FLAMCLE/P-API

1

Generated by Doxygen 1.8.20

Chapter 1

LIMES Command Line Executor and Processor (FLAMCLEP)

1.1 License

This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

Permission is granted to anyone to use this software for any purpose, including commercial applications, and to alter it and redistribute it freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not claim that you wrote the original software. If you use this software in a product, an acknowledgment in the product documentation would be appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

If you need professional services or support for this library please contact support@flam.de.

1.2 Overview

We developed CLE/P because we didn't find an existing library meeting our requirements for a consistent command line over all platforms. We decided therefore to build such a compiler which is table-controlled and which transforms property files and command line inputs into a machine-processable data structure of arbitrary nesting depth.

The result is a very powerful tool briefly described in the following. The full interface documentation, programming reference, the GIT-repository, and the license can be downloaded from GITHUB.

We are posting CLP/E as OpenSource on the terms of a Zlib-based license above, making it freely available to everyone in the form of a GIT project.

With the command line executor (CLE), you can simply realize a complex command line through the definition of some tables. No programming is required to get the values parsed and stored in a free defined data structure. For this, a compiler with its own language was implemented to provide the same command line interface on each

available platform. This command line parser (CLP) is used by the CLE. Both components provide extensive help for each command and many other support functions. For a provided list of commands, the CLE uses a dedicated command line processor (CLP) to make all these commands available, together with several built-in functions.

To achieve this, a table must be defined where each row describes one command. This table provides the input for the execution function doing the command line interpretation. The whole library consists of only one function and a structure to define the command table.

Beside the specified user defined commands, the CLE provides several powerful built-in functions. All built-in functions contain a manual page to get more information at runtime.

Based on the keyword, the short help message and the detailed description, the built-in function GENDOCU or HTMLDOC can be used to generate a complete user manual. Based on this capability, the CLE/P completely describes itself.

The self explanation of the whole program was one of the main targets of this general command line interface. To understand this interface specification it is advisable to read the CLE/P documentation.

1.3 Compilation concerns

For compilation the defines below must be set:

```
__DEBUG__         for a debug build
__RELEASE__       for a release build
__WIN__           for WINDOWS platforms
__ZOS__           for ZOS mainframe platforms
__USS__           for unix system services (USS) on ZOS mainframe platforms
__BUILDNR__       to define the build number (integer, default is 0)
__BUILD__         to define the build string ("debug", "release", "alpha", ...)
__HOSTSHORTING__ to short function names to 8 character for mainframes
```

On z/OS or USS the CELP and the using project must be compiled with the same CONVLIT() parameter (we recommend IBM-1047) for correct conversion of the literals. Don't use the literal replacements (S_xxx or C_xxx or the exprintf() functions) in front of the CleExecute call (see siCleExecute()) to ensure the environment of the configuration file is used for character conversion.
Few of the strings provided to the CleExecute function are converted for EBCDIC system. These conversions are done, based on stack allocations. We expect string literals with a limited length for these values. Please be aware of the security issues if you provide variable length strings in this case.

1.4 Command Line Executor (FLAMCLE)

With the command line executor (FLAMCLE) the user can simply realize a complex command line using command and keyword definitions contained in some tables.
Command Line Executor (FLAMCLE) uses the Command Line Parser (FLAMCLP) to provide the selected commands and keywords on each used platform. To achieve this, a table is defined where each row describes one command or keyword. This table provides the input for the execution function doing the command line interpretation. The whole library consists of only one function and a structure to define the command table. One of these commands or a built-in function can be defined as default, which will be executed if the first keyword (argv[1]) don't fit one of the user- defined commands or built-in functions. If no command or built-in function is defined and no default set the built-in function syntax will be executed to show the capabilities of the command line program.
Beside the specified user-defined commands, the FLAMCLE provides several powerful built-in functions (listed below). All built-in functions have a manual page, implemented to display more information at runtime.
With the built-in function help a short help message or the detailed description can determined for each parameter and command using a dotted path. The built-in function GENDOCU can be used to generate a part or the complete user manual. Based on this capability the FLAMCLE completely describes itself.
The self-documenting style of the whole program was one of the main targets of this general command line interface. To understand the interface specification, it is recommended to read the FLAMCLE documentation.

1.4.1 FLAMCLE-Features

Below, you can find a possibly incomplete list of FLAMCLE feature:

- Support of an unlimited amount of commands
- Support of hidden commands (not documented)
- Support of a default command (optional)
- Includes a lot of useful built-in functions
- Simple owner management to differentiate configurations
- The logical program name can be freely defined
- Different view for property and command line parsing
- Case sensitive or in-sensitive command line interpretation
- Output file can be defined (stdout, stderr, or a real file)
- Complete trace file management for FLAMCLP and commands
- The look and feel can be defined freely
- Syntax, help and manpage support for program, commands and arguments
- Extensive documentation generation in ASCIIDOC format for the user manual
- Powerful property file management (generation, activation, update, ...)
- Simple configuration data management (own environment variables)
- Environment variable replacement in the command string('<'envar'>')
- Automatic keyword shortening for arguments
- Support for many data types, like:
  - Number (decimal, hexadecimal, octal, binary and time)
  - Float (decimal in all variants)
  - String (binary text/ASCII/EBCDIC/HEX or from a file (for passwords))
  - Object (Structure) with parameter file support
  - Overlay (Union) with parameter file support
  - Array (List (realized as simplified notation) with parameter file support)
- Support of constant definitions used as selection of values over keywords
- Internal calculated values are available as link (amount of values in an array, length of a string, object identifier in overlays, ...)
- The main table for a command can be defined as object or overlay
- Keyword, help message and detailed description can be freely defined for the program, each command, argument or constant definition
- Aliases for each argument can also be defined and are handled as options for the same value.
- Available and usable on each platform including WIN, UNIX, USS, ZOS, ...
- Support of 'STDENV' as DD name or DSN '&SYSUID..STDENV' for environment variables on mainframes
- Support property definitions over environment variables to override hard coded default properties
- Keywords (commands, built-in functions, ON, OFF, ALL, DEPTH1, ...) can start optional with "-" or "--"

- Support for parameter files per command, object, overlay or array
- File name mapping and DD:NAME support (see <<CLEP.CLEPMAIN,CLEP>>)
- Return/condition/exit code and reason code handling
- On EBCDIC systems we use a code page specific interpretation of punctuation characters **(!$#@[]^`{|}~)** dependent on the environment variable LANG
- Extensive manual page management including replacement of owner (&{OWN}) and program name (&{PGM})
- Own tool to generate description strings from text files including replacement of constant definitions (${**VERSION**})
- Definition of maximum and minimum condition code (MAXCC) for command execution
- Support SILENT and QUITE to control outputs
- Special condition code handling (incl. description for manual and built-in function ERRORS)
- Strings can be read from files to increase protection and prevent logging of passwords
- Default parameter file name for system supporting static file allocation ("DD:MYPAR")
- You can exclude the run after mapping if you provide the corresponding return code (siNoR)
- Own file to string callback function for parameter files

1.4.2 FLAMCLE-Built-in Functions

All these built-in functions are available:

- SYNTAX - Provides the syntax for each command
- HELP - Provides quick help for arguments
- MANPAGE - Provides manual pages (detailed help)
- GENDOCU - Generates auxiliary documentation
- HTMLDOC - Generates documentation using a call pack interface for each page
- GENPROP - Generates a property file
- SETPROP - Activates a property file
- CHGPROP - Updates property values in the current property file
- DELPROP - Removes a property file from configuration
- GETPROP - Displays current properties
- SETOWNER - Defines the current owner
- GETOWNER - Displays current owner setting
- SETENV - Defines environment variables in the config file
- GETENV - Displays the environment variables set in the config file
- DELENV - Deletes environment variables in the config file
- TRACE - Manages trace capabilities
- CONFIG - Displays or clears the current configuration settings
- GRAMMAR - Displays the grammar for commands and properties

- LEXEM - Displays the regular expressions accepted in a command
- LICENSE - Displays the license text for the program
- VERSION - Lists version information for the program
- ABOUT - Displays information about the program
- ERRORS - Displays information about return and reason codes of the program

To read the manual page, please use:

```
program MANPAGE function
```

Below, you can find the syntax for each built-in function:

- SYNTAX [command[.path] [DEPTH1 | ... | DEPTH9 | ALL]]
- HELP [command[.path] [DEPTH1 | ... | DEPTH9 | ALL]] [MAN]
- MANPAGE [function | command[.path][=filename]] | [filename]
- GENDOCU [command[.paht]=]filename [NONBR] [SHORT]
- GENPROP [command=]filename
- SETPROP [command=]filename
- CHGPROP command [path[=value]]
- DELPROP [command]
- GETPROP [command[.path] [DEPTH1 | ... | DEPTH9 | DEPALL | DEFALL]]
- SETOWNER name
- GETOWNER
- SETENV variable=value
- GETENV
- DELENV variable
- TRACE ON | OFF | FILE=filename
- CONFIG [CLEAR]
- GRAMMAR
- LICENSE
- LEXEM
- VERSION
- ABOUT
- ERRORS

