astrium

mdb

MDB Access Tool

provides access operations to the
Columbus Mission Database

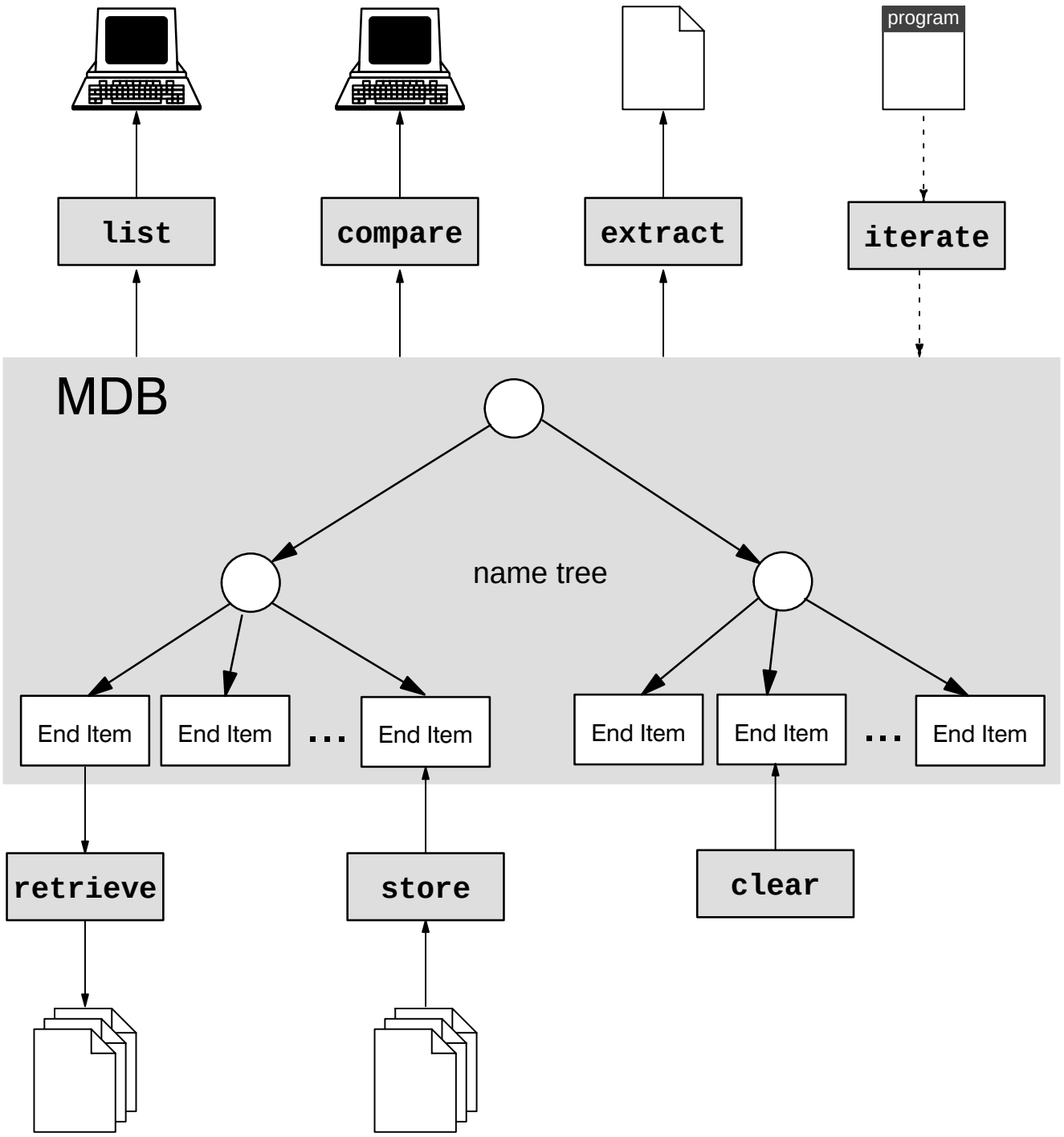
Franz Kruse
Astrium Bremen, IO 62

2002-05-14

Table of Contents

1 The mdb Tool

The mdb tool provides a simple access to the Columbus Mission Database (MDB) from within a UNIX shell. It has several subcommands that represent different operations on the database: to list (parts of) the database name tree and compare CCU/CDU versions, to move item contents between database and files, to clear item contents, to extract name tree information to be used in the database emulation, and to iterate programs over groups of items. This is depicted below:



The various tasks of the `mdb` command are expressed by dedicated subcommands:

`list`

- lists (part of) the database name tree. This function may be applied to the uppermost level or to any deeper level. It may be used to list just one level in the tree, or to list a complete subtree (or the complete name tree) recursively.

  The `list` function may be applied to single end items as well, in which case certain attributes of the end item are displayed. For certain end item classes additional information on data contents is displayed.

- lists all existing item types.

- lists all existing aliases (”nicknames”).

`compare`

compares the contents of two CCUs/CDUs or CCU/CDU versions.

`retrieve`

retrieves end item contents from the database and writes them into files.

`store`

reads end item data from files and stores it into end items in the database.

`clear`

deletes end item contents from the database, making the end item empty.

`extract`

extracts name tree information in a form that can be used to configure the emulated database, i. e. it may be copied into the configuration file `mdb.def`.

`iterate`

iterates over a selected subset of items and executes a program (given with its command line) for each item.

A word on item type mapping

To understand the handling of item types, it is necessary to know the concept of item type mapping. All items in the database have a specific item type. A set of basic item types is predefined in CGS, they are the same in any database instance. Other item types are said to be “user defined”, these item types may be different in different database instances.

User defined item types may be mapped on CGS standard types. Items of mapped types may then be regarded differently from different views, either as having the original type or as having the mapped type. This concept is typically used to allow a ground–specific view on on–board items when regarded from within a ground environment. An example is on–board software variables. They are variables when accessed from an on–board program, e. g. they may be assigend a value. But when accessed from ground they are just measurements, and it is not possible to do direct assignments to them. So the original item type (a software variable type) is mapped to a measure item type.

