain quit statements (q), since this will terminate the OShell program;

Parameters:

- `file_name` - complete path to the file containing the instructions. When no file name is passed, the procedure must be defined in the current command file or has to be loaded explicitly (`LOAD`)
- `entry_point` - when an entry point is appended to the file name the procedure is not processed from the beginning but from the entry point, which must be a defined entry point in the procedure.
- `parameter` - Any number of parameters may be passed to the procedure called. The receiving procedure may refer to parameters by `%1`, `%2` etc.

When calling a procedure file name with an entry point like

```
CALL myprocs.osh@start
```

the entry point name must precede the file name without space. The entry point must be defined in the file on a single line preceded by @, in this case `@start`. In order to return from the procedure, `return` has to be called. Otherwise, the procedure returns after last line in the command file.

```
Syntax:
  call [file_name][@entry_point] [parameter [parameter] ...]
```

3 Common OShell Commands

Common **OShell** commands are those that do (usually) not require an opened data source or data collection, but may also be called when data sources or collections are opened.

3.1 Display command help (HELP)

The command shows available commands or details for one command.

Parameters:

- `command` - name of the command to be displayed.
- `-d` - show details for each command (default for 'command')
- `-a` - show all commands (ignored in connection with 'command')
- `-e` - show extended commands (ignored in connection with 'command')

```
Syntax:
  help [command] [-d] [-a] [-e]
```

3.2 Load procedure file (LOAD)

The command loads commands from the file. A loaded procedure or entry points in a loaded procedure can be called at any time after the procedure is loaded.

As long as entry points in all loaded procedures are unique one may call an entry point without prefixing the procedure path (`call`). If entry point names are not unique, the procedure path must be entered before the entry point name.

Parameters:

- `file_name` - complete path to the file containing the procedure(s).

```
Syntax:
  load file_name
```

3.3 Set option variable (SET)

The function allows setting or displaying a value for an option or environment variable. Not passing any parameter will display all variables set. Option variables are not case sensitive.

Parameters:

- `var_name` - Name for the option variable (may be referenced as %var_name%)
- `value` - value to be set for the variable

```
Syntax:
  set var_name [[=]value]

Examples:
  set NAME='Smith';   // Setting value for NAME to 'Smith'
  set NAME            // display current settings for NAME
  set NAME=           // reset current settings for NAME
  set                 // display current settings for all variables
```

3.4 Redirect OShell output (REDIR)

The function allows the permanent redirection of output for the subsequent commands. Only data output is redirected to the file location passed to the command. Error and system messages are still written to the standard output. Calling `redir` without parameter will reset the redirection of the output.

Redirection of the output is also reset at the end of a do block, when being specified within the block or at the end of a called procedure, when being activated within a procedure.

Parameters:

- `path` - location (file name) for redirection.

```
Syntax:
  redir [path]

Examples:
  redir %HOME%/out.txt  // write output to file out.txt
  redir                 // reset redirection for getting the output on
                        // standard output
```

3.5 Echo command input (ECHO)

With 'echo on' you will cause **OShell** to display commands entered directly or via procedure. 'echo off' will switch off this feature. 'echo 'any text' will display the message on the console. The function is not intended to display instance data. In order to display data, `print` may be used.

```
Syntax:
  echo [ on | off | 'any text' ]
```

3.6 List data sources (LD)

The command displays available data sources defined in a catalog. The command does not display data sources defined in a configuration or ini-file.

```
Syntax:
  ld
```

3.7 Change data source (CD)

The command allows opening a data source or switching to another data source context. Not passing any parameter will show a list of opened data sources preceded by a data source number (dsid). Passing a dot (.) as parameter will close the current data source.