1.4.3 FLAMCLE-Sample program

This sample program is the main of our FLCL command line utility. This code is not functional and results in compile errors because of missing other parts of the FL5 project, but it visible the principles of CLE usage.

```
#include "CLEPUTL.h"
#include "FLAMCLE.h"
#include "CLEMAN.h"
#define DEFINE_STRUCT
#include "CLPMAC.h"
#include "FL5TAB.h"
#include "FL5STC.h"
int main(const int argc, const char * argv[])
{
   static TsFlcConvPar     stFlcConvPar;
   static TsClpConvPar     stClpXcnvPar;
   static TsClpConvPar     stClpConvPar;
   static TsClpIcnvPar     stClpIcnvPar;
   static TsFlcInfoPar     stFlcInfoPar;
   static TsClpInfoPar     stClpInfoPar;
   #undef DEFINE_STRUCT
   #include "CLPMAC.h"
   #include "FL5CON.h"
   #include "FL5ARG.h"
   CLECMD_OPN(asCmdTab) = {

       CLETAB_CMD("CONV",asClpConvPar,&stClpConvPar,&stFlcConvPar,NULL,siIniConv,siMapConv2Conv,siFluc,siFinConv,1,MAN_FLCL_C
       data conversion")

       CLETAB_CMD("XCBV",asClpXcnvPar,&stClpXcnvPar,&stFlcConvPar,NULL,siIniXcnv,siMapXcnv2Conv,siFluc,siFinConv,0,MAN_FLCL_X
       data conversion")

       CLETAB_CMD("ICNV",asClpIcnvPar,&stClpIcnvPar,&stFlcConvPar,NULL,siIniIcnv,siMapIcnv2Conv,siFluc,siFinConv,1,MAN_FLCL_I
       conversion with libiconv")

       CLETAB_CMD("INFO",asClpInfoPar,&stClpInfoPar,&stFlcInfoPar,NULL,siIniInfo,siMapInfo2Info,siInfo,siFinInfo,1,MAN_FLCL_IN
       information")
      CLECMD_CLS
   };
   CLEOTH_OPN(asOthTab) = {
      CLETAB_OTH("flcbyt","READ-FORMAT"  ,asClpWrtFmtPar,MAN_FLCBYT_READ_FORMAT ,HLP_FLCBYT_READ_FORMAT
       ,TRUE)
      CLETAB_OTH("flcbyt","WRITE-FORMAT"
       ,asClpRedFmtPar,MAN_FLCBYT_WRITE_FORMAT,HLP_FLCBYT_WRITE_FORMAT,TRUE)
      CLETAB_OTH("flcbyt","CONV-READ"    ,asClpElmCnvRed,MAN_FLCBYT_CONV_READ   ,HLP_FLCBYT_CONV_READ
       ,TRUE)
      CLETAB_OTH("flcbyt","CONV-WRITE"   ,asClpElmCnvWrt,MAN_FLCBYT_CONV_WRITE  ,HLP_FLCBYT_CONV_WRITE
       ,TRUE)
      CLETAB_OTH("flcbyt","FROM-TO-CONV" ,asClpElmCnv   ,MAN_FLCBYT_CONV        ,HLP_FLCBYT_CONV
       ,TRUE)
      CLETAB_OTH("flcbyt","STATE"        ,asClpExtPar   ,MAN_FLCBYT_STATE       ,HLP_FLCBYT_STATE
       ,FALSE)
      CLETAB_OTH("flcbyt","LOG"          ,asClpMemoryLog,MAN_FLCBYT_LOG         ,HLP_FLCBYT_LOG
       ,FALSE)
      CLEOTH_CLS
   };
   CLEDOC_OPN(asDocTab) = {
      CLETAB_DOC(CLE_DOCTYP_COVER         ,1,NULL     ,NULL               ,NULL
       ,"FLCL manual"                       ,MAN_FLCL_COVER                         ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,NULL     ,NULL               ,NULL
       ,"Trademarks"                        ,MAN_FLCL_TRADEMARKS                    ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,NULL     ,NULL               ,NULL
       ,"Abstract"                          ,MAN_FLCL_ABSTRACT                      ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,NULL     ,NULL               ,NULL
       ,"Supported systems"                 ,MAN_FLCL_SUPPORTED_SYSTEMS             ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,NULL     ,NULL               ,NULL
       ,"Use cases"                         ,MAN_FLCL_USECASES                      ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,NULL     ,CLE_DOCKYW_PREFACE ,NULL
       ,"Preface"                           ,MAN_FLCL_PREFACE                       ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,"1."     ,NULL               ,"clep.main"
       ,"Command line parser"               ,MAN_CLE_CLEPMAIN                       ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"1.1."   ,NULL               ,NULL
       ,"Command line considerations"       ,MAN_CLE_CLEPMAIN_CONSID                ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"1.2."   ,NULL               ,"clep.main.usedenv"
       ,"Used environment variables"        ,MAN_CLE_CLEPMAIN_USEDENV               ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"1.3."   ,NULL               ,NULL
       ,"Environment variable mapping"      ,MAN_CLE_CLEPMAIN_ENVARMAP              ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"1.4."   ,NULL               ,"clep.main.filemap"
       ,"Filename mapping"                  ,MAN_CLE_CLEPMAIN_FILEMAP               ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"1.5."   ,NULL               ,NULL
       ,"Key label name mapping"            ,MAN_CLE_CLEPMAIN_KEYLABMAP             ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"1.6."   ,NULL               ,"clep.main.ebcdic"
       ,"Special EBCDIC code page support"  ,MAN_CLE_CLEPMAIN_EBCDIC                ,NULL)
      CLETAB_DOC(CLE_DOCTYP_BUILTIN       ,3,"1.7."   ,NULL               ,CLE_ANCHOR_BUILTIN_FUNCTIONS
       ,"Built-in functions"                ,MAN_CLE_BUILTIN_FUNCTIONS              ,NULL)
      CLETAB_DOC(CLE_DOCTYP_PROGRAM       ,2,"2."     ,NULL               ,NULL
       ,"FLCL Utility"                      ,MAN_FLCL_MAIN                          ,NULL)
```

```
      CLETAB_DOC(CLE_DOCTYP_PGMSYNOPSIS   ,3,"2.1."   ,NULL                ,NULL
       ,"Synopsis"                          ,MAN_CLE_PROGRAM_SYNOPSIS                ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.2."   ,NULL                ,NULL
       ,"Environment Variables"             ,MAN_FLCL_MAIN_ENVIRONMENT_VARIABLES     ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.3."   ,NULL                ,NULL
       ,"Filename Handling"                 ,MAN_FLCL_MAIN_FILENAME_HANDLING         ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.4."   ,NULL                ,NULL
       ,"Directory Support"                 ,MAN_FLCL_MAIN_DIRECTORY_SUPPORT         ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.5."   ,NULL                ,NULL
       ,"Input to Output Name Mapping"      ,MAN_FLCL_MAIN_INPUT_TO_OUTPUT_NAME_MAPPING,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.6."   ,NULL                ,NULL
       ,"Key Label Handling"                ,MAN_FLCL_MAIN_KEY_LABEL_HANDLING        ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.7."   ,NULL                ,NULL
       ,"Password Handling"                 ,MAN_FLCL_MAIN_PASSWORD_HANDLING         ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.8."   ,NULL                ,NULL
       ,"Handling of Empty Records"         ,MAN_FLCL_MAIN_HANDLING_OF_EMPTY_RECORDS ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.9."   ,NULL                ,NULL
       ,"Table Support"                     ,MAN_FLCL_MAIN_TABLE_SUPPORT             ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.10."  ,NULL                ,NULL
       ,"Pre- and Post-processing"          ,MAN_FLCL_MAIN_PRE_AND_POST_PROCESSING   ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.11."  ,NULL                ,NULL
       ,"Use of Built-in Functions"         ,MAN_FLCL_MAIN_BUILT_IN_FUNCTIONS        ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.12."  ,NULL                ,NULL
       ,"CONV versus XCNV and ICNVs"        ,MAN_FLCL_MAIN_CONV_VERSUS_XCNV_AND_ICNV ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.13."  ,NULL                ,NULL
       ,"CONV READ/WRITE overview"          ,MAN_FLCL_MAIN_CONV_READ_WRITE_OVERVIEW  ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.14."  ,NULL                ,NULL
       ,"JCL Considerations"                ,MAN_FLCL_MAIN_JCL_CONSIDERATIONS        ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.15."  ,NULL                ,NULL
       ,"SAF Consideration"                 ,MAN_FLCL_MAIN_SAF_CONSIDERATIONS        ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,3,"2.16."  ,NULL                ,NULL
       ,"Installation"                      ,MAN_FLCL_MAIN_INSTALLATION              ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,4,"2.16.1.",NULL                ,NULL
       ,"License"                           ,MAN_FLCL_MAIN_INSTALLATION_LICENSE      ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,4,"2.16.2.",NULL                ,NULL
       ,"Download"                          ,MAN_FLCL_MAIN_INSTALLATION_DOWNLOAD     ,NULL)
      CLETAB_DOC(CLE_DOCTYP_PGMSYNTAX     ,3,"2.17."  ,NULL                ,NULL
       ,"Syntax of FLCL"                    ,MAN_CLE_PROGRAM_SYNTAX                  ,NULL)
      CLETAB_DOC(CLE_DOCTYP_PGMHELP       ,3,"2.18."  ,NULL                ,NULL
       ,"Help for FLCL"                     ,MAN_CLE_PROGRAM_HELP                    ,NULL)
      CLETAB_DOC(CLE_DOCTYP_COMMANDS      ,3,"2.19."  ,NULL                ,NULL
       ,"Available FLCL commands"           ,MAN_CLE_PROGRAM_COMMANDS                ,NULL)
      CLETAB_DOC(CLE_DOCTYP_OTHERCLP      ,2,NULL     ,CLE_DOCKYW_APPENDIX,NULL
       ,"Other CLP strings"                 ,MAN_FLCL_OTHERCLP                       ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,NULL     ,CLE_DOCKYW_APPENDIX,NULL
       ,"FLUC Filesystem for Linux"         ,MAN_FLCL_APPENDIX_FLUCFS                ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,NULL     ,CLE_DOCKYW_APPENDIX,NULL
       ,"FLUC Subsystem for z/OS"           ,MAN_FLCL_APPENDIX_FLUCSUB               ,NULL)
      CLETAB_DOC(CLE_DOCTYP_LEXEM         ,2,NULL     ,CLE_DOCKYW_APPENDIX,CLE_ANCHOR_APPENDIX_LEXEM
       ,"Lexemes"                           ,MAN_CLE_APPENDIX_LEXEM                  ,NULL)
      CLETAB_DOC(CLE_DOCTYP_GRAMMAR       ,2,NULL     ,CLE_DOCKYW_APPENDIX,CLE_ANCHOR_APPENDIX_GRAMMAR
       ,"Grammar"                           ,MAN_CLE_APPENDIX_GRAMMAR                ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,NULL     ,CLE_DOCKYW_APPENDIX,CLE_ANCHOR_APPENDIX_PROPERTIES
       ,"Properties"                        ,MAN_CLE_APPENDIX_PROPERTIES             ,NULL)
      CLETAB_DOC(CLE_DOCTYP_PROPREMAIN    ,3,NULL     ,NULL                ,NULL
       ,"Remaining documentation"           ,MAN_CLE_APPENDIX_PROP_REMAINING         ,NULL)
      CLETAB_DOC(CLE_DOCTYP_PROPDEFAULTS  ,3,NULL     ,NULL                ,NULL
       ,"Predefined defaults"               ,MAN_CLE_APPENDIX_PROP_DEFAULTS          ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,NULL     ,CLE_DOCKYW_APPENDIX,CLE_ANCHOR_APPENDIX_RETURNCODES
       ,"Return codes"                      ,MAN_CLE_APPENDIX_RETURNCODES            ,NULL)
      CLETAB_DOC(CLE_DOCTYP_SPECIALCODES  ,3,NULL     ,NULL                ,NULL
       ,"Special condition codes"           ,MAN_FLCL_SPECIALCODES                   ,NULL)
      CLETAB_DOC(CLE_DOCTYP_REASONCODES   ,3,NULL     ,NULL                ,CLE_ANCHOR_APPENDIX_REASONCODES
       ,"Reason codes"                      ,MAN_CLE_APPENDIX_REASONCODES            ,NULL)
      CLETAB_DOC(CLE_DOCTYP_VERSION       ,2,NULL     ,CLE_DOCKYW_APPENDIX,CLE_ANCHOR_APPENDIX_VERSION
       ,"Version"                           ,MAN_CLE_APPENDIX_VERSION                ,NULL)
      CLETAB_DOC(CLE_DOCTYP_ABOUT         ,2,NULL     ,CLE_DOCKYW_APPENDIX,CLE_ANCHOR_APPENDIX_ABOUT
       ,"About"                             ,MAN_CLE_APPENDIX_ABOUT                  ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,NULL     ,CLE_DOCKYW_GLOSSARY,NULL
       ,"Glossary"                          ,MAN_FLCL_GLOSSARY                       ,NULL)
      CLETAB_DOC(CLE_DOCTYP_CHAPTER       ,2,NULL     ,CLE_DOCKYW_COLOPHON,NULL
       ,"Imprint"                           ,MAN_FLCL_IMPRINT                        ,NULL)
      CLEDOC_CLS
   };
   TsFl5Gbl*   psGbl=psFl5GblOpn(sizeof(acMsg),acMsg);
   siErr=siCleExecute(psGbl,asCmdTab,argc,argv,FLM_CLEP_DEFAULT_OWNER,"flcl",
            "limes datentechnik(R) gmbh","support@flam.de",
            FLM_CLEP_CASE_SENSITIVITY,TRUE,TRUE,FALSE,FLM_CLEP_MINIMAL_KEYWORDLEN,
            pfErr,pfTrc,FLM_CLEP_DEPTH_STRING_1047,FLM_CLEP_OPTION_STRING,
            FLM_CLEP_SEPARATION_STRING,psMain->acLicTxt,
            FLM_VSN_STR"-"__BUILDNRSTR__,psMain->acVersion,psMain->acAbout,
            "Frankenstein Limes(R) Command Line for FLUC, FLAM and FLIES",
            NULL,pcFlmErrors,asOthTab,NULL,siClpFile2String,
            NULL,(psGbl->isSafClpControl)?siClpSaf:NULL,pcDpa,0,asDocTab);
   vdFl5GblCls(psGbl);
}
```