The `mdb list` subcommand shows both item types, and in all subcommands both the original and the mapped item type can be used.

2 Usage

2.1 Program Invocation

The program is invoked from the UNIX command line with the command

```
mdb subcommand parameters options
```

Each subcommand has its own specific parameter and option list.

```
mdb list [item] options
mdb list item_types
mdb list aliases
mdb compare item 2nd_environment [filename] options
mdb retrieve item [filename] options
mdb store item [filename] options
mdb clear item options
mdb extract item [filename] options
mdb iterate item options
```

Note, however, that the mdb command itself has global options that apply to all subcommands.

Subcommand and option names may be abbreviated by dropping characters from the end of the name, as long as the name remains unique. Each word part (separated by underscore) may be abbreviated separately. E. g. the option `-symbol_table` may be abbreviated as `-sym` or `-sym_tab`.

Subcommand and option names are not case sensitive, they may be written in lower, upper or mixed case.

Path names in the command line need a special input convention, since the UNIX shell treats backslashes in the command line as escape characters. Path names must therefore be given in quotes or with doubled (escaped) backslashes. Instead of the path name, the SID (short identifier) or the alias (”nickname”) in the form \.xyz may be used.

2.2 Global Command Options

The following global options apply to the command itself, they are valid for all subcommands.

`-environment` *environment*

defines the database user environment. If omitted, the environment will be obtained from the environment variable `MDA_ENVIRONMENT` in the same syntactic form:

```
<environment> ::=
  CCU <element_config> <mission> <system_tree_version> <ccu> |
  CDU <element_config> <mission> <system_tree_version> <cdu>

<ccu> ::=
  <pathname> <ccu_name> <version>.<issue>.<revision>

<cdu> ::=
  <pathname> <version>.<issue>.<revision> <test_version>
  <instance>
```

Example:

```
-environment ”CCU APM MASTER 4 \APM\FLTSYS FLAP_TEST 1.0.0”
```

2.3 Common parameter and option syntax

The following parameters and options appear in several subcommands. They always have the same syntax:

`item`

path name, SID or alias of a database subtree or end item. Aliases must be given in the form `\.xyz`.

- If omitted, the uppermost level of the name tree will be listed by the `list` subcommand.
- If the item denotes a subtree (or "\" for the complete name tree), one level of that (sub)tree will be treated. In this case, "..." may be appended to item identifier, which requests the complete (sub)tree to be processed recursively.
- If the item denotes an end item, that end item will be processed, and detailed information for that end item will be listed by the `list` subcommand.

Examples:

`\`	complete name tree
`\APM\EGSE`	a subtree (one level)
`\APM\EGSE...`	a complete subtree (recursively)
`\APM\EGSE\LIB\GROUND_LIBRARY`	an end item
`\.GROUND_LIBRARY`	an alias (”nickname”) of an item
`4711`	an SID of a subtree (one level)
`4711...`	an SID of a subtree (recursively)

`-selection` *`expression`*

restricts processing to a subset of items that match the given expression. The expression is a single pattern, possibly containing wildcards, for a path name, an SID or an alias (”nickname”), or several such patterns combined with the keywords **`or`**, **`and`**, **`not`**, where **`and`** has higher precedence than **`or`**, and **`not`** has highest precedence. The expression should be masked (enclosed in quotes), in order to avoid misinterpretation or expansion by the UNIX shell.

Wildcards `*` and `?` may be used within the patterns:

- If a pattern starts with a single backslash, it denotes a path name pattern. Only items whose path name matches the pattern are selected.
- If it starts with a backslash followed by a dot (`\.`), it denotes an alias pattern. Only items whose alias (”nickname”) matches the pattern are selected.
- Otherwise it denotes an item type pattern. Only items whose item type name matches the pattern are selected.

Examples:

`-selection "*_MEASUREMENT"`

all measurements (all items whose item type ends in `_MEASUREMENT`)

`-selection "ucl* and \*_ap_* or hlcl*"`

all UCL items whose path name contains `_ap_` and all HLCL items.

`-selection "\.*_LIB or \.*_LIBRARY"`

all libraries (all items whose alias ends in `_LIB` or `_LIBRARY`)

2.4 Syntax Summary

```
mdb                                          <- MDB access tool
  [-environment <value>...="$MDA_ENVIRONMENT"]
  [-verbose]                                 <- print verbose output

  list                                       <- list (sub)tree or end item
    [<item>="\"]                             <- path name or SID of (sub)tree or item
                                                 or object to be listed
    [-selection <value>]                     <- restrict output to subset of
                                                 item types, pathnames or aliases
    [-format <value>]                        <- output format specification

  compare                                    <- compare two CCUs or CDUs
    <item>                                   <- path name or SID of (sub)tree or item
    <2nd_environment>                        <- second environment
    [<filename>]                             <- output file name
    [-selection <value>]                     <- restrict iteration to subset of
                                                 item types, pathnames or aliases
    [-source]                                <- compare source code too
    [-onboard_view]                          <- use database on-board view
    [-temp_directory <value>="/tmp"]         <- directory for temporary files

  retrieve                                   <- retrieve item contents into files
    <item>                                   <- path name or SID of item
    [<filename>]                             <- target file name
    [-source [<value>]]                      <- retrieve source
    [-specification [<value>]]               <- retrieve spec. source
    [-code [<value>]]                        <- retrieve I-code
    [-symbol_table [<value>]]                <- retrieve symbolic table
    [-debug_table [<value>]]                 <- retrieve debug table
    [-xref_list [<value>]]                   <- retrieve xref list
    [-spec_xref_list [<value>]]              <- retrieve spec. xref list
    [-parameter_list [<value>]]              <- retrieve parameter list
    [-sid_location_list [<value>]]           <- retrieve sid location list
    [-onboard_flag]                          <- retrieve onboard flag
    [-id]                                    <- retrieve system library id
    [-interactive_flag]                      <- retrieve interactive flag

  store                                      <- store item contents from files
    <item>                                   <- path name or SID of item
    [<filename>]                             <- target file name
    [-source [<value>]]                      <- store source
    [-specification [<value>]]               <- store spec. source
    [-code [<value>]]                        <- store I-code
    [-symbol_table [<value>]]                <- store symbolic table
    [-debug_table [<value>]]                 <- store debug table
    [-xref_list [<value>]]                   <- store xref list
    [-spec_xref_list [<value>]]              <- store spec. xref list
    [-parameter_list [<value>]]              <- store parameter list
    [-sid_location_list [<value>]]           <- store sid location list
    [-onboard_flag <value>]                  <- store onboard flag
    [-id <value>]                            <- store system library id
    [-interactive_flag <value>]              <- store interactive flag

  clear                                      <- delete item contents
    <item>                                   <- path name or SID of item
    [-source]                                <- delete source
    [-specification]                         <- delete spec. source
    [-code]                                  <- delete I-code
    [-symbol_table]                          <- delete symbolic table
    [-debug_table]                           <- delete debug table
    [-xref_list]                             <- delete xref list
    [-spec_xref_list]                        <- delete spec. xref list
    [-parameter_list]                        <- delete parameter list
    [-sid_location_list]                     <- delete sid location list
```

```
extract                                 <- extract name tree definition file
  <item>                                <- path name or SID of (sub)tree or item
  [<filename>]                          <- output file name, default: standard output
  [-selection <value>]                  <- restrict output to subset of
                                            item types, pathnames or aliases
  [-parents [<value>]]                  <- list parent subtrees

iterate                                 <- iterate a program over items in
                                            (sub)tree
  [<item>="\"]                          <- path name or SID of (sub)tree or item
  [-selection <value>]                  <- restrict iteration to subset of
                                            item types, pathnames or aliases
  [-execute <value>]                    <- program command line including
                                            placeholders (in quotes!)
```