Parameters:

- `dsname` - data source name as defined in the catalog or configuration or ini-file section.
- `dsid` - data source identifier for opened data source (internal number).
- `*` - create a copy of current data source.
- `.` - close currently selected data source
- `-Dn` - re-direct command to data source referenced by internal number n.

```
Syntax:
  CD [dsname|*|dsid|.] -Dn:

Examples:
  cd           // Displays currently opened data sources
  cd ProjectDB // Open data source 'ProjectDB' (must be defined in
               // catalog  or ini-file
  cd 2         // Switch to data source 2 (must bee opened calling CD)
  cd .         // Close current data source and switch to previous.
  cd * -D2     // Create a data source copy from data source 2
  cd . -D3     // Close data source 3
```

4 OShell data source commands

Data source commands are available when a data source has been opened. Data source commands do not require an opened collection but may also be called, when a collection has been opened.

4.1 List extent names (LCN)

When no collection is opened (CC), the command lists the collection names for the data source (extents). When a collection is opened, the command lists the sub-collection names available for instances in the opened collection.

Parameters:

- `mask` - display collection names according to the mask, only. The mask may contain * as place holder(s).
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  lcn [mask] -Dn -Cn

Examples:
  lcn          // list all extent names
  lcn ABC*     // list all extent names beginning with ABC
```

4.2 Change collection (CC)

The command allows closing, opening or switching to an opened collection. Passing an invalid collection name will also be accepted, but later commands may fail. Calling the command without parameters shows the current hierarchy. Collection numbers are prefixed with following meaning:

- + - An istance is selected in the collection
- * - collection is the currently active collection
- - - no instance selected in the collection

Parameters:

- `prop_path` - extent or property name for the collection to be opened. One may also define an access path or view path as property path.
- `col_id` - change to an opened collection in the hierarchy.
- `.` - close current collection and return to parent. One may also close more than one collections by passing more than one dot.
- `/` - close all collections in a hierarchy.
- `-Dn` - re-direction to other data source.

```
Syntax:
  cc [ prop_path | coll_id | . | / ] -Dn

Examples:
  cc Person     // Open the 'Person' extent in the current data source
                // In this case, no collections must be opened.
  cc /Person    // Open the 'Person' collection in the current data
                // source after closing the currently opened hierarchy
  cc .          // Close last collection in hierarchy
  cc ..children // Close the last two collections and open the collection
                // 'children'
  cc 2          // Switch to collection 2 in the current hierarchy
                // without closing
  cc            // Displays the current collection hierarchy and
                // collection ids
```

4.3 Run OSI statement(s) (OSI)

The command allows running an **OSI** expression for the selected collection. One operand on the same line may follow the **OSI** command. For running complex expressions, a list of statements can be included in a `DO-END` or `BEGIN-END` block.

Parameters:

- `expression` - The expression must be an osi expression. Expressions containing special characters must be quoted (no semicolon at the end!). The expression must be defined completely on the same line.
- `statement` - A valid OSI statement or a block of statements enclosed in `BEGIN/DO-END`..
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  osi expression  // single osi call
  osi do          // block of statements
  ...             // osi statements
  end             // end of block

Examples:
  osi TotalIncome  // call OSI function defined for the currently
                   // selected data type
  // call OSI expression with special characters
  osi "if count > 0 Message('Income: ' + (string)TotalIncome)"
  // call OSI with statement(s) returning a view result
  osi do
    select (name, first_name)
    from  (Person)
    where (name > 'H');
  end
  fa p             // show results
```

Notes: Since OSI command works on current collection (except in case of `BEGIN` block), the selection for the current collection may have changed. Moreover, OSI commands may change database content.

4.4 Display formatted data (FORMAT)

The command allows displaying the content of the currently selected instance in a formatted string. When no instance is selected, instance variables remain empty. One may use the print command in a `WHILE` of `FA` loop for displaying a collection of instances.

The format command does not automatically creates a new line at ent of string, i.e. new lines must be specified explicitly.