1.5 Command Line Parser (FLAMCLP)

The command line parser (FLAMCLP) is a complier which reads a command string using the lexems and grammar below to fill a structure with the corresponding values given in this line. The FLAMCLP works only in memory (except parameter files are used for objects, overlays, arrays, arguments or string files) and the syntax and semantic will be defined by a tree of tables. Such a table can represent an object (struct) or an overlay (union). Each argument in such a table can be a object or overlay again in using another table for this type. Basic types are switches, numbers, floats or strings (time and date are implemented as number in seconds from 1970). With each argument you can define the required minimum and possible maximum amount of occurrences. This means that each argument can be an array and arrays are implemented as simplified notations. Arrays and strings can be a fixed length part of the data structure or dynamic allocated by CLP. In the last case, the fix part of the data structure is a pointer to the dynamic allocated data area (use '->' instead of '.'). All dynamic allocated data blocks are managed by CLP. If you close the CLP you can define if anything including the dynamic parts of the CLP structure is closed. Or anything is freed except the dynamic blocks allocated for the CLP structure. In this case you can keep the CLP handle open, to free the remaining buffers later or you can close the CLP handle and the dynamic allocated memory of the CLP structure must be free by the application.
For object, overlays and arrays you can provide parameter files (OBJECT='filename') containing the parameter string in the corresponding syntax for these object, overlay or array (KYW[='filename']). With '=>' you can also use parameter files for normal arguments. The operator '=>' is also permitted for objects, overlays and arrays.
To read such a parameter file as string into the memory a handle and a callback function can be provided. If the parameter NULL a default implementation is used. If you provide your own function you can include for example URL support, remote access, character conversion, etc. The handle is given to the callback function. The default implementation don't need any handle, but you can use it for example for the character conversion module, a remote session or something else.
To handle passwords and passphrase more secure, you can provide a filename as string (PASSWD=f'filename'), which contains the corresponding string value. This prevents for example passwords from logging.
An optional callback function with handle for additional authorization checking can be provided. The resource will be each path written to the CLP structure. If the pointer to the callback function is NULL then the function is not called. This feature is mainly for RACF on z/OS.
To support critical punctuation characters on EBCDIC systems a complex support was implemented to make the whole source independent of the used EBCDIC code page. The code page to use must be defined in the environment variable LANG or just for CLP strings with the environment variable CLP_STRING_CCSID or inside the CLP string ("&nnnn;"). Last but not least single character escaping ("&xxx;") is supported as well.
In the command string (everywhere, where the scanner start to read a lexem) each value in angle brackets will be transparently replaced by the corresponding environment variable, except in strings.
The FLAMCLP uses these tables as symbol tables to define the syntax and semantic of a command. The same table provides the offset used to store the parsed values. This offset occurs in a real data structure and with CLPMAC.h you can use the same macro to build the tables and corresponding structures and unions. This is not mandatory, but we recommend to use the macro in order to be in sync.
The FLAMCLP provides also all internally calculated values in this data structure. The mechanism is called linking. Thus you have to use the same keyword for linking eventually with a calculated value of that argument. For example, if you define an array of numbers then you can define a link to determine the amount of entered numbers or for an overlay you can link the corresponding object identifier to determine which of the arguments are chosen by the user. If you define overlays of overlays an additional dimension for each level is used. In this case you must define an array for this link and you get the child (lnk[0]) before the parent (lnk[1]) written in the CLP structure. If the OID is 0, then it will not be add to the array. This is useful if the OIDs of the children are already unique.
You can also get the string length and other features. The kind of link is defined over the flags field. There are a lot of other flags supported beside links, for example the PWD flag, which tells CLP that this value are only clear in the data structure but always obfuscated in logs, traces and other printouts to keep the value secret. Another flag can be used for numbers. With CLPFLG_DEF you can activate a extension of the syntax. If this flag used for a number then the object identifier is assigned as value if no assignment done for this number. This means that with this extended syntax you can define a switch, which you can assign a number. This is useful for example to activate a feature with a default value by using only the key word and the user can change the default value by an optional assignment of another value.
The FLAMCLP also supports aliases. An alias points to another argument and is only an additional keyword that can be used. The maximum length of a keyword or alias cannot exceed 63 character.
To be compatible with certain shells the features below are implemented.

- Strings can be enclosed with " or "" or ``

- Strings can also be defined without quotes
- Explicit keywords can start with "-" or "--" in front of the qualifier
- If it is unique then parenthesis and the dot can be omitted for objects and overlays
- On EBCDIC systems we use a code page specific interpretation of punctuation characters

Besides arguments you can also have a constant definition for selections. A feature is useful in order to define keywords for values (numbers, floats and strings). With help of the selection flag you can enforce the pure acceptance of predefined keywords.
Additional hard coded key words (see lexems) can be used in constant expressions to build values and strings (value=64KiB).
For each argument or constant you must define a keyword and a short help message. If you provide a detailed description, then this argument becomes an own chapter in the generated documentation, a manual page will be available and extensive help is displayed. The description string can contain &{OWN} for the current owner or &{PGM} for the current program name. The letter case of the replacement string depends and the letter case of the keyword: PGM = upper case, pgm = lower case, Pgm = title case, pGm = original string. All other content inside of &{...} is ignored. This can be used, for example, to insert comments into the source of the manual page.
For each argument you can define a default value and use the property parser or environment variables to overwrite it again. The default value replaces the entered value. This means that if a default value, environment variable or property is defined, then this will have the same effect as the entry of the value in the command line. With the latter you can still override the hard coded or property default value. The property management can make use of a function that extracts a property list for the argument table tree.
For each path you can also define the default value as environment variable. The path are prefixed with the owner ID and the program name first, then only the program name and at the last the path only starting with the command name will be use to determine a environment variable. For this the path is converted to upper case and all '.' are replaced by '_'. The value of the environment variable must contain the same supplement string which are required for the property definition. All possible path values can be determine with the property generation function.
With the CLP flags CMD (for command) and PRO (property) you can define if a parameter is only visible in the command line or property file. These flags have no influence of property or command line parsing. It only reflects the online help/syntax and docu/property generation. This means that you can still use such a parameter in the property file or in the command line, but it is not directly visible to the user. If the flags CMD and PRO are not set then the parameter will be visible in both areas. With the flag DMY (for dummy) you can enforce that this parameter is not visible in a generated property file, on the command line help, syntax and documentation. In this case, the parameter is no part of the symbol table. It is only part of the CLP structure.
For binary strings the default interpretation can be free defined over a additional set of flags (CLPFLG_HEX/CHR/ASC/EBC). This is useful for hex strings or passwords. If you want use arrays in overlays you cannot use a link to determine the count or length. In this case you can use the DLM flag. In this case for fix size types an additional empty element are used as delimiter. For the static case the max count are reduced by 1 and in the dynamic case one additional element is allocated to determine the end of the array. For variable (CLPFLG_FIX is not defined) strings the end of the list of strings are marked with 0xFF.
The FLAMCLP calculates automatically the minimum amount of letters required to make the meaning of a keyword unique. Depending on the case mode the required letters are highlighted in the interactively used help function. The syntax function provides also data when an argument is required or optional, based on the minimum amount of occurrences.
If you intend to apply the FLAMCLP first of all an open will be necessary. Then you are able to parse a property list before doing this with the command line. Both property list and command line are provided as zero terminated strings. This means that the FLAMCLP does not know whether the command line results from a file or argc/argv.
If the isPfl (is parameter file) flag TRUE: For objects, overlays and arrays you can use the assignment letter '=' or '=>' to define a parameter file containing the command string for this object, overlay or array. For simple arguments you must use '=>' to define a parameter file but all these capabilities are only supported if the flag defined to true. This means that for each object, overlay, array or argument a dedicated parameter file can be used. The parameter file must contain a command string which syntax is valid for the certain object, overlay, array or argument. CLP open the file with format string "r". To use DD names on mainframes the file name must like "DD:name".
If the flag CLPFLG_PWD is used, string outputs containing passwords will result in "###SECRECT###" and float or number outputs in a value of 0.
For zero terminated strings in local character set (s'...') several special mapping and conversions can be activated over the flags CLPFLG_FIL/LAB/UPP. The replacement of environment variables is done for each string but

you can also activate prefix adjustment and tilde replacement for files, and tilde, circumflex and exclamation mark replacement for key labels. Additional you can ensure that each such string are converted to upper case.
Parsing of the properties (can be done a lot of times over different sources) only change the default values in the symbol table and has no effect for the CLP structure. First after parsing the command line the corresponding FLAMCLP structure is filled with the properties or entered values and the FLAMCLP can be closed or another command line parsed.
Attention: If pointer to values in the CLP handle used (ppLst, psErr) then you cannot close the CLP or you must copy the values before.
The normal procedure to use the CLP:

```
ClpOpen()
ClpParsePro()
ClpParseCmd()
ClpClose()
```

Beside property and command line parsing the FLAMCLP offers an interactive syntax and help function. Additionally, you can use a very powerful function to generate single manual pages or complete user manuals, You can make use of the supported grammar and regular expressions (lexems). Provided manual pages must be in ASCIIDOC and will be converted on EBCDIC systems from the compile code page in the local code page.
Only ClpParseCmd() uses the pvDat pointer. All other functions only work on the symbol table. This means if you don't use ClpParseCmd() the pointer to the CLP structure (pvDat), it can be NULL. This is useful if only help, syntax, documentation or property management are required. For these functions no corresponding CLP structure must be allocated.
The implementation of the FLAMCLP is finished with the Command Line Executor (FLAMCLE) with which you can define your list of commands by using an additional table. You can make use of only one new function that is executed eventually. The FLAMCLE offers an extensive built-in functionality and is the optimal access method to the FLAMCLP capabilities.
Additional there is an interface to browse the symbol table. These interface can for example used to build several graphical user interfaces or other things based on the tables.