3 Subcommands

3.1 The `list` Subcommand

This subcommand has two forms:

```
mdb list [item[...]="\"] [options]
mdb list item_types [options]
mdb list aliases [options]
```

The first form lists the database name tree or parts of it, the second lists existing item types, and the third existing aliases (”nicknames”).

Parameters:

`item`

the database item to be treated, see 2.3 for a description.

Options:

`-selection expression`

restricts the output to a subset of items that match the given expression, see 2.3 for a description.

`-format format_string`

requires a specific output format expressed in the format string. If omitted, all output is in standard format, as shown in the examples. The format string specifies the form of an output line. It should be enclosed in single quotes to prevent it from interpretation by the shell. The format string contains text with placeholders of the form `{}` or `{name}`, or with a width field: `{name:width}`, `{name:<width}` or `{name:>width}`, see below.

For each selected item, one line of output will be generated. The text will appear unchanged in the output, whereas the placeholders will be substituted with attributes of the processed item in each iteration step, they may be abbreviated.

Placeholders:

`{}`	pathname (short form for `{pathname}`)
`{pathname}`	pathname
`{alias}`	alias (”nickname”)
`{sid}`	SID
`{item_type}`	item type
`{sw_type}`	software type (UCL syntax)
`{access_class}`	access class
`{owner}`	owner
`{ci_number}`	CI number
`{status}`	status
`{subitem_flag}`	subitem flag (FALSE, TRUE)
`{parameter_flag}`	parameter flag (FALSE, TRUE)
`{onboard_flag}`	onboard flag (TRUE, FALSE)
`{interactice_flag}`	interactive flag for libraries (TRUE, FALSE)
`{id}`	system library ID (0 .. 255)

{priority}	execution priority (LOW, HIGH)
{compilation_time}	compilation time of the body (UCL syntax)
{spec_compilation_time}	compilation time of the specification (UCL syntax)
{compiled_flag}	compiled flag of the body (FALSE, TRUE)
{spec_compiled_flag}	compiled flag of the specification (FALSE, TRUE)
{up_to_date}	item body is up to date (FALSE, TRUE)
{spec_up_to_date}	item specification is up to date (FALSE, TRUE)
{complete}	item is complete, i. e. ready to run (FALSE, TRUE)
{change_date}	date and time of the last change to the item (UCL syntax)

Size and length attribute placeholders (size = bytes, length = number of elements):

{source.size}	size of body source
{specification.size}	size of specification source
{code.size}	size of code
{debug_table.size}	size of debug table
{symbol_table.size}	size of symbol table
{parameter_list.length}	length of parameter list
{xref_list.length}	length of body cross–reference list
{spec_xref_list.length}	length of specification cross–reference list
{sid_location_list.length}	length of specification cross–reference list

Placeholders may contain a width field, they are then of one of the the following forms:

{*name*:*width*}	default alignment (numbers right, text left)
{*name*:<*width*}	left alignment
{*name*:>*width*}	right alignment

When a width field is given, the placeholder will be expanded to at least that width. Numeric values are right aligned, text values are left aligned. Example: print SIDs with a width of 5 characters:

```
{sid:5}
```

Spaces may be freely inserted within the brackets.

Example: explicit output format requested

When the –format option is given to define a specific output format, one line is output for each listed item. Each line has the form of the format string with placeholders replaced with actual data.

```
mdb list ... –format “{sid:5}: {pathname}”
```

```
    2: \MTFF\EGSE\RECEIVER
    3: \MTFF\EGSE\X_MITTER
  503: \MTFF\EGSE\TEST
  620: \MTFF\EGSE\PM
  627: \MTFF\EGSE\HARN
  638: \MTFF\EGSE\MEA1
  639: \MTFF\EGSE\V1
  640: \MTFF\EGSE\ST1
  650: \MTFF\EGSE\V2
  660: \MTFF\EGSE\ST2
  ...  ...
```

Please note:

The following examples do not have an explicit –format option and, therefore, use standard ouput format.