Parameters:

- `fstring` - the format string contains fixed text and '%s' variables, which are replaced by values of the parameters passed to the command.
- `parm` - any variable name or expression, which can be evaluated for the selected instance, can be passed as parameter. All parameters are converted into strings automatically, before being passed to the format string.
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  format|f fstring [parm(*)] -Dn -Cn

Examples:
  format "hello world\n"  // prints a constant text
  // print person data in a formaatted line
  format "The address of %s %s is %s\n" first_name last_name address
```

4.5 Execute database context action (DATABASEACTION)

The function runs the action named in action_name passing optional parameters. The action called must be implemented in the database context.

Parameters:

- `act_name` - action name for the database action as defined in the database context class.
- `parm` - parameters passed to the action.
- `-Dn` - re-direction to other data source.

```
Syntax:
  databaseAction act_name [parm(*)] -Dn
```

4.6 Execute object space context action (OBJECTSPACEACTION)

The function runs the action named in action_name passing optional parameters. The action called must be implemented in the object space context.

Parameters:

- `act_name` - action name for the object space action as defined in the object space context class.
- `parm` - parameters passed to the action.
- `-Dn` - re-direction to other data source.

```
Syntax:
  objectSpaceAction act_name [parm(*)] -Dn
```

5 OShell data collection commands

Data collection commands are available when a data collection has been opened.

5.1 Collection list commands

In order to display data and meta data for an instance/collection, a number of list commands has been provided.

5.1.1 List collection property names (LCN)

The command lists the collection property names available for the data type of the opened collection.

Parameters:

- `mask` - display collection names according to the mask, only. The mask may contain * as place holder(s).
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  lcn [mask] -Dn -Cn

Examples:
  lcn          // list all collection property names
  lcn ABC*     // list all collection property names beginning with ABC
  lcn -D3 -c2  // list all collection property names for collection 2
               // of data source 3
```

5.1.2 List attribute property names (LAN)

The command lists the attribute names available for instances of the current or referenced collection.

Parameters:

- `mask` - display collection names according to the mask, only. The mask may contain * as place holder(s).
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  lan [mask] -Dn -Cn

Examples:
  lan         // list all attribute names
  lan name*   // list all attribute names beginning with 'name'
  lan -D3 -c2 // list all attribute names for collection 2, data source 3
```

5.1.3 List keys (LK)

The command lists the keys defined for data type of currently opened/active data collection. This is not identical with the list of indexes (sort orders) for the collection, which can be displayed with the 'lo' command.

Parameters:

- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  lk  -Dn -Cn
```

5.1.4 List order keys (LO)

The command lists the order keys (key names) available for instances of the current or referenced collection.

Parameters:

- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  lo  -Dn -Cn
```

5.1.5 List instances (LI)

The command shows lists all instances of the collection by key value or position. Passing the `position` parameter (`p`) will display the index-position in front of the key value.

Parameters:

- `p` - show instance position in front of each line
  - `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  li [p[osition]] -Dn -Cn -In
```

5.1.6 Show attribute or expression values (PRINT)

The command shows the attribute value(s) for an operation path, a property, parameter or local or global variable. When an variable is complex, all attribute values for the complex data type are displayed. When `expression` is an operation path, the operation result will be displayed. Calling the command without parameters will list all attribute values for the currently selected data type.

Parameters:

- `expression` - attribute name or operation path to be displayed.
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  print|p [expression] -Dn -Cn
```

Notes: Since `print` may call OSI functions, which may update data in the database, the command allows implicitly also updating data.

5.1.7 Set attribute list (SAL)

The command allows setting an attribute list for the selected collection, that are listed when calling the print (p) command.

Parameters:

- `attr_name_n` - attribute name to be displayed
- `-Dn` - re-direction to other data source context
- `-Cn` - re-direction to other collection th the hierarchy
- `-A` - append current attribute list. When this option is not set the list is deleted and rebuilt.