1.5.1 FLAMCLP-Lexemes

Call siClpLexemes() to get the current supported lexemes. The list below could be a older state of the implementation.
Lexemes (regular expressions) for argument list or parameter file:

```
--| COMMENT   '#' [:print:]* '#'                              (will be ignored)
--| LCOMMENT  ';' [:print:]* 'nl'                             (will be ignored)
--| SEPARATOR [:space: | :cntr: | ',']*                  (abbreviated with SEP)
--| OPERATOR '=' | '.' | '(' | ')' | '[' | ']' | (SGN, DOT, RBO, RBC, SBO, SBC)
--|  '=>'| '+' | '-' | '*' | '/' | '{' | '}' (SAB, ADD, SUB, MUL, DIV, CBO,CBC)
--| KEYWORD   ['-'['-']][:alpha:]+[:alnum: | '_']*          (always predefined)
--| NUMBER    ([+|-]  [ :digit:]+)  |                       (decimal (default))
--| num       ([+|-]0b[ :digit:]+)  |                                  (binary)
--| num       ([+|-]0o[ :digit:]+)  |                                   (octal)
--| num       ([+|-]0d[ :digit:]+)  |                                 (decimal)
--| num       ([+|-]0x[ :xdigit:]+) |                             (hexadecimal)
--| num       ([+|-]0t(yyyy/mm/tt.hh:mm:ss)) |  (relativ (+|-) or absolut time)
--| FLOAT     ([+|-]  [ :digit:]+.[:digit:]+e|E[:digit:]+) | (decimal(default))
--| flt       ([+|-]0d[ :digit:]+.[:digit:]+e|E[:digit:]+)            (decimal)
--| STRING         ''' [:print:]* ''' |          (default (if binary c else s))
--| str       [s|S]''' [:print:]* ''' |                (null-terminated string)
--| str       [c|C]''' [:print:]* ''' |  (binary string in local character set)
--| str       [a|A]''' [:print:]* ''' |                (binary string in ASCII)
--| str       [e|E]''' [:print:]* ''' |               (binary string in EBCDIC)
--| str       [x|X]''' [:print:]* ''' |         (binary string in hex notation)
--| str       [f|F]''' [:print:]* ''' | (read string from file (for passwords))
--|           Strings can contain two '' to represent one '.
--|           Strings can also be enclosed in " or ` instead of '.
--|           Strings can directly start behind a '=' without enclosing ('`").
--|           In this case the string ends at the next separator or operator
--|           and keywords are preferred. To use keywords, separators or
--|           operators in strings, enclosing quotes are required.
--|
--| The predefined constant keyword below can be used in a value expressions
--| NOW       NUMBER - current time in seconds since 1970 (+0t0000)
--| MINUTE    NUMBER - minute in seconds (60)
--| HOUR      NUMBER - hour in seconds   (60*60)
--| DAY       NUMBER - day in seconds    (24*60*60)
```

```
--| YEAR      NUMBER - year in seconds   (365*24*60*60)
--| KiB       NUMBER - kilobyte          (1024)
--| MiB       NUMBER - megabyte          (1024*1024)
--| GiB       NUMBER - gigabyte          (1024*1024*1024)
--| TiB       NUMBER - terrabyte         (1024*1024*1024*1024)
--| RNDn      NUMBER - simple random number with n * 8 bit in length (1,2,4,8)
--| PI        FLOAT  - PI (3.14159265359)
--| LCSTAMP   STRING - current local stamp in format:           YYYYMMDD.HHMMSS
--| LCDATE    STRING - current local date in format:            YYYYMMDD
--| LCYEAR    STRING - current local year in format:            YYYY
--| LCYEAR2   STRING - current local year in format:            YY
--| LCMONTH   STRING - current local month in format:           MM
--| LCDAY     STRING - current local day in format:             DD
--| LCTIME    STRING - current local time in format:            HHMMSS
--| LCHOUR    STRING - current local hour in format:            HH
--| LCMINUTE  STRING - current local minute in format:          MM
--| LCSECOND  STRING - current local second in format:          SS
--| GMSTAMP   STRING - current Greenwich mean stamp in format:  YYYYMMDD.HHMMSS
--| GMDATE    STRING - current Greenwich mean date in format:   YYYYMMDD
--| GMYEAR    STRING - current Greenwich mean year in format:   YYYY
--| GMYEAR2   STRING - current Greenwich mean year in format:   YY
--| GMMONTH   STRING - current Greenwich mean month in format:  MM
--| GMDAY     STRING - current Greenwich mean day in format:    DD
--| GMTIME    STRING - current Greenwich mean time in format:   HHMMSS
--| GMHOUR    STRING - current Greenwich mean hour in format:   HH
--| GMMINUTE  STRING - current Greenwich mean minute in format: MM
--| GMSECOND  STRING - current Greenwich mean second in format: SS
--| GMSECOND  STRING - current Greenwich mean second in format: SS
--| SnRND10   STRING - decimal random number of length n (1 to 8)
--| SnRND16   STRING - hexadecimal random number of length n (1 to 8)
--|
--| SUPPLEMENT     '"' [:print:]* '"' |   (null-terminated string (properties)).
--|           Supplements can contain two "" to represent one ".
--|           Supplements can also be enclosed in ' or ` instead of ".
--|           Supplements can also be enclosed in ' or ` instead of ".
--| ENVIRONMENT VARIABLES '\<'varnam'\>' will replaced by the corresponding value
--| Escape sequences for critical punctuation characters on EBCDIC systems
--|    '!' = '\&EXC;'   - Exclamation mark
--|    '$' = '\&DLR;'   - Dollar sign
--|    '#' = '\&HSH;'   - Hashtag (number sign)
--|    '@' = '\&ATS;'   - At sign
--|    '[' = '\&SBO;'   - Square bracket open
--|    '\' = '\&BSL;'   - Backslash
--|    ']' = '\&SBC;'   - Square bracket close
--|    '^' = '\&CRT;'   - Caret (circumflex)
--|    '`' = '\&GRV;'   - Grave accent
--|    '{' = '\&CBO;'   - Curly bracket open
--|    '|' = '\&VBR;'   - Vertical bar
--|    '}' = '\&CBC;'   - Curly bracket close
--|    '~' = '\&TLD;'   - Tilde
--| Define CCSIDs for certain areas in CLP strings on EBCDIC systems (0-reset)
--|    '&' [:digit:]+ ';  (..."&1047;get.file='&0;%s&1047;'",f)
--| Escape sequences for hexadecimal byte values
--|    '&' ['X''x'] :xdigit: :xdigit: ';' ("&xF5;")
```

1.5.2 FLAMCLP-Grammar for command line

Call siClpGrammar() to get the current supported grammar for the command lines. The list below could be a older state of the implementation.
Grammar for argument list or parameter file:

```
--| command        -> ['('] parameter_list [')']       (main=object)
--|                 |  ['.'] parameter                  (main=overlay)
--| parameter_list -> parameter SEP parameter_list
--|                 |  EMPTY
--| parameter      -> switch | assignment | object | overlay | array
--| switch         -> KEYWORD
--| assignment     -> KEYWORD '=' value
--|                 |  KEYWORD '=' KEYWORD # SELECTION #
--|                 |  KEYWORD '=>' STRING # parameter file #
--| object         -> KEYWORD ['('] parameter_list [')']
--|                 |  KEYWORD '=' STRING # parameter file #
--|                 |  KEYWORD '=>' STRING # parameter file #
```

```
--| overlay        -> KEYWORD ['.'] parameter
--|                |  KEYWORD '=' STRING # parameter file #
--|                |  KEYWORD '=>' STRING # parameter file #
--| array          -> KEYWORD '[' value_list   ']'
--|                |  KEYWORD '[' object_list  ']'
--|                |  KEYWORD '[' overlay_list ']'
--|                |  KEYWORD '=' value_list # with certain limitations #
--| It is recommended to use only enclosed array lists to know the end
--|                |  KEYWORD '[=' STRING ']' # parameter file #
--|                |  KEYWORD '[=>' STRING ']' # parameter file #
--| value_list     -> value SEP value_list
--|                |  EMPTY
--| object_list    -> object SEP object_list
--|                |  EMPTY
--| overlay_list   -> overlay SEP overlay_list
--|                |  EMPTY
--| A list of objects requires parenthesis to enclose the arguments
--|
--| value          -> term '+' value
--|                |  term '-' value
--|                |  term
--| term           -> factor '*' term
--|                |  factor '/' term
--|                |  factor
--| factor         -> NUMBER | FLOAT | STRING
--|                |  selection | variable | constant
--|                |  '(' value ')'
--| selection      -> KEYWORD # value from a selection table        #
--| variable       -> KEYWORD # value from a previous assignment    #
--|                |  KEYWORD '{' NUMBER '}' # with index for arrays #
--| constant       -> KEYWORD # see predefined constants at lexem   #
--| For strings only the operator '+' is implemented as concatenation
--| Strings without an operator in between are also concatenated
--| A number followed by a constant is a multiplication (4KiB=4*1024)
--|
--| Property File Parser
--| properties     -> property_list
--| property_list  -> property SEP property_list
--|                |  EMPTY
--| property       -> keyword_list '=' SUPPLEMENT
--| keyword_list   -> KEYWORD '.' keyword_list
--|                |  KEYWORD
--| SUPPLEMENT is a string in double quotation marks ("property")
```

A list of objects requires parenthesis to enclose the arguments. Only for one object of a certain level you can omit the round brackets. If you want define more than one object of a certain level you must use parenthesis to separate the objects from each other. In parameter files the command string for an overlay can be start with a dot '.' or not. The same is valid for the parenthesis '(...)' of an object.

1.5.3 FLAMCLP-Grammar for property file

Call siClpGrammar() to get the current supported grammar for property files. The list below could be a older state of the implementation.
Grammar for property file:

```
--| Property File Parser
--| properties     -> property_list
--| property_list  -> property SEP property_list
--|                |  EMPTY
--| property       -> keyword_list '=' SUPPLEMENT
--| keyword_list   -> KEYWORD '.' keyword_list
--|                |  KEYWORD
--| SUPPLEMENT is a string in double quotation marks ("property")
```

1.6 Utility functions for FLAMCLE/CLP

This interface provides additional several utility functions for the Command Line Executor (CLE) and Processor (CLP). These help functions makes the CLEP project platform independent and can also be used for other things. The CLEPUTL object should be static linked to CLP and/or CLE.

Chapter 2

Module Index

2.1 API definitions

Here is a list of the API parts:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 CLE Docu Types

Documentation types used in columns of table below.

Macros

- #define CLE_DOCTYP_COVER 1U

  Cover page (level must be 1).
- #define CLE_DOCTYP_CHAPTER 2U

  A chapter (level must > 1 and < 6).
- #define CLE_DOCTYP_PROGRAM 10U

  The main program chapter (like chapter but level must < 5).
- #define CLE_DOCTYP_PGMSYNOPSIS 11U

  The program synopsis.
- #define CLE_DOCTYP_PGMSYNTAX 12U

  The program syntax.
- #define CLE_DOCTYP_PGMHELP 13U

  The program help.
- #define CLE_DOCTYP_COMMANDS 20U

  The commands part.
- #define CLE_DOCTYP_OTHERCLP 21U

  Other CLP strings.
- #define CLE_DOCTYP_BUILTIN 22U

  The built-in function section.
- #define CLE_DOCTYP_LEXEMES 30U

  The appendix which prints the lexemes.
- #define CLE_DOCTYP_GRAMMAR 31U

  The appendix which prints the grammar.
- #define CLE_DOCTYP_VERSION 32U

  The appendix which prints the version (pcVsn must be given).
- #define CLE_DOCTYP_ABOUT 33U

  The appendix which prints the about (pcAbo must be given).
- #define CLE_DOCTYP_PROPREMAIN 41U

  The appendix which prints the remaining parameter documentation.
- #define CLE_DOCTYP_PROPDEFAULTS 42U

  The appendix which prints the default parameter documentation.
- #define CLE_DOCTYP_SPECIALCODES 51U

  The appendix which prints the special condition codes.
- #define CLE_DOCTYP_REASONCODES 52U

  The appendix which prints the reason codes (pfMsg must be provided).