Example: item omitted

This example shows the basic `list` output for one level in the name tree: for subtrees only the root node is given, the subtree itself is not listed but indicated as a trailing "\...".

```
mdb list

Database tree \
VIRTUAL                              2 Path ---- \RECEIVER\...
VIRTUAL                            620 Path ---- \PM\...
STAU_HARNESS                       627 ---- ---- \HARN
EGSE_FLOAT_MEASUREMENT             638 Read Real \MEA1
EGSE_DISCRETE_SW_VARIABLE          639 Writ Bool \V1
EGSE_ANALOG_STIMULUS               640 ---- ---- \ST1
EGSE_INTEGER_SW_VARIABLE           641 Writ Int  SW_INT: \SW_INT
EGSE_DISCRETE_SW_VARIABLE          642 Writ Bool SW_BOOL: \SW_BOOL
VIRTUAL                            668 Path ---- LUEDER: \LUEDER\...
VIRTUAL                            667 Path ---- GERD: \GERD\...
```

The columns have the following meaning:

item type — the database item type as defined in MPS_DEFINITIONS (type T_ITEM_TYPE)

short ID — the "short identifier", a 32–bit number (type SID in MPS_DEFINITIONS)

access class — a mnemonic identifying the kind of access from within UCL/HLCL:

`----:`	none	no access class
`Read:`	READ	read access only
`Writ:`	READ_WRITE	read and write access
`Path:`	PATH_SELECT	may be used as an incomplete path name (root of a subtree)
`Node:`	NODE_SELECT	defines a node (computer) in a network
`Send:`	SEND	a command/stimulus definition
`Imp:`	IMPORT	a library (may be imported)
`Exec:`	EXECUTE	an executable object (AP)

software type — object type for access from within UCL/HLCL:

`----:`	none
`Int:`	INTEGER
`Unsg:`	UNSIGNED_INTEGER
`Real:`	REAL
`Dbl:`	LONG_REAL
`Bool:`	BOOLEAN
`Char:`	CHAR
`Strg:`	STRING
`BStr:`	BYTE_STRING
`Stat:`	STATE_CODE
`Byte:`	BYTE
`Word:`	WORD
`Long:`	LONG_WORD
`Set:`	BITSET
`Path:`	PATHNAME
`Time:`	TIME
`Comp:`	COMPLETION_CODE
`Puls:`	PULSE
`Burs:`	BURST_PULSE

pathname — the path name of the item, possibly preceded by its alias, if any (a trailing "\..." marks the root node of a subtree).

Example: list a subtree (one level)

One level of the corresponding subtree is listed in the same form as above. Note that a tilde (~) replaces the current level prefix (here \MTFF\DUMMY):

```
mdb list "\mtff\dummy"

Database subtree \MTFF\DUMMY
VIRTUAL                          769 Path ---- GROUND: ~\GROUND\...
VIRTUAL                          772 Path ---- ONBOARD: ~\ONBOARD\...
UCL_AUTOMATED_PROCEDURE          775 Exec ---- ~\AP
UCL_SYSTEM_LIBRARY               776 Imp  ---- ~\SYS_LIB
UCL_USER_LIBRARY                 777 Imp  ---- ~\USER_LIB
```

Example: list a complete subtree recursively

Now a complete subtree is recursively listed, i.e. the specified subtree and all deeper subtrees. Single subtrees are separated by empty lines. Note that, again, the tilde (~) replaces the name prefix and component names of levels above the current level.

```
mdb list "\mtff\dummy..."

Database subtree \MTFF\DUMMY...
VIRTUAL                          769 Path ---- GROUND: ~\GROUND\...
WDU_GROUND_SYMBOL                770 ---- ---- ~\~\SYMBOL
WDU_GROUND_SYNOPTIC_DISPLAY      771 ---- ---- ~\~\SYN_DISPLAY

VIRTUAL                          772 Path ---- ONBOARD: ~\ONBOARD\...
WDU_GROUND_SYMBOL                773 ---- ---- ~\~\SYMBOL
WDU_GROUND_SYNOPTIC_DISPLAY      774 ---- ---- ~\~\SYN_DISPLAY

UCL_AUTOMATED_PROCEDURE          775 Exec ---- ~\AP
UCL_SYSTEM_LIBRARY               776 Imp  ---- ~\SYS_LIB
UCL_USER_LIBRARY                 777 Imp  ---- ~\USER_LIB

VIRTUAL                         1004 Path ---- EGSE: ~\EGSE\...
VIRTUAL                         1005 Path ---- ~\~\POWER_SCOE\...
VIRTUAL                         1007 Path ---- ~\~\~\MMSU\...
EGSE_DISCRETE_MEASUREMENT       1008 Read Stat ~\~\~\~\HEALTH_STATUS
EGSE_FLOAT_MEASUREMENT          1012 Read Real ~\~\~\~\VOLTAGE
GENERAL_PURPOSES                1014 ---- ---- ~\~\~\~\GENERAL
HOUSEKEEPING_DATA               1015 ---- ---- ~\~\~\~\PACKET

VIRTUAL                         1006 Path ---- ~\~\COMMS_SCOE\...
VIRTUAL                         1009 Path ---- DMS: ~\DMS\...
VIRTUAL                         1010 Path ---- ~\~\MEASUREMENT\...
EGSE_ANALOG_STIMULUS            1011 ---- ---- ~\~\~\TEST_END
```

Example: list the complete name tree recursively

The following command lists the complete name tree recursively.

```
mdb list "\..."

Database tree \...
... (complete name tree will be listed)
```

Example: list items by selection

The following command lists the complete name tree recursively.

```
mdb list "\mtff\dummy..." -selection "\*_on and *ucl*"

Database subtree \MTFF\DUMMY\...
... (list of items whose path name ends in "_ON" and whose item type name contains "UCL")
```

Example: list a general end item

If the path name, SID or alias of an end item is given, detailed information on this end item is displayed:

```
mdb list "\mtff\egse\meas_1"

EGSE_INTEGER_MEASUREMENT  \MTFF\EGSE\MEAS_1
--  Alias          =
--  SID            = 1008001
--  Access class   = READ
--  Software type  = INTEGER
--  Owner          = <undefined>
--  CI Number      = <undefined>
--  Status         = NONE
--  Last change    = 22.03.2002 14:17:30
```