```
Syntax:
  sal attrname_1 ... attrname_n -A -Dn -Cn

Examples:
  sal name income   // creates a new attribute list with 'name' and
                    // 'income'
  sal first_name -A // adds 'first_name' to the attribute list
  sal name -D2 -C1  // creates a new attribute list with 'name' for
                    // collection 1 in datasource context 2
```

5.2 Collection settings

Collection settings commands provide some features for collection access control.

5.2.1 Change sort order (CO)

The command allows changing the sort order for the current collection. In order to get valid index names for (the collection, LO may be used.

Parameters:

- `key_name` - key name for an index defined for the collection. When no key name is passed the default order is set.
- `gen_attr_val` - optional to the key name a value for generic orders (usually language) can be passed in order to change to a language dependent index.
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  co key_name [gen_attr_val]-Dn -Cn

Examples:
  co sk_name         // set collection order to index 'sk_name'
  co                 // reset default order
  co sk_name English // set collection oder to key 'sk_name' for Englisch
  co -C0             // reset default order in collection 0 (top
                     // collection)
```

5.2.2 Set filter condition (SF)

The command restricts the number of visible instances in the current or referenced collection by setting a filter condition. In order to reset the filter, the command may be called without condition.

Parameters:

- `condition` - filter expression for the selection. The expression may be a predefined OSI function or an inline expression. When defining inline expressions, those usually have to be quotes. In case of string delimiters within the expression, alternative string delimiters have to be used for the expression. As string delimiters ' and " may be used.
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  SetFilter|sf [condition] -Dn -Cn

Examples:
  sf 'name > "S"'   // display instances with name greater than S, only.
                    // Constant string values must be put in quotes.
  sf                // Reset filter condition
```

5.3 Collection manipulation commands

Collection manipulation commands are provided in order to change current instance selection for a collection, but also for creating, updating and deleting data. Most of these command results could also be achieved by calling **OSI** functions, but **OShell** commands are a bit more comfortable.

5.3.1 Locate instance (LOC)

The command locates an instance in the referenced collection. Locating an instance will reset the selections for all subsequent collections in the hierarchy. After locating an instance it is selected in the collection.

Parameters:

- `key_value` - key value for the instance to be located (ordered collection). When the key value is a numeric key or contains special characters, it has to be put in quotes.
- `pos` - position of instance to be located in the collection (relative 0).
- `-S` - show key for the instance located
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  loc [key_value|pos] -S -Dn -Cn

Examples:
  loc Miller|Paul  // locate instance for key 'Paul|Miller'
  loc 0            // locate first instance n collection
  loc "0"          // locate instance with key value '0'
```

5.3.2 Position forward (NEXT)

The command locates the next instance by skipping a number of count instances. The command will reset the selections for all subsequent collections in the hierarchy. After locating an instance it is selected in the collection.

Parameters:

- `count` - number of instances to be skipped (default is 0).
- `-S` - show key for the instance located
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  next [count] -S -Dn -Cn
```

5.3.3 Position backward (PREV)

The command locates the previous instance by skipping a number of count instances. The command will reset the selections for all subsequent collections in the hierarchy. After locating an instance it is selected in the collection.

Parameters:

- `count` - number of instances to be skipped (default is 0).
- `-S` - show key for the instance located
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  prev [count] -S -Dn -Cn
```

5.3.4 Create instance (CRT)

The command creates a new instance in the referenced collection. Creating an instance will reset the selections for all subsequent collections in the hierarchy.

Parameters:

- `keyval` - key value for the new instance, when the collection is ordered and key is not auto-number. Key components have to be separated by '|'. When the key value contains special characters, it has to be quoted.
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  crt [keyval] -Dn -Cn

Examples:
  crt Miller|Paul  // Create an instzance for the person Paul Miller
  crt              // Create an empty instance in the current collection
  crt -D3 -c2      // Create an empty instance in collection 2 in data
                   // source 3
```

5.3.5 Copy instance (CPY)

The command copies a single instance or a collection of instances from the current collection to the referenced collection. Copying an instance will change the selections for all subsequent collections in the target hierarchy.

Parameters:

- `key_value` - key value for the instance to be copied (ordered collection). Key components have to be separated by '|'. When the key value contains special characters, it has to be quoted.
- `position` - position of instance to be copied
- `.` - cpoy instance currently selected in the collection
- `*` - copy all instances filtered in the current colletion (--> sfc)
- `new_key` - copy and rename instance of an ordered collection (copy single instance, only
- `-Dn` - target data source for copy. When no target is defined, the instance is copied to the current data source (e.g. rename)
- `-Cn` - target collection for copy. When no target collection is passed the instance is copied to the current collection in the referenced data source (`-Dn`)