5.1.1 Detailed Description

Documentation types used in columns of table below.

5.1.2 Macro Definition Documentation

5.1.2.1 CLE_DOCTYP_COVER

```
#define CLE_DOCTYP_COVER 1U
```

Cover page (level must be 1).

5.1.2.2 CLE_DOCTYP_CHAPTER

```
#define CLE_DOCTYP_CHAPTER 2U
```

A chapter (level must $>$ 1 and $<$ 6).

5.1.2.3 CLE_DOCTYP_PROGRAM

```
#define CLE_DOCTYP_PROGRAM 10U
```

The main program chapter (like chapter but level must $<$ 5).

5.1.2.4 CLE_DOCTYP_PGMSYNOPSIS

```
#define CLE_DOCTYP_PGMSYNOPSIS 11U
```

The program synopsis.

5.1.2.5 CLE_DOCTYP_PGMSYNTAX

```
#define CLE_DOCTYP_PGMSYNTAX 12U
```

The program syntax.

5.1.2.6 CLE_DOCTYP_PGMHELP

```
#define CLE_DOCTYP_PGMHELP 13U
```

The program help.

5.1.2.7 CLE_DOCTYP_COMMANDS

```
#define CLE_DOCTYP_COMMANDS 20U
```

The commands part.

5.1.2.8 CLE_DOCTYP_OTHERCLP

```
#define CLE_DOCTYP_OTHERCLP 21U
```

Other CLP strings.

5.1.2.9 CLE_DOCTYP_BUILTIN

```
#define CLE_DOCTYP_BUILTIN 22U
```

The built-in function section.

5.1.2.10 CLE_DOCTYP_LEXEMES

```
#define CLE_DOCTYP_LEXEMES 30U
```
The appendix which prints the lexemes.

5.1.2.11 CLE_DOCTYP_GRAMMAR

```
#define CLE_DOCTYP_GRAMMAR 31U
```
The appendix which prints the grammar.

5.1.2.12 CLE_DOCTYP_VERSION

```
#define CLE_DOCTYP_VERSION 32U
```
The appendix which prints the version (pcVsn must be given).

5.1.2.13 CLE_DOCTYP_ABOUT

```
#define CLE_DOCTYP_ABOUT 33U
```
The appendix which prints the about (pcAbo must be given).

5.1.2.14 CLE_DOCTYP_PROPREMAIN

```
#define CLE_DOCTYP_PROPREMAIN 41U
```
The appendix which prints the remaining parameter documentation.

5.1.2.15 CLE_DOCTYP_PROPDEFAULTS

```
#define CLE_DOCTYP_PROPDEFAULTS 42U
```
The appendix which prints the default parameter documentation.

5.1.2.16 CLE_DOCTYP_SPECIALCODES

```
#define CLE_DOCTYP_SPECIALCODES 51U
```
The appendix which prints the special condition codes.

5.1.2.17 CLE_DOCTYP_REASONCODES

```
#define CLE_DOCTYP_REASONCODES 52U
```
The appendix which prints the reason codes (pfMsg must be provided).

5.2 CLE Docu Keywords

ASCIIDOC key words used in fourth column of table below.

Macros

- #define CLE_DOCKYW_PREFACE "preface"

  Mark level 2 chapter as preface.
- #define CLE_DOCKYW_APPENDIX "appendix"

  Mark level 2 chapter as appendix.
- #define CLE_DOCKYW_GLOSSARY "glossary"

  Mark level 2 chapter as glossary.
- #define CLE_DOCKYW_COLOPHON "colophon"

  Mark level 2 chapter as colophon.

5.2.1 Detailed Description

ASCIIDOC key words used in fourth column of table below.

5.2.2 Macro Definition Documentation

5.2.2.1 CLE_DOCKYW_PREFACE

```
#define CLE_DOCKYW_PREFACE "preface"
```

Mark level 2 chapter as preface.

5.2.2.2 CLE_DOCKYW_APPENDIX

```
#define CLE_DOCKYW_APPENDIX "appendix"
```

Mark level 2 chapter as appendix.

5.2.2.3 CLE_DOCKYW_GLOSSARY

```
#define CLE_DOCKYW_GLOSSARY "glossary"
```

Mark level 2 chapter as glossary.

5.2.2.4 CLE_DOCKYW_COLOPHON

```
#define CLE_DOCKYW_COLOPHON "colophon"
```

Mark level 2 chapter as colophon.

5.3 CLE Docu Anchors

The anchors for chapters and appendixes (5. column).

Macros

- #define CLE_ANCHOR_BUILTIN_FUNCTIONS "CLEP.BUILTIN.FUNCTIONS"

  Chapter built-in functions.
- #define CLE_ANCHOR_APPENDIX_ABOUT "CLEP.APPENDIX.ABOUT"

  Appendix About.
- #define CLE_ANCHOR_APPENDIX_VERSION "CLEP.APPENDIX.VERSION"

  Appendix Version.
- #define CLE_ANCHOR_APPENDIX_LEXEMES "CLEP.APPENDIX.LEXEMES"

  Appendix Lexemes.
- #define CLE_ANCHOR_APPENDIX_GRAMMAR "CLEP.APPENDIX.GRAMMAR"

  Appendix Grammar.
- #define CLE_ANCHOR_APPENDIX_RETURNCODES "CLEP.APPENDIX.RETURNCODES"

  Appendix Return codes.
- #define CLE_ANCHOR_APPENDIX_REASONCODES "CLEP.APPENDIX.REASONCODES"

  Appendix Reason codes.
- #define CLE_ANCHOR_APPENDIX_PROPERTIES "CLEP.APPENDIX.PROPERTIES"

  Appendix Properties.

5.3.1 Detailed Description

The anchors for chapters and appendixes (5. column).
This anchors are required to fulfill the CLEP internal links and must be assigned to the corresponding chapters in the table below.

5.3.2 Macro Definition Documentation

5.3.2.1 CLE_ANCHOR_BUILTIN_FUNCTIONS

```
#define CLE_ANCHOR_BUILTIN_FUNCTIONS "CLEP.BUILTIN.FUNCTIONS"
```

Chapter built-in functions.

5.3.2.2 CLE_ANCHOR_APPENDIX_ABOUT

```
#define CLE_ANCHOR_APPENDIX_ABOUT "CLEP.APPENDIX.ABOUT"
```

Appendix About.

5.3.2.3 CLE_ANCHOR_APPENDIX_VERSION

```
#define CLE_ANCHOR_APPENDIX_VERSION "CLEP.APPENDIX.VERSION"
```

Appendix Version.

5.3.2.4 CLE_ANCHOR_APPENDIX_LEXEMES

```
#define CLE_ANCHOR_APPENDIX_LEXEMES "CLEP.APPENDIX.LEXEMES"
```

Appendix Lexemes.

5.3.2.5 CLE_ANCHOR_APPENDIX_GRAMMAR

```
#define CLE_ANCHOR_APPENDIX_GRAMMAR "CLEP.APPENDIX.GRAMMAR"
```

Appendix Grammar.

5.3.2.6 CLE_ANCHOR_APPENDIX_RETURNCODES

```
#define CLE_ANCHOR_APPENDIX_RETURNCODES "CLEP.APPENDIX.RETURNCODES"
```

Appendix Return codes.

5.3.2.7 CLE_ANCHOR_APPENDIX_REASONCODES

```
#define CLE_ANCHOR_APPENDIX_REASONCODES "CLEP.APPENDIX.REASONCODES"
```

Appendix Reason codes.

5.3.2.8 CLE_ANCHOR_APPENDIX_PROPERTIES

```
#define CLE_ANCHOR_APPENDIX_PROPERTIES "CLEP.APPENDIX.PROPERTIES"
```

Appendix Properties.

5.4 CLE Docu Table

The structure and corresponding macros are used to define a table in order to generate documentation.

Data Structures

- struct CleDoc

  CLE Structure for documentation table.

Macros

- #define CLEDOC_OPN(name) TsCleDoc name[ ]

  Starts the documentation generation table overview.
- #define CLETAB_DOC(typ, lev, num, kyw, anc, hdl, man, idt) {(typ),(lev),(num),(kyw),(anc),(hdl),(man),(idt)},

  Starts the documentation generation table.
- #define CLEDOC_CLS { 0 , 0 , NULL, NULL, NULL, NULL, NULL, NULL}

  Ends a table with constant definitions.

Typedefs

- typedef struct CleDoc TsCleDoc

  CLE Structure for documentation table.

5.4.1 Detailed Description

The structure and corresponding macros are used to define a table in order to generate documentation.

5.4.2 Macro Definition Documentation

5.4.2.1 CLEDOC_OPN

```
#define CLEDOC_OPN(
                name ) TsCleDoc name[ ]
```

Starts the documentation generation table overview.

Parameters

name	Name of this table.

5.4.2.2 CLETAB_DOC

```
#define CLETAB_DOC(
                typ,
                lev,
                num,
                kyw,
                anc,
                hdl,
                man,
                idt ) {(typ),(lev),(num),(kyw),(anc),(hdl),(man),(idt)},
```

Starts the documentation generation table.

Parameters

in	*typ*	Documentation type.
in	*lev*	Level of the chapter.
in	*num*	Prefix for number string.
in	*kyw*	Optional ASCIIDOC key word (printed in front of headline in single square brackets).
in	*anc*	Optional anchor for this chapter (printed in front of headline in double square brackets).
in	*hdl*	Headline for this chapter.
in	*man*	Optional manual page for this chapter.
in	*idt*	Optional new line separated list of index terms for this chapter.

5.4.2.3 CLEDOC_CLS

```
#define CLEDOC_CLS { 0 , 0 , NULL, NULL, NULL, NULL, NULL, NULL}
```

Ends a table with constant definitions.

5.4.3 Typedef Documentation

5.4.3.1 TsCleDoc

```
typedef struct CleDoc TsCleDoc
```

CLE Structure for documentation table.

This structure is used to build a table with the macros CLEDOC_OPN, CLETAB_DOC and CLEDOC_CLS. This table must be provided to the function siCleExecute for documentation generation using the built-in functions GENDOCU and HTMLDOC.

5.5 CLE Function Pointer (call backs)

Type definitions for the callback functions used by CLE.

Typedefs

- typedef void *() TfCleOpenPrint(FILE *pfOut, FILE *pfErr, const char *pcPat, const char *pcOwn, const char *pcPgm, const char *pcBld, int *piHdr, int *piAnc, int *piIdt, int *piPat, int *psPs1, int *piPs2, int *piPr3)

  Function 'opnHtmlDoc' of library 'libhtmldoc' called if built-in function HTMLDOC used.
- typedef int() TfCleClosePrint(void *pvHdl)

  Function 'clsHtmlDoc' of library 'libhtmldoc' called if built-in function HTMLDOC used.
- typedef int() TfIni(void *pvHdl, FILE *pfOut, FILE *pfTrc, void *pvGbl, const char *pcOwn, const char *pcPgm, void *pvClp)

  Type definition for initialization FLAMCLE command structure.
- typedef int() TfMap(void *pvHdl, FILE *pfOut, FILE *pfTrc, void *pvGbl, int *piOid, void *pvClp, void *pvPar)

  Type definition for mapping parsed values from the FLAMCLP structure.
- typedef int() TfRun(void *pvHdl, FILE *pfOut, FILE *pfTrc, void *pvGbl, const char *pcOwn, const char *pcPgm, const char *pcVsn, const char *pcAbo, const char *pcLic, const char *pcFkt, const char *pcCmd, const char *pcLst, const void *pvPar, int *piWrn, int *piScc)

  Type definition for CLE run function.
- typedef int() TfFin(FILE *pfOut, FILE *pfTrc, void *pvGbl, void *pvPar)

  Type definition for the CLE fin function.
- typedef const char *() TfMsg(const int siRsn)

  Type definition for the CLE message function.