In case the item is mapped from some user defined item type, the original item type, together with its attributes, is displayed as well:

```
mdb list "\mtff\egse\var_1"

EGSE_DISCRETE_MEASUREMENT  \MTFF\EGSE\VAR_1
--  Alias          =
--  SID            = 3001140
--  Access class   = READ
--  Software type  = statecode
--  Owner          = <undefined>
--  CI Number      = <undefined>
--  Status         = NONE
--  Last change    = 22.03.2002 14:22:36
mapped from SW_DISCRETE_VAR
-- Access class   = READ_WRITE
-- Software type  = statecode ($OFF, $ON)
```

Example: list a CLS end item

For CLS end items, including parameterized items, additional information on data contents of single end item components is displayed, giving mainly the size of the data contained in the components, as well as the parameter list (if any) and compilation state information:

```
mdb list "\.ap_1"

UCL_AUTOMATED_PROCEDURE  \MTFF\EGSE\APS\AP_1 (...)
  in SILENT   : BOOLEAN := TRUE
  in TIME_TAG : TIME := ~:~
--  Alias          = AP_1
--  SID            = 9001001
--  Access class   = EXECUTE
--  Software type  = NONE
--  Owner          = <undefined>
--  CI Number      = <undefined>
--  Status         = NONE
--  Last change    = 22.03.2002 14:33:29
Contents:
  .SOURCE       = size 1369
  .PARAM_LIST   = <null>
  .CODE         = size 1022
  .DEBUG_TABLE  = size 134
  .SYMBOL_TABLE = size 21
  .X_REF_LIST   = defined (2 entries)
  .ONBOARD      = FALSE
Compilation:
  Body  : compiled 20.03.2002 15:46:18, up-to-date
  State : complete
```

If recursive mode is requested (appending '`...`' to the path name, SID or alias), a list of all (direct and indirect) dependencies of the item is appended to the output:

```
mdb list "\.ap_1..."

UCL_AUTOMATED_PROCEDURE  \MTFF\EGSE\APS\AP_1 (...)

...
... like above
...
Dependencies:
  UCL_SYSTEM_LIBRARY     \MTFF\EGSE\SYSTEM_LIBS\GROUND_LIBRARY
  UCL_USER_LIBRARY       \MTFF\EGSE\USER_LIBS\SUPPORT_LIBRARY
  UCL_USER_LIBRARY       \MTFF\EGSE\USER_LIBS\TIME_LIBRARY
  EGSE_FLOAT_MEASUREMENT \MTFF\EGSE\TM\MEAS_1
  EGSE_FLOAT_MEASUREMENT \MTFF\EGSE\TM\MEAS_2
  ...
```

Example: list items with subitems

CSS simulation items (function blocks) have two special properties: a) they may have subitems (which are inputs and outputs of the function block) and b) composite function blocks represent subtrees. When listing a composite function block, similar rules apply as for virtual items. For an atomic function block details are listed like for an ordinary end item. In either case, the subitems of the function block are listed.

```
mdb list "\apm\sim\model_1"

TOPLEVEL_COMPOSITE_FB  \APM\SIM\MODEL_1
--  SID            = 7
--  Access class   = PATH_SELECT
--  Software type  = NONE
--  Owner          = <undefined>
--  CI Number      = <undefined>
--  Status         = NONE
--  Last change    = 18.03.2002 14:32:18
Subitems:
- FB_IO_INPUT    1 Writ Int     ~.INT_IN
- FB_IO_INPUT    3 Writ Puls    ~.PULSE_IN
- FB_IO_INPUT    6 Writ Real    ~.REAL_IN
- FB_IO_OUTPUT   9 Read Real    ~.REAL_OUT1
- FB_IO_OUTPUT  11 Read Int     ~.INT_OUT1
- FB_IO_OUTPUT  12 Read Burs    ~.BURST_OUT
database subtree \APM\SIM\MODEL_1
ASYNCHRONOUS_FB                 79 ---- ---- ~\FB_1 ~.*
SYNCHRONOUS_FB                  82 ---- ---- ~\FB_2 ~.*
```

If recursive output is requested ('`...`'), the subtrees (function blocks on deeper levels) will be listed recursively, together with their subitems.

Example: list subitems

For subitems some details of the subitem are listed:

```
mdb list "\apm\sim\model_1."

FB_IO_INPUT  \APM\SIM\MODEL_1.PULSE_IN
--  Subitem ID     = 3
--  Access class   = READ_WRITE
--  Software type  = PULSE
--  Software value = FALSE
```

3.2 The `compare` Subcommand

The `compare` subcommand compares the contents of two CCU or CDU versions and reports all differences found. The first environment to be compared is the environment selected with the global `-environment` option (see description in chapter 2.2), the second environment is given as a parameter. The following differences are considered:

- pathname differences:
  Do the two environments contain the same pathnames?
- item type differences:
  Do items with the same pathname have the same item type?
- source code differences:
  Do two otherwise identical items have identical source code? Differences are shown in the UNIX `diff` style.

Call:

`mdb compare` *`item 2nd_environment options`*

Parameters:

`item`

the database item to be treated, see 2.3 for a description.

`2nd_environment`

the second environment (CCU/CDU version) to be compared against the current environment. It is given in the same form as the value of the global `-environment` option (see chapter 2.2).

Options:

`-filename`

the name of an output file. If omitted, differences are written on standard output.

`-selection` *`pattern`*

restricts the output to a subset of items that match the given expression. See the `list` subcommand for a description.

`-source`

requests the source code to be compared too. If omitted, source code is not compared.

`-onboard_view`

selects the onboard database view, i. e. only unmapped item types are seen, default: ground view.

`-temp_directory`

sets the directory to be used for temparary files, default: `/tmp`.