```
Syntax:
  cpy keyval|position|.|* [new_key] -Dn -Cn

Examples
  cpy Miller|Paul  // Copy person instance for Paul Miller to selected
                   // instance
  cpy 0            // Copy first instance to selected instance
  cpy . -D3 -C2    // Copy selected instance to collection 2 in data
                   // source 3
```

5.3.6 Move instance (MOV)

The command moves a single instance or a collection of instances from the current collection to the referenced collection. Moving an instance will remove the instance from the source collection and change the selections for all subsequent collections in the target hierarchy.

Parameters:

- `key_value` - key value for the instance to be copied (ordered collection). Key components have to be separated by '|'. When the key value contains special characters, it has to be quoted.
- `position` - position of instance to be moved
- `.` - move instance currently selected in the collection
- `*` - move all instances filtered in the current collection (--> sfc)
- `new_key` - move and rename instance of an ordered collection (move single instance, only
- `-Dn` - target data source for move. When no target is defined, the instance is moved to the current data source (e.g. rename)
- `-Cn` - target collection for move. When no target collection is passed the instance is moved to the current collection in the referenced data source (`-Dn`)

```
Syntax:
  mov keyval|position|.|* [new_key] -Dn -Cn

Examples:
  mov Miller|Paul // Move person instance for Paul Miller to selected
                  // instance
  mov 0           // Move first instance to selected instance
  mov . -D3 -C2   // Move selected instance to collection 2 in data
                  // source 3
```

5.3.7 Delete instance (DEL)

The command deletes an instance in the referenced collection. Deleting an instance will reset the selections for all subsequent collections in the hierarchy. Option `-E` allows deleting defect (not readable) instances.

Parameters:

- `key_value` - key value for the instance to be deleted (ordered collection). Key components have to be separated by '|'. When the key value contains special characters, it has to be quoted.
- `position` - position of instance to be deleted
- `.` - delete instance currently selected in the collection
- `*` - delete all instances filtered in the current collection (--> sfc)
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.
- -E -Delete instances that cannot be read (usually by index).

```
Syntax:
  del keyval|position|.|* -Dn -Cn  // normal instance
  del keyval|position -E           // delete defekt instance

Examples:
  del Miller|Paul // Delete person instance for Paul Miller to selected
                  // instance
  del 0           // Delete first instance to selected instance
  del . -D3 -C2   // Delete selected instance to collection 2 in data
                  // source 3
```

5.3.8 Set attribute value (SAV)

The command assigns a new value to the attribute of the instance selected in the current or referenced collection.

Parameters:

- `attrname` - name of the attribute for assignment. In case of complex data types or not unique attribute names in a inheritance hierarchy, one may also use an property path as attribute name (like `a.b.c`)
- `value` - value to be assigned to the attribute. The value is a constant, a variable valid for the data type of the selected instance or an expression.
- -Q -do not display updated value
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

An assignment operator (=) may be passed between `attrname` and `value`, but need not.

```
Syntax:
  sav attrname [=] value -Dn -Cn -Q

Examples:
  sav name='Smith'      // change name value to 'Smith'
```

5.3.9 Execute property action (PROPERTYACTION)

The function runs the action named in `action_name` passing optional parameters. The action called must be implemented in the property context of current collection.

Parameters:

- `act_name` - action name for the property action as defined in the property context class.
- `parm` - parameters passed to the action.
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  propertyAction act_name [parm(*)] -Dn -Cn
```

5.3.10 Execute instance action (INSTANCEACTION)

The function runs the action named in `action_name` passing optional parameters. The action called must be implemented in the data type context of the data type for the currently selected instance.