5.5.1 Detailed Description

Type definitions for the callback functions used by CLE.

5.5.2 Typedef Documentation

5.5.2.1 TfCleOpenPrint

```
typedef void*() TfCleOpenPrint(FILE *pfOut, FILE *pfErr, const char *pcPat, const char *pcOwn,
const char *pcPgm, const char *pcBld, int *piHdr, int *piAnc, int *piIdt, int *piPat, int *ps↩
Ps1, int *piPs2, int *piPr3)
```

Function 'opnHtmlDoc' of library 'libhtmldoc' called if built-in function HTMLDOC used.
The built-in function HTMLDOC use a service provider interface to create the documentation using a callback function for each page/chapter. This function is called in front of the generation process to establish a handle for the callback function.
The resulting handle is given to the callback function TfClpPrintPage defined by CLP. This callback function is called for each page/chapter written. After the documentation generation process the handle will be released with the function TfCleClosePrint.

Parameters

in	*pfOut*	File pointer for normal output (NULL for no output).
in	*pfErr*	File pointer for error messages (NULL for no output).
in	*pcPat*	Path where the documentation is written to.
in	*pcOwn*	String of the current Owner.
in	*pcPgm*	String with the current program name.
in	*pcBld*	Optional build/version number (could be NULL).
in,out	*piHdr*	Boolean to define if header information for cover page generated by CLEP.

Parameters

in,out	*piAnc*	Boolean to define if anchors generated by CLEP.
in,out	*piIdt*	Boolean to define if index terms generated by CLEP.
in,out	*piPat*	Boolean to define if path string part of the synopsis.
in,out	*piPs1*	Character to separate parts before command strings of the file path (use '/' for sub folders or '-' to get file names).
in,out	*piPs2*	Character to separate parts inside command strings of the file path (use '/' for sub folders or '-' to get file names).
in,out	*piPr3*	Character to replace non alpha-numerical characters in the file name.

Returns

Pointer to an handle or NULL if open failed

5.5.2.2 TfCleClosePrint

```
typedef int() TfCleClosePrint(void *pvHdl)
```

Function 'clsHtmlDoc' of library 'libhtmldoc' called if built-in function HTMLDOC used.
The built-in function HTMLDOC use a service provider interface to create the documentation using a callback function for each page/chapter. A callback function is a function that is passed as a parameter to another function and is called later by this function under defined conditions with defined arguments.
This built-in function HTMLDOC is called after the generation process to free resources associated with this handle. The handle will be generated with the function TfCleOpenPrint in front of documentation generation.

Parameters

in	*pvHdl*	Pointer the the print handle.

Returns

Return code (0 is OK else error).

5.5.2.3 TfIni

```
typedef int() TfIni(void *pvHdl, FILE *pfOut, FILE *pfTrc, void *pvGbl, const char *pcOwn,
const char *pcPgm, void *pvClp)
```

Type definition for initialization FLAMCLE command structure.
This FLAMCLE structure is used by the command line processor (FLAMCLP) to save the parsed values. The I↩ NI function uses this structure to pre-define values. These values are still valid, if no values are defined over the properties or command line.
The current owner and the program name are provided by the function CleExecute (see siCleExecute()), to know these values during initialization.
If additional dynamic memory is required in the CLP structure the provided handle can be used for calling function pvClpAlloc.

Parameters

in	*pvHdl*	Pointer to the CLP handle for allocation of memory in the CLP structure.
in	*pfOut*	File pointer for outputs (given over CleExecute, see siCleExecute()).
in	*pfTrc*	File pointer for tracing (given over CleExecute, see siCleExecute()).
in	*pvGbl*	Pointer to a global handle as black box pass through (given over CleExecute, see siCleExecute()).

Parameters

in	*pcOwn*	Current owner name (given over CleExecute, see siCleExecute()).
in	*pcPgm*	Current program name (given over CleExecute, see siCleExecute()).
out	*pvClp*	Pointer to the corresonding FLAMCLP structure for initialisation.

Returns

Reason code (!=0) for termination or 0 for success.

5.5.2.4 TfMap

```
typedef int() TfMap(void *pvHdl, FILE *pfOut, FILE *pfTrc, void *pvGbl, int *piOid, void *pv↩
Clp, void *pvPar)
```

Type definition for mapping parsed values from the FLAMCLP structure.
This function is used to map the parsed values from the FLAMCLP structure to the parameter structure of the corresponding command. This mapping can simply a memcpy or pointer assignment but it is often useful to make a real mapping for example:
The executed subprogram needs a table with some definitions. These definitions can easily managed in a file. Over the command line the user define the file name and the mapping function open this file read the definition, allocates the memory and stores the definition in the parameter structure. This means that the user defines the file name, but the executed subprogram gets the content of this file. Such things are realized over a real mapping between the parsed values and the needed arguments.
For overlay based commands the pointer to the object identifier is provided (taken from siCleExecute()). This integer can then be used to choose the correct data structure from the corresponding CLP union for mapping. (The piOid can also (mis)used to give back a integer to the caller of siCleExecute() from mapping if this pointer not NULL.)
If additional dynamic memory required in the CLP structure the provided handle can be used for pvClpAlloc.

Parameters

in	*pvClp*	Pointer to the CLP handle for allocation of memory in the CLP structure.
in	*pfOut*	File pointer for outputs (mainly error messages, given over CleExecute, see siCleExecute()).
in	*pfTrc*	File pointer for tracing (mainly for complex stuff, given over CleExecute, see siCleExecute()).
in	*pvGbl*	Pointer to a global handle as black box pass through (given over CleExecute, see siCleExecute()).
in	*piOid*	Pointer to the object identifier for overlay commands, if the pointer set at siCleExecute().
in	*pvClp*	Pointer to the filled FLAMCLP structure (output from the command line parser).
out	*pvPar*	Pointer to the parameter structure, which will be filled based on the FLAMCLP structure with this function.

Returns

Reason code (!=0) for termination or 0 for success. If a no run reason code (!=0) defined then the run function is not executed.

5.5.2.5 TfRun

```
typedef int() TfRun(void *pvHdl, FILE *pfOut, FILE *pfTrc, void *pvGbl, const char *pcOwn,
const char *pcPgm, const char *pcVsn, const char *pcAbo, const char *pcLic, const char *pcFkt,
const char *pcCmd, const char *pcLst, const void *pvPar, int *piWrn, int *piScc)
```

Type definition for CLE run function.
The run function is used to execute the corresponding subprogram for a command using the mapped parameter structure. For logging or other purpose all collected information are also provided at input.

The run function returns like all other functions executed by CLE a reason code. Additional to the other functions there will be another indicator to define the condition code (return code of the program). For the run function 2 possible condition codes are defined. First an error with value 8 and additional 4 in a case of a warning. The warning means that the run don't fail, but something was happen.
If additional dynamic memory required in the FLC structure (from mapping) the provided handle can be used for pvClpAlloc().

Parameters

in	*pvHdl*	Pointer to the CLP handle for allocation of memory in the FLC structure.
in	*pfOut*	File pointer for outputs (given over structure CleExecute, see siCleExecute()).
in	*pfTrc*	File pointer for tracing (given over structure CleExecute, see siCleExecute()).
in	*pvGbl*	Pointer to a global handle as black box pass through (given over structure CleExecute, see siCleExecute()).
in	*pcOwn*	Current owner name (given over structure CleExecute, see siCleExecute()).
in	*pcPgm*	Current program name (given over structure CleExecute, see siCleExecute())
in	*pcVsn*	Current version information (given from structure CleExecute, see siCleExecute()).
in	*pcAbo*	Current about information (given from structure CleExecute, see siCleExecute()).
in	*pcLic*	Current license text (given from structure CleExecute, see siCleExecute()).
in	*pcFkt*	Current function name (key word of the command).
in	*pcCmd*	Current command (complete entered line of user).
in	*pcLst*	Current list of parsed arguments (given from FLAMCLP, could be NULL or empty).
in	*pvPar*	Pointer to the filled parameter for the run of the subprogram.
out	*piWrn*	Pointer to an integer (the fist half word is true (0x0001), if warnings collated by directory walk, the second halfword is true (0x0001) if warnings are logged).
out	*piScc*	Pointer to an integer containing a special condition code (if greater CLERTC_MAX(64) then used instead of CLERTC_RUN(8)).

Returns

Reason code (!=0) for termination or warning, 0 for success

5.5.2.6 TfFin

```
typedef int() TfFin(FILE *pfOut, FILE *pfTrc, void *pvGbl, void *pvPar)
```

Type definition for the CLE fin function.
This function is called at the end after the run to clean up the parameter structure. For example, it could be there was a file pointer which must be closed or memory which must be freed.

Parameters

in	*pfOut*	File pointer for outputs (given over CleExecute, see siCleExecute()).
in	*pfTrc*	File pointer for tracing (given over CleExecute, see siCleExecute()).
in	*pvGbl*	Pointer to a global handle as black box pass through (given over CleExecute, see siCleExecute()).
in	*pvPar*	Pointer to the filled parameter structure for cleanup.

Returns

Reason code (!=0) for termination or 0 for success.

5.5.2.7 TfMsg

```
typedef const char*() TfMsg(const int siRsn)
```

Type definition for the CLE message function.
This function is called to generate a appendix for the reason codes and to provide better messages in a case of an error.
In a loop starting with 1 the messages are printed to the appendix. If a NULL pointer (no message) returned the loop is finished. A empty message ("") indicates an reason code which is not printed to the appendix. Is a NULL pointer provided for this function the appendix for the reason codes and additional error messages are not generated.

Parameters

in	*siRsn*	Reason code from INI, MAP, RUN and FIN function.

Returns

Pointer to the corresponding message.

5.6 CLE Command Table

Command structure and corresponding macros used to define CLE command tables.

Data Structures

- struct CleCommand

  CLE structure for command table.

Macros

- #define CLECMD_OPN(name) TsCleCommand name[ ]

  Starts a table with command definitions.
- #define CLETAB_CMD(kyw, tab, clp, par, oid, ini, map, run, fin, flg, man, hlp) {(kyw),(tab),(clp),(par),(oid),(ini),(map),(run),(fin),(fl

  Defines a command with the command line keyword kyw.
- #define CLECMD_CLS { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, 0 , NULL, NULL}

  Ends a table with constant definitions.

Typedefs

- typedef struct CleCommand TsCleCommand

  CLE structure for command table.

5.6.1 Detailed Description

Command structure and corresponding macros used to define CLE command tables.

5.6.2 Macro Definition Documentation

5.6.2.1 CLECMD_OPN

```
#define CLECMD_OPN(
                name ) TsCleCommand name[]
```

Starts a table with command definitions.

Parameters

in	*name*	Name of this table.