Example:

This command compares two CCU versions. The first environment is taken from the MDA_ENVIRONMENT variable, the second environment is given as a parameter. Comparison is restricted to APs only, source code is compared too, and onboard view is used.

```
mdb compare ”\APM\FLTSYS\APP_SW\FLAP...” \
            ”CCU APM MASTER 6 \APM\FLTSYS TLSE_DEL_V211 1.0.0” \
            –sel ”UCL*PROCEDURE” –source –onboard

Compared path: \APM\FLTSYS\APP_SW\FLAP...

Selection:     UCL*PROCEDURE

Environment 1: CCU APM MASTER 6 \APM\FLTSYS TLSE_DEL_V210 1.0.0
Environment 2: CCU APM MASTER 6 \APM\FLTSYS TLSE_DEL_V211 1.0.0

======================
| PATHNAME DIFFERENCE |
======================

These items are in environment 1 but not in environment 2
-------------------------------------------------------------------------------
\APM\FLTSYS\APP_SW\FLAP\ATOMIC_DMSS\XCMU\N_PWR_A_OFF
\APM\FLTSYS\APP_SW\FLAP\ATOMIC_DMSS\XCMU\N_PWR_A_ON

These items are in environment 2 but not in environment 1
-------------------------------------------------------------------------------
\APM\FLTSYS\APP_SW\FLAP\ATOMIC_DMSS\XCMU\R_RESET

======================
| ITEMTYPE DIFFERENCE |
======================

None

=========================
| SOURCE CODE DIFFERENCE |
=========================

-------------------------------------------------------------------------------
\APM\FLTSYS\APP_SW\FLAP\ATOMIC_DMSS\DMC\RESET2
UCL_AUTOMATED_PROCEDURE
-------------------------------------------------------------------------------
66c68
<     if Discrete_Sensor_Value = $READY then
---
>     if Discrete_Sensor_Value <> $READY then
95c97
<         if Discrete_Sensor_Value = $READY then
---
>         if Discrete_Sensor_Value <> $READY then
116a119
>
```

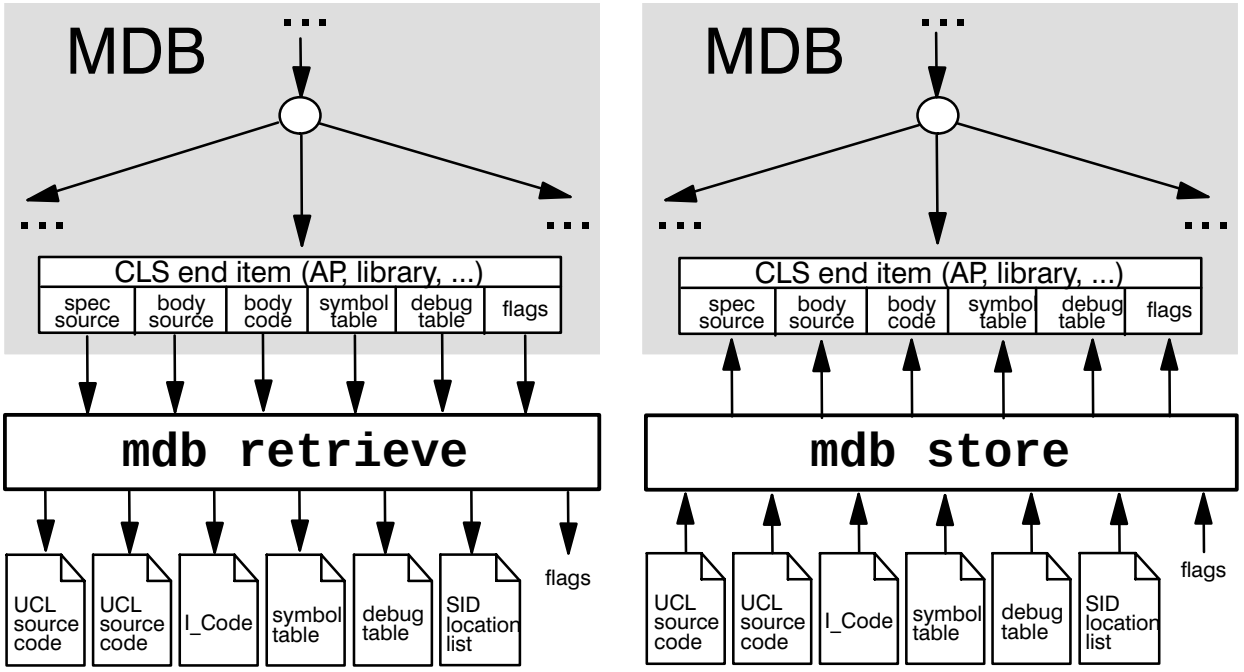
```
------------------------------------------------------------------------------
\APM\FLTSYS\APP_SW\FLAP\ATOMIC_DMSS\MMC\RESET2
UCL_AUTOMATED_PROCEDURE
------------------------------------------------------------------------------
16a17,18
> -- 25.08.2000 Mma                 SPR MMT-USM-6091 : change the check of
the
> --                                  ready status  &READY -> <> &READY
66c68
<     if Discrete_Sensor_Value = $READY then
---
>     if Discrete_Sensor_Value <> $READY then
95c97
<         if Discrete_Sensor_Value = $READY then
---
>         if Discrete_Sensor_Value <> $READY then
116a119
>

------------------------------------------------------------------------------
\APM\FLTSYS\APP_SW\FLAP\ATOMIC_DMSS\MMU1\RESET
UCL_AUTOMATED_PROCEDURE
------------------------------------------------------------------------------
16a17,18
> -- 25.08.2000 Mma                 SPR MMT-USM-6091 : change the check of
the
> --                                  ready status  &READY -> <> &READY
66c68
<     if Discrete_Sensor_Value = $READY then
---
>     if Discrete_Sensor_Value <> $READY then
95c97
<         if Discrete_Sensor_Value = $READY then
---
>         if Discrete_Sensor_Value <> $READY then
```

3.3 The `retrieve` and `store` Subcommands

The `retrieve` and `store` subcommands move data (end item contents) between the database and files: `retrieve` retrieves end items from the database and writes their contents into files, `store` reads data from files and stores it in end items in the database. The commands copy the contents of single components of end items, as shown in the picture (not all components shown). Each component of an end item corresponds to a specific file and can be selected with a specific option.