Parameters:

- `act_name` - action name for the database action as defined in the database context class.
- `parm` - parameters passed to the action.
- `-Dn` - re-direction to other data source.
- `-Cn` - re-direction to other collection in the hierarchy or in the data source referenced by `-Dn`.

```
Syntax:
  instanceAction act_name [parm(*)]
```

6 Debug commands

Debug commands are available when debugging OSI functions. Debug mode is indicated by DEBUG at the beginning of the command line prompt. Debug mode may be enabled (`OSI.DEBUG=true`) when running OSI, OShell but also when running GUI applications.

Beside spcific debug commands, all other OShell commands are available during debugging. When changing collections or debug frames, those changes will be reset before executing the next statement.

6.1 Debug commands controlling execution

While debugging, several commands allow controlling execution of a script.

6.1.1 Set break point (BREAK)

The command sets a break point at passed or current position.

Parameters:

- `fname` - function name for the function for setting the break point. The function is searched in the current context, i.e. the currently selected data type and its base types.
- `class` - In order to list functions in a different class, the function name has to be scoped.
- `line_number` - In order to set a break point on another than the current line a line number may be passed, which is valid in the selected frame. Valid line numbers are displayed when calling the list (L) command
- `proc_name` - A procedure name containing a list of commands to be executed when reaching the break point. The procedure name is either an entry point defined in the calling OShell script file or a file path to an OShell script file with a preceding entry point name (e.g. ~/procs/breakpoints.osh@break12)
- `do|begin` - In order to enter break point procedure statements at run time DO or BEGIN may be passed instead of procedure name. After entering one or more commands, END may be entered in order to terminate the procedure. In contrast to file procedures, RETURN is not required.

```
Syntax:
  break|b [[class::]fname] [line_number] [proc_name|do|begin]

Examples:
  b                 // set breakpont at current line
  b 10              // set breakpont at line number 10
  b bp10            // set breakpont at current line and call procedure
                    // bp10 always when reaching the breakpoint.
```

6.1.2 Reset break point (DISABLE)

The command disables a break point at passed or current position.

Parameters:

- `fname` - Function name for the function for resetting the break point. The function is searched in the current context, i.e. the currently selected data type and its base types.
- `class` - In order to list functions in a different class, the function name has to be scoped.
- `line_number` - In order to reset a break point on another than the current line a line number may be passed, which is valid in the selected frame. Valid line numbers are displayed when calling the list (LIST) command.

```
Syntax:
  disable|d [[class::]fname] [line_number]

Examples:
  d                 // reset breakpont at current line
  d 10              // reset breakpont at line number 10
```

6.1.3 Break at each statement (BREAKALWAYS)

This is a run command, which causes the debugger to stop at each statement.

```
Syntax:
  breakAlways|ba
```

6.1.4 Execute next statement (N)

Executes the current statement without stepping into function.

```
Syntax:
  n
```

6.1.5 Step into function (STEP)

Step program until it reaches a different source line. Argument N means do this N times (or till program stops for another reason). The function is also called 'step into' in some environments.

```
Syntax:
  step|s
```

6.1.6 Go to line (JUMP)

The function allows changing the current line in a function. The next statement executed is the statement at the function addressed by number. When the line is empty, the next valid statement will be selected. When the line is at the end of function, the function returns to the calling function. When changing statement in another frame, called functions are canceled.

Parameters:

- `number` - line number to be called (use L to see numbers)

```
Syntax:
  jump|j [number]

Examples:
  j 10              // continues with executing line 10
  j 999             // leaves the function that less than 999 lines
```

Notes: Accessing statements in blocks (e.g. while block) may cause problems.