5.6.2.2 CLETAB_CMD

```
#define CLETAB_CMD(
                kyw,
                tab,
                clp,
                par,
                oid,
                ini,
                map,
                run,
                fin,
                flg,
```

```
man,
hlp ) {(kyw),(tab),(clp),(par),(oid),(ini),(map),(run),(fin),(flg),(man),(hlp)},
```

Defines a command with the command line keyword *kyw*.

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*tab*	Pointer to the main table for this command.
in	*clp*	Pointer to the corresponding FLAMCLP structure (generated with the CLPMAC.h marcros).
in	*par*	Pointer to the corresponding parameter structure.
in	*oid*	Pointer to an integer to define the main table as overlay or NULL to define the main table as object If the pointer is set the object identifier of the chosen argument of the overlay is given back.
in	*ini*	Pointer to the initialization function for the FLAMCLP structure (see TfIni).
in	*map*	Pointer to the mapping function (see TfMap). The mapping functions maps the parsed content of the FLAMCLP structure in the PAR structure.
in	*run*	Pointer to the run function to execute the subprogram with the PAR structure (see TfRun).
in	*fin*	Pointer to the finalization function to clean up the parameter structure (see TfFin).
in	*flg*	Flag to indicate a hidden (==0) or visible (!=0) command, For correct numbering, put hidden commands to the end of the table.
in	*man*	Pointer to a null-terminated string for a detailed description of this command. (in ASCIDOC format, content behind .DESCRIPTION, usually some paragraphs plus .OPTIONS and/or .EXAMPLES) It is recommended to use a header file with a define for this long string).
in	*hlp*	String for a short context sensitive help to this command.

5.6.2.3 CLECMD_CLS

```
#define CLECMD_CLS { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, 0 , NULL, NULL}
```

Ends a table with constant definitions.

5.6.3 Typedef Documentation

5.6.3.1 TsCleCommand

```
typedef struct CleCommand TsCleCommand
```

CLE structure for command table.

The command table defines all the commands which could be executed by CLE. To simplify the definition of command tables it is recommended to use the CLPTAB macros.

5.7 CLE Other CLP string table

Structure and corresponding macros containing CLP string table in order to append to generated documentation.

Data Structures

- struct CleOtherClp

  CLE table structure for other CLP strings.

Macros

- #define CLEOTH_OPN(name) TsCleOtherClp name[ ]

  Starts a table with other CLP strings.
- #define CLETAB_OTH(rot, kyw, tab, man, hlp, ovl) {(rot),(kyw),(tab),(man),(hlp),(ovl)},

  Defines a appendix for the object or overlay cmd of the root rot with the headline of hdl for a certain other CLP string. .
- #define CLEOTH_CLS { NULL, NULL, NULL, NULL, NULL, 0}

  Ends a table with other CLP strings.

Typedefs

- typedef struct CleOtherClp TsCleOtherClp

  CLE table structure for other CLP strings.

5.7.1 Detailed Description

Structure and corresponding macros containing CLP string table in order to append to generated documentation.

5.7.2 Macro Definition Documentation

5.7.2.1 CLEOTH_OPN

```
#define CLEOTH_OPN(
              name ) TsCleOtherClp name[ ]
```

Starts a table with other CLP strings.

Parameters

in	*name*	Name of this table.

5.7.2.2 CLETAB_OTH

```
#define CLETAB_OTH(
              rot,
              kyw,
              tab,
              man,
              hlp,
              ovl ) {(rot),(kyw),(tab),(man),(hlp),(ovl)},
```

Defines a appendix for the object or overlay *cmd* of the root *rot* with the headline of *hdl* for a certain other CLP string. .

Parameters

in	*rot*	Pointer to the program/root key word for this string (:alpha:[:alnum:|'_']∗).
in	*kyw*	Pointer to the key word for this string (:alpha:[:alnum:|'_']∗).
in	*tab*	Pointer to the main argument table for this CLP string.
in	*man*	Pointer to a null-terminated string for a detailed description for this CLP string (in ASCIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs plus .OPTIONS and/or.EXAMPLES). It is recommended to use a header file with a define for this long string. "&{OWN}" and "&{PGM}" are replaced with the current owner and program name. The resulting text is converted on EBCDIC systems.
in	*hlp*	String for a short context sensitive help for this CLP string (converted on EBCDIC systems). n
in	*ovl*	True if provided table must be interpreted as overlay else as object.

5.7.2.3 CLEOTH_CLS

```
#define CLEOTH_CLS { NULL, NULL, NULL, NULL, NULL, 0}
```

Ends a table with other CLP strings.

5.7.3 Typedef Documentation

5.7.3.1 TsCleOtherClp

```
typedef struct CleOtherClp TsCleOtherClp
```

CLE table structure for other CLP strings.
This structure is used to define a table of CLP strings in order to add as appendix to the generated documentation.

5.8 CLP Error Codes

Error codes for command line parsing.

Data Structures

- struct ClpError

  Defines a structure with error information.

Macros

- #define CLP_OK 0

  Return code for a successful parsing: 0, otherwize > 0.
- #define CLPERR_LEX -1

  Lexical error (determined by scanner).
- #define CLPERR_SYN -2

  Syntax error (determined by parser).
- #define CLPERR_SEM -3

  Semantic error (determined by builder).
- #define CLPERR_TYP -4

  Type error (internal error with argument types).
- #define CLPERR_TAB -5

  Table error (internal error with argument tables).
- #define CLPERR_SIZ -6

  Size error (internal error with argument tables and data structures).
- #define CLPERR_PAR -7

  Parameter error (internal error with argument tables and data structures).
- #define CLPERR_MEM -8

  Memory error (internal error with argument tables and data structures).
- #define CLPERR_INT -9

  Internal error (internal error with argument tables and data structures).
- #define CLPERR_SYS -10

  System error (internal error with argument tables and data structures).
- #define CLPERR_AUT -11

  Authorization request failed.
- #define CLPSRC_CMD ":command line:"

  From command line.
- #define CLPSRC_PRO ":property list:"

  From property list.
- #define CLPSRC_DEF ":default value:"

  From default value.
- #define CLPSRC_ENV ":environment variable:"

  From environment variable.
- #define CLPSRC_PRF ":property file:"

  From property file.
- #define CLPSRC_CMF ":command file:"

  From command file.
- #define CLPSRC_PAF ":parameter file:"

  Parameter file.
- #define CLPSRC_SRF ":string file:"

  String file.

Typedefs

- typedef struct ClpError TsClpError

  Defines a structure with error information.

5.8.1 Detailed Description

Error codes for command line parsing.

5.8.2 Macro Definition Documentation

5.8.2.1 CLP_OK

```
#define CLP_OK 0
```

Return code for a successful parsing: 0, otherwize $>$ 0.

5.8.2.2 CLPERR_LEX

```
#define CLPERR_LEX -1
```

Lexical error (determined by scanner).

5.8.2.3 CLPERR_SYN

```
#define CLPERR_SYN -2
```

Syntax error (determined by parser).

5.8.2.4 CLPERR_SEM

```
#define CLPERR_SEM -3
```

Semantic error (determined by builder).

5.8.2.5 CLPERR_TYP

```
#define CLPERR_TYP -4
```

Type error (internal error with argument types).

5.8.2.6 CLPERR_TAB

```
#define CLPERR_TAB -5
```

Table error (internal error with argument tables).

5.8.2.7 CLPERR_SIZ

```
#define CLPERR_SIZ -6
```

Size error (internal error with argument tables and data structures).

5.8.2.8 CLPERR_PAR

```
#define CLPERR_PAR -7
```

Parameter error (internal error with argument tables and data structures).

5.8.2.9 CLPERR_MEM

```
#define CLPERR_MEM -8
```
Memory error (internal error with argument tables and data structures).

5.8.2.10 CLPERR_INT

```
#define CLPERR_INT -9
```
Internal error (internal error with argument tables and data structures).

5.8.2.11 CLPERR_SYS

```
#define CLPERR_SYS -10
```
System error (internal error with argument tables and data structures).

5.8.2.12 CLPERR_AUT

```
#define CLPERR_AUT -11
```
Authorization request failed.

5.8.2.13 CLPSRC_CMD

```
#define CLPSRC_CMD ":command line:"
```
From command line.

5.8.2.14 CLPSRC_PRO

```
#define CLPSRC_PRO ":property list:"
```
From property list.

5.8.2.15 CLPSRC_DEF

```
#define CLPSRC_DEF ":default value:"
```
From default value.

5.8.2.16 CLPSRC_ENV

```
#define CLPSRC_ENV ":environment variable:"
```
From environment variable.

5.8.2.17 CLPSRC_PRF

```
#define CLPSRC_PRF ":property file:"
```
From property file.

5.8.2.18 CLPSRC_CMF

```
#define CLPSRC_CMF ":command file:"
```
From command file.

5.8.2.19 CLPSRC_PAF

```
#define CLPSRC_PAF ":parameter file:"
```
Parameter file.

5.8.2.20 CLPSRC_SRF

```
#define CLPSRC_SRF ":string file:"
```
String file.

5.8.3 Typedef Documentation

5.8.3.1 TsClpError

```
typedef struct ClpError TsClpError
```
Defines a structure with error information.
A pointer to this structure can be provided at siClpOpen() to have access to the error information managed in the CLP handle.
The pointers are set by CLP and valid until CLP is closed.

5.9 CLP Data Types

Data types of parameter in the argument table.

Macros

- #define CLPTYP_NON 0

  No type - Mark the end of an argument table.
- #define CLPTYP_SWITCH 1

  Switch (single keyword representing a number (OID)).
- #define CLPTYP_NUMBER 2

  Signed or unsigned integer number (8, 16, 32 or 64 bit).
- #define CLPTYP_FLOATN 3

  Floating point number (32 or 64 bit).
- #define CLPTYP_STRING 4

  String literal (binary (HEX, ASCII, EBCDIC, CHARS) or null-terminated (default)).
- #define CLPTYP_OBJECT 5

  Object (KEYWORD(parameter_list)) can contain arbitrary list of other types.
- #define CLPTYP_OVRLAY 6

  Overlay (KEYWORD.KEYWORD...) contains one of its list as in a C union.
- #define CLPTYP_XALIAS -1

  For alias definition (used in the corresponding table macro)

5.9.1 Detailed Description

Data types of parameter in the argument table.

5.9.2 Macro Definition Documentation

5.9.2.1 CLPTYP_NON

```
#define CLPTYP_NON 0
```

No type - Mark the end of an argument table.

5.9.2.2 CLPTYP_SWITCH

```
#define CLPTYP_SWITCH 1
```

Switch (single keyword representing a number (OID)).

5.9.2.3 CLPTYP_NUMBER

```
#define CLPTYP_NUMBER 2
```

Signed or unsigned integer number (8, 16, 32 or 64 bit).

5.9.2.4 CLPTYP_FLOATN

```
#define CLPTYP_FLOATN 3
```

Floating point number (32 or 64 bit).

5.9.2.5 CLPTYP_STRING

```
#define CLPTYP_STRING 4
```

String literal (binary (HEX, ASCII, EBCDIC, CHARS) or null-terminated (default)).

5.9.2.6 CLPTYP_OBJECT

```
#define CLPTYP_OBJECT 5
```

Object (KEYWORD(parameter_list)) can contain arbitrary list of other types.