These commands can only be applied to items processed by the Columbus Language System (CLS) software:

- UCL automated procedures
- UCL system and user libraries
- HLCL command procedures
- CPL scripts
- parameterized end items
- derived values



Call:

```
mdb retrieve item [filename] options
mdb store item [filename] options
```

Parameters:

`item`

the database item to be treated, see 2.3 for a description.

`filename`

optional file name body (without file extension). If provided, the file names of all files read or written will be derived from this name by appending specific file extensions. Otherwise the file names will be derived from the item name (the last component of the path name).

Options:

When no options are provided, `retrieve` retrieves all components from the item and puts them in single files, and `store` finds all related files and copies their contents into the corresponding item components. If one or more options are provided, only the specified components are retrieved or stored.

For options with a file name value: the file name is optional. If a file name is given, that file will be used, otherwise the file name will be derived from the item name with the appropriate file extension.

`-source [`*`filename`*`]`

retrieve or store only the source text of the item. For UCL user libraries this selects the source text of the library body. File name extensions: `*.ucl`, `*.hlcl`, `*.cpl`

`-specification [`*`filename`*`]`

retrieve or store only the source text of the library specification (only for UCL user libraries and parameterized items). File name extension: `*_.ucl`

`-code [`*`filename`*`]`

retrieve or store only the I–Code. File name extension: `*.bin`

`-symbol_table [`*`filename`*`]`

retrieve or store only the symbol table. File name extension: `*_.sym`

`-debug_table [`*`filename`*`]`

retrieve or store only the debug table. File name extension: `*.sym`

`-xref_list [`*`filename`*`]`

retrieve or store only the cross reference list. For UCL user libraries this is the cross reference list of the body. File name extension: `*.xref`

`-spec_xref_list [`*`filename`*`]`

retrieve or store only the cross reference list of the specification (only for UCL user libraries and parameterized items). File name extension: `*_.xref`

`-sid_location_list [`*`filename`*`]`

retrieve or store only the SID location list of a derived value. File name extension: `*.loc`

`-parameter_list [`*`filename`*`]`

retrieve or store only the parameter list. File name extension: `*.par`

`-onboard_flag [`*`value`*`]`

retrieve or store the onboard flag (values: FALSE, TRUE).

`-id [`*`value`*`]`

retrieve or store the library ID of a system library.

`-interactive_flag [`*`value`*`]`

retrieve or store the interactive flag (values: FALSE, TRUE).

File names:

By default, the program creates and searches file names with the following standard extensions:

name.**ucl**	UCL source text of a body (AP, user library, derived value)
*name*_.**ucl**	UCL source text of a specification (system/user library, parameterized item)
name.**hlcl**	HLCL source text (HLCL command sequence)
name.**cpl**	CPL source text (CPL script)
name.**bin**	binary code (I–Code)
name.**sym**	debug table (AP, user library)
*name*_.**sym**	symbol table (AP, system/user library, parameterized item)
name.**xref**	cross–reference list of a body (AP, user library, CPL script, derived value)
*name*_.**xref**	cross–reference list of a specification (AP, system/user library, parameterized item, derived value)
name.**par**	parameter list (AP, HLCL command sequence, parameterized item)
name.**loc**	SID location list (derived value)

Examples:

Retrieve all data from an AP end item (the debug table component contains no data), file names are derived from the end item name:

```
mdb retrieve “\mtff\dummy\ap”

File ap.ucl retrieved from \MTFF\DUMMY\AP.SOURCE
File ap.bin retrieved from \MTFF\DUMMY\AP.CODE
Nothing retrieved from \MTFF\DUMMY\AP.DEBUG_TABLE
...
```

Store I–Code from a user library end item:

```
mdb store “\mtff\dummy\user_lib” –code

File test_lib.bin stored to \MTFF\DUMMY\USER_LIB.CODE
```

Retrieve the onboard flag of an AP:

```
mdb retrieve “\mtff\dummy\ap” –onboard

\MTFF\DUMMY\AP.ONBOARD = TRUE
```

Store set the interactive flag of a user library to TRUE:

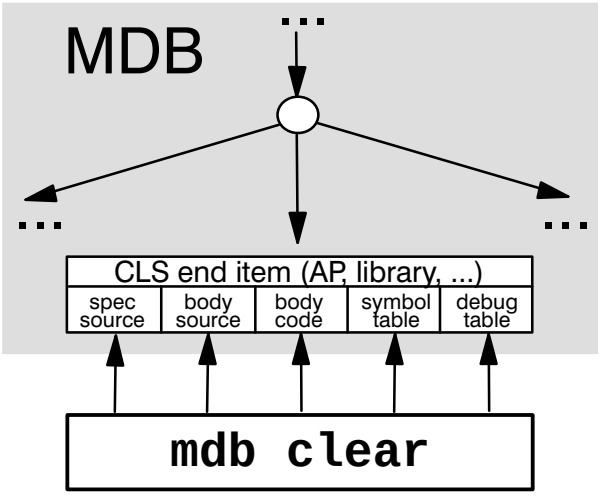
```
mdb store “\mtff\dummy\user_lib” –interactive true

TRUE stored to \MTFF\DUMMY\USER_LIB.INTERACTIVE
```

3.4 The `clear` Subcommand

The `clear` subcommand deletes the contents of end items in the database. It does not remove the end item from the database, but just makes it empty. This function is specific for different end item classes as described below.

This command can only be applied to items processed by the Columbus Language System (CLS) software:

- UCL automated procedures
- UCL system and user libraries
- HLCL command procedures
- CPL scripts
- parameterized end items
- derived value



Call:

`mdb clear` *`item options`*

Parameters:

`item`

the database item to be treated, see 2.3 for a description.

Options:

When no options are provided, `clear` clears all components of the item. If one or more options are provided, only the specified components are cleared.

`–source`

clear the source text of the item. For UCL user libraries this clears the source text of the library body.

`–specification`

clear the source text of the library specification (only for UCL user libraries).

`-code`

clear the I–Code.

`-symbol_table`

clear the symbol table.

`-debug_table`

clear the debug table.

`-xref_list`

clear the cross reference list. For UCL user libraries this is the cross reference list of the body.