6.1.7 Skip statement (JUMPOVER)

This is a run command, which causes the debugger to ignore the current and continue with the next statement. This might be a statement in the current expression or in the calling expression.

```
Syntax:
  jumpOver|o
```

6.1.8 Leave function (FINISH)

This command causes the debugger to stop at next statement in the calling function or at the next break point.

```
Syntax:
  finish|fi
```

6.1.9 Continue application (CONTINUE)

This is a run command, which causes the debugger to run the application until the next break point.

```
Syntax:
  continue|c
```

6.1.10 Run application (RUN)

This is a run command, which causes the debugger to run the application without stopping at break points any more. The debugger will still stop in case of errors.

```
Syntax:
  run|r
```

6.1.11 Terminate debugging (QUIT)

This is a debug command, which immediately terminates the application. When running the debugger under OShell, it does not terminate the OShell, but the debugger, only.

```
Syntax:
  quit|q
```

6.1.12 Change stack limit (STACKLIMIT)

The function changes the stack limit to the passed number. When no number has been passed, the current stack limit will be displayed.

Parameters:

- `number` - stack limit to be used

```
Syntax:
  stackLimit|sl [number]

Examples:
  sl 200            // limit stack frames to 200
  sl                // show stack limit
```

6.2 Information commands

Information commands provide additional information about the process state.

6.2.1 List call stack (BACKTRACE)

The command displays the current call stack beginning with the last called

OSI function. It shows all stack frames, or last COUNT frames.

Passing a negative COUNT argument, stack frames above COUNT frames are displayed.

Parameters:

- `count` - maximum stack frames to be displayed

```
Syntax:
  backtrace|bt [count]
```

6.2.2 Change stack frame (FRAME)

The function changes the stack frame to a stack level shown in the back trace list. After changing the frame, local variable in the frame selected may be inspected. Not passing a frame number or 0 resets the frame to the current stack level. Passing an invalid frame number does not change the frame.

Parameters:

- `number` - stack level number to be activated

```
Syntax:
  frame|f [number]

Examples:
  frame 2           // activate fram on stack level 2
  frame             // reactivate current frame
```

6.2.3 List function (LIST)

The command lists the current function or the function passed in *fname*. In order to list functions stored in external source files, those have to be loaded before (e.g. by calling LOAD).

Parameters:

- `fname` - function name for the function to be listed. The function is searched in the current context, i.e. the currently selected data type and its base types.
- `class` - In order to list functions in a different class, the function name has to be scoped.

```
Syntax:
  list|l [[class::]fname]

Examples:
  l                  // list current function
  l Print            // list function Print of current class or base types
  l 'Person::Print' // list function Print defined in Person class or
                     // it's base types. Scoped names have to be quoted.
```

6.2.4 List current line (LISTCURRENT)

The command lists the line with the current statement.

```
Syntax:
  listCurrent|lc
```

6.2.5 Define watch variable (WATCH)

The command defines a watch expressions for the current OSI function. Watch expressions may be defined as operation path, property name, parameter or local or global variable. When an expression return a complex data type, all attribute values for the complex data type are displayed. When expression is an operation path, the operation result will be displayed Watch variables are displayed always when a break point within the OSI function is reached.

Calling the function without expression will show all active watch expressions. Watch expressions can be removed by calling dw or deletewatch.

Prameters:

- `expression` - variable, operation path or expression to be evaluated")

```
Syntax:
  watch|w [expression]

Examples:
  w first_name      // display 'first_name' value for selected instance
  w parm1.name      // display the 'name' attribute of parameter 'parm1'
  w myFunction      // display the result of calling myFunction
```

6.2.6 Delete watch expression (DELETEWATCH)

The command allows deleting a watch expression defined within the context of the current OSI function. When passing a number, the watch expression at corresponding position is deleted. Positions are listed when calling watch without parameters. When passing an expression (e.g. variable name), the corresponding watch expression will be removed.

Parameters:

- `number` - Position of watch expression in the watch list
- `expression` - Watch expression set by a watch command.

```
Syntax:
  deletewatch|dw number|expression

Examples:
  dw 2              // Delete watch expression at position 2
  dw name           // Delete watch expression 'name'.
```