`-spec_xref_list`

clear the cross reference list of the specification.

`-parameter_list`

clear the parameter list.

`-sid_location_list`

clear the SID location list of a derived value.

Examples:

Clear all components of an AP:

```
mdb clear "\mtff\dummy\ap"

\MTFF\DUMMY\AP.SOURCE cleared
\MTFF\DUMMY\AP.CODE cleared
\MTFF\DUMMY\AP.DEBUG_TABLE cleared
...
```

Clear the I–Code component of an AP:

```
mdb clear "\mtff\dummy\ap" -code

\MTFF\DUMMY\AP.CODE cleared
```

3.5 The `extract` Subcommand

The `extract` subcommand extracts name tree information in a form that can be used to configure the emulated database, i. e. it may be copied into the configuration file `mdb.def`. See the documentation on the MDB Emulation.

Call:

`mdb extract` *`item[...]`* [*`filename`*] *`options`*

Parameters:

`item`

the database item to be treated, see 2.3 for a description.

`filename`

optional name of the outpout file. If omitted, the output is written on standard output.

Options:

`–selection` *`expression`*

restricts the output to a subset of items that match the given expression, see 2.3 for a description.

`–parents` [*`levels`*]

requests that not only the selected items be output, but also (all or some) virtual nodes (parent nodes) above the current subtree level. If the option is given without a value, all parent virtual nodes are output; if a value is given, it indicates the number of parent levels to be output.

Example:

`mdb extract “\apm\fltsys\app_sw...” –selection “*ucl*” –parents`

3.6 The `iterate` Subcommand

The `iterate` subcommand iterates over a selected subset of items and executes a program (given with its command line) for each item.

Call:

`mdb iterate [`*`item`*`[...]] `*`options`*

Parameters:

`item`

the database item to be treated, see 2.3 for a description.

Options:

`–selection `*`expression`*

restricts processing to a subset of items that match the given expression, see 2.3 for a description.

`–execute '`*`command_line`*`'`

specifies the command to be called for each selected item. This option is followed by the complete command line for the invokation of the program. It should be enclosed in single quotes to prevent it from interpretation by the shell. The command line may contain any shell specific elements, like environment variables etc., which will be interpreted each time the program is started. For special treatment of input/output redirection see below. Placeholders of the form `{}` or `{`*`name`*`}` in the command line will be substituted with attributes of the processed item in each iteration step, they may be abbreviated.

Placeholders for general attributes:

`{}`	pathname (short form for `{pathname}`)
`{pathname}`	pathname
`{alias}`	alias (”nickname”)
`{sid}`	SID
`{item_type}`	item type
`{sw_type}`	software type (UCL syntax)
`{access_class}`	access class
`{owner}`	owner
`{ci_number}`	CI number
`{status}`	status
`{subitem_flag}`	subitem flag (`FALSE`, `TRUE`)
`{parameter_flag}`	parameter flag (`FALSE`, `TRUE`)
`{onboard_flag}`	onboard flag (`FALSE`, `TRUE`)
`{interactice_flag}`	interactive flag for libraries (`TRUE`, `FALSE`)
`{id}`	system library ID (0 .. 255)
`{priority}`	execution priority (`LOW`, `HIGH`)
`{compilation_time}`	compilation time of the body (UCL syntax)
`{spec_compilation_time}`	compilation time of the specification (UCL syntax)
`{compiled_flag}`	compiled flag of the body (`FALSE`, `TRUE`)
`{spec_compiled_flag}`	compiled flag of the specification (`FALSE`, `TRUE`)
`{up_to_date}`	item body is up to date (`FALSE`, `TRUE`)
`{spec_up_to_date}`	item specification is up to date (`FALSE`, `TRUE`)

{complete}	item is complete, i. e. ready to run (FALSE, TRUE)
{change_date}	date and time of the last change to the item (UCL syntax)

Placeholders for size and length attributes (size = bytes, length = number of elements):

{source.size}	size of body source
{specification.size}	size of specification source
{code.size}	size of code
{debug_table.size}	size of debug table
{symbol_table.size}	size of symbol table
{parameter_list.length}	length of parameter list
{xref_list.length}	length of body cross–reference list
{spec_xref_list.length}	length of specification cross–reference list
{sid_location_list.length}	length of specification cross–reference list

Placeholders for data contents in temporary files (files are read–only):

{source}	name of file containing the source text of the body
{specification}	name of file containing the source text of the specification
{code}	name of file containing the code
{symbol_table}	name of file containing the symbol table
{debug_table}	name of file containing the debug table
{parameter_list}	name of file containing the parameter list
{xref_list}	name of file containing the cross–reference list of the body
{spec_xref_list}	name of file containing the cross–reference list of the specification
{sid_location_list}	name of file containing the SID location list of a derived value

Input/output redirection:

The called program may read the data contents from standard input and write modified data on standard output, using the following placeholders:

{<source}	program will read the source of the body from standard input
{<specification}	program will read the source of the specification from standard input
...	same for other data contents (see above)
{>source}	program will write the source of the body to standard output
{>specification}	program will write the source of the specification to standard output
...	same for other data contents (see above)

Other input/output redirection may be used in the usual way, but only one input redirection or output redirection, resp., may be used in a command call.

Examples:

```
mdb iterate “\apm\fltsys...” –sel “ucl*” \
            –exec ’cls_editor {} –make –batch’
```

This command calls the CLS Editor for all end items whose item type starts with “UCL” in the specified subtree, i. e. it will recompile all UCL APs and libraries in that subtree in make mode. This emulates the MDB “batch compilation”.

```
mdb iterate “\apm\fltsys...” –sel “ucl*” –exec ’echo {sid}, {path}’
```

This command creates a list of pathnames together with their SID, using the UNIX `echo` command.

```
mdb iterate “\apm\fltsys...” –sel “ucl*” \
            –exec ’sed s/COF/COL/g {source} {>source}’
```

This command uses the UNIX `sed` command to replace all occurrences of “COF” by “COL” in all selected UCL compilation units. The modified source text, written by `sed` on standard output, is stored back in the database.