5.9.2.7 CLPTYP_OVRLAY

```
#define CLPTYP_OVRLAY 6
```

Overlay (KEYWORD.KEYWORD...) contains one of its list as in a C union.

5.9.2.8 CLPTYP_XALIAS

```
#define CLPTYP_XALIAS -1
```

For alias definition (used in the corresponding table macro)

5.10 CLP Close Method

Method used to close the CLP (see vdClpClose()).

Macros

- #define CLPCLS_MTD_ALL 1

  Complete close, free anything including the dynamic allocated buffers in the CLP structure.
- #define CLPCLS_MTD_KEP 0

  Free anything except the allocated memory in CLP structure and keep the handle open to close it later with method ALL.
- #define CLPCLS_MTD_EXC 2

  Free anything including the handle except the allocated memory in the CLP structure, the application must free the dynamic allocated buffers in the CLP structure it self.

5.10.1 Detailed Description

Method used to close the CLP (see vdClpClose()).

5.10.2 Macro Definition Documentation

5.10.2.1 CLPCLS_MTD_ALL

```
#define CLPCLS_MTD_ALL 1
```

Complete close, free anything including the dynamic allocated buffers in the CLP structure.

5.10.2.2 CLPCLS_MTD_KEP

```
#define CLPCLS_MTD_KEP 0
```

Free anything except the allocated memory in CLP structure and keep the handle open to close it later with method ALL.

5.10.2.3 CLPCLS_MTD_EXC

```
#define CLPCLS_MTD_EXC 2
```

Free anything including the handle except the allocated memory in the CLP structure, the application must free the dynamic allocated buffers in the CLP structure it self.

5.11 CLP Property Method

Method for property printing (see siClpProperties()).

Macros

- #define CLPPRO_MTD_ALL 0

  All properties are printed (manual pages added as comment).
- #define CLPPRO_MTD_SET 1

  Only defined properties are printed (no manual pages used).
- #define CLPPRO_MTD_CMT 2

  All properties are printed, but not defined properties are line comments .
- #define CLPPRO_MTD_DOC 3

  All property only parameter are printed as documentation.

5.11.1 Detailed Description

Method for property printing (see siClpProperties()).

5.11.2 Macro Definition Documentation

5.11.2.1 CLPPRO_MTD_ALL

```
#define CLPPRO_MTD_ALL 0
```

All properties are printed (manual pages added as comment).

5.11.2.2 CLPPRO_MTD_SET

```
#define CLPPRO_MTD_SET 1
```

Only defined properties are printed (no manual pages used).

5.11.2.3 CLPPRO_MTD_CMT

```
#define CLPPRO_MTD_CMT 2
```

All properties are printed, but not defined properties are line comments .

5.11.2.4 CLPPRO_MTD_DOC

```
#define CLPPRO_MTD_DOC 3
```

All property only parameter are printed as documentation.

5.12 CLP Flags

Flags for command line parsing.

Macros

- #define CLPFLG_NON 0x00000000U

  To define no special flags.
- #define CLPFLG_ALI 0x00000001U

  This parameter is an alias for another argument (set by macros).
- #define CLPFLG_CON 0x00000002U

  This parameter is a constant definition (no argument, no link, no alias (set by macros)).
- #define CLPFLG_CMD 0x00000004U

  If set the parameter is only used within the command line (command line only).
- #define CLPFLG_PRO 0x00000008U

  If set the parameter is only used within the property file (property file only).
- #define CLPFLG_SEL 0x00000010U

  If set only the predefined constants over the corresponding key words can be selected (useful to define selections).
- #define CLPFLG_FIX 0x00000020U

  This argument has a fixed length (only useful for strings if a typedef defines a fixed length per element, else set internally).
- #define CLPFLG_BIN 0x00000040U

  This argument can contain binary data without null termination (length must be known or determined with a link).
- #define CLPFLG_DMY 0x00000080U

  If set the parameter is not put in the symbol table, meaning it is only a peace of memory in the CLP structure.
- #define CLPFLG_CNT 0x00000100U

  This link will be filled by the calculated amount of elements (useful for arrays).
- #define CLPFLG_OID 0x00000200U

  This link will be filled by the object identifier (OID) of the chosen argument (useful for overlays).
- #define CLPFLG_IND 0x00000400U

  This link will be filled with the index (position) in the CLP string (byte offset of the current key word).
- #define CLPFLG_HID 0x00000800U

  If set the parameter is not visible, meaning it is a hidden parameter.
- #define CLPFLG_ELN 0x00001000U

  This link will be filled by the calculated length of an element (fixed types == data size, packed types == data length).
- #define CLPFLG_SLN 0x00002000U

  This link will be filled by the calculated string length for an element (only for null-terminated strings).
- #define CLPFLG_TLN 0x00004000U

  This link will be filled by the calculated total length for the argument (sum of all element lengths).
- #define CLPFLG_DEF 0x00010000U

  This flag enables to use the OID as default for numbers if no value is assigned (only the keyword is used (syntax extension)).
- #define CLPFLG_CHR 0x00020000U

  This flag will set the default method of interpretation of a binary string to local character string (DEFAULT).
- #define CLPFLG_ASC 0x00040000U

  This flag will set the default method of interpretation of a binary string to ASCII.
- #define CLPFLG_EBC 0x00080000U

  This flag will set the default method of interpretation of a binary string to EBCDIC.
- #define CLPFLG_HEX 0x00100000U

  This flag will set the default method of interpretation of a binary string to hexadecimal.
- #define CLPFLG_PDF 0x00200000U

This flag will be set if a property value was defined from outside, it will be FALSE if the property value was hard coded in the tables.

- #define CLPFLG_TIM 0x00400000U

  This flag mark a number as time value (only used to print out the corressponing time stamp).
- #define CLPFLG_DYN 0x00800000U

  This flag mark a string or array as dynamic (only a pointer to allocated memory is used and must be freeed by the user).
- #define CLPFLG_PWD 0x01000000U

  This flag will ensure that the clear value is only put into the data structure but not traced, logged or given away elsewhere.
- #define CLPFLG_DLM 0x02000000U

  This flag ensures that fix size arrays has a empty (initialized) last element (max-1) as delimiter. Additional you enforce 0xFF at the and of a non fix size string array (size-1).
- #define CLPFLG_UNS 0x04000000U

  Marks a number as unsigned (prevent negative values).
- #define CLPFLG_XML 0x08000000U

  Marks zero terminated string as XML path where '(' and ')' are used to replace environment variables.
- #define CLPFLG_FIL 0x10000000U

  Marks zero terminated string as file and replace additional '~' by HOME and corrects the prefix for different platforms.
- #define CLPFLG_LAB 0x20000000U

  Marks zero terminated string as label and replace additional '~' by USER, '^' by OWNER and '!' by ENVID .
- #define CLPFLG_UPP 0x40000000U

  Converts zero terminated strings to upper case.
- #define CLPFLG_LOW 0x80000000U

  Converts zero terminated strings to lower case.

5.12.1 Detailed Description

Flags for command line parsing.

5.12.2 Macro Definition Documentation

5.12.2.1 CLPFLG_NON

```
#define CLPFLG_NON 0x00000000U
```

To define no special flags.

5.12.2.2 CLPFLG_ALI

```
#define CLPFLG_ALI 0x00000001U
```

This parameter is an alias for another argument (set by macros).

5.12.2.3 CLPFLG_CON

```
#define CLPFLG_CON 0x00000002U
```

This parameter is a constant definition (no argument, no link, no alias (set by macros)).

5.12.2.4 CLPFLG_CMD

```
#define CLPFLG_CMD 0x00000004U
```

If set the parameter is only used within the command line (command line only).

5.12.2.5 CLPFLG_PRO

```
#define CLPFLG_PRO 0x00000008U
```
If set the parameter is only used within the property file (property file only).

5.12.2.6 CLPFLG_SEL

```
#define CLPFLG_SEL 0x00000010U
```
If set only the predefined constants over the corresponding key words can be selected (useful to define selections).

5.12.2.7 CLPFLG_FIX

```
#define CLPFLG_FIX 0x00000020U
```
This argument has a fixed length (only useful for strings if a typedef defines a fixed length per element, else set internally).

5.12.2.8 CLPFLG_BIN

```
#define CLPFLG_BIN 0x00000040U
```
This argument can contain binary data without null termination (length must be known or determined with a link).

5.12.2.9 CLPFLG_DMY

```
#define CLPFLG_DMY 0x00000080U
```
If set the parameter is not put in the symbol table, meaning it is only a peace of memory in the CLP structure.

5.12.2.10 CLPFLG_CNT

```
#define CLPFLG_CNT 0x00000100U
```
This link will be filled by the calculated amount of elements (useful for arrays).

5.12.2.11 CLPFLG_OID

```
#define CLPFLG_OID 0x00000200U
```
This link will be filled by the object identifier (OID) of the chosen argument (useful for overlays).

5.12.2.12 CLPFLG_IND

```
#define CLPFLG_IND 0x00000400U
```
This link will be filled with the index (position) in the CLP string (byte offset of the current key word).

5.12.2.13 CLPFLG_HID

```
#define CLPFLG_HID 0x00000800U
```
If set the parameter is not visible, meaning it is a hidden parameter.

5.12.2.14 CLPFLG_ELN

```
#define CLPFLG_ELN 0x00001000U
```
This link will be filled by the calculated length of an element (fixed types == data size, packed types == data length).

5.12.2.15 CLPFLG_SLN

```
#define CLPFLG_SLN 0x00002000U
```

This link will be filled by the calculated string length for an element (only for null-terminated strings).

5.12.2.16 CLPFLG_TLN

```
#define CLPFLG_TLN 0x00004000U
```

This link will be filled by the calculated total length for the argument (sum of all element lengths).

5.12.2.17 CLPFLG_DEF

```
#define CLPFLG_DEF 0x00010000U
```

This flag enables to use the OID as default for numbers if no value is assigned (only the keyword is used (syntax extension)).

5.12.2.18 CLPFLG_CHR

```
#define CLPFLG_CHR 0x00020000U
```

This flag will set the default method of interpretation of a binary string to local character string (DEFAULT).

5.12.2.19 CLPFLG_ASC

```
#define CLPFLG_ASC 0x00040000U
```

This flag will set the default method of interpretation of a binary string to ASCII.

5.12.2.20 CLPFLG_EBC

```
#define CLPFLG_EBC 0x00080000U
```

This flag will set the default method of interpretation of a binary string to EBCDIC.

5.12.2.21 CLPFLG_HEX

```
#define CLPFLG_HEX 0x00100000U
```

This flag will set the default method of interpretation of a binary string to hexadecimal.

5.12.2.22 CLPFLG_PDF

```
#define CLPFLG_PDF 0x00200000U
```

This flag will be set if a property value was defined from outside, it will be FALSE if the property value was hard coded in the tables.

5.12.2.23 CLPFLG_TIM

```
#define CLPFLG_TIM 0x00400000U
```

This flag mark a number as time value (only used to print out the corressponing time stamp).

5.12.2.24 CLPFLG_DYN

```
#define CLPFLG_DYN 0x00800000U
```

This flag mark a string or array as dynamic (only a pointer to allocated memory is used and must be freeed by the user).

5.12.2.25 CLPFLG_PWD

```
#define CLPFLG_PWD 0x01000000U
```

This flag will ensure that the clear value is only put into the data structure but not traced, logged or given away elsewhere.

5.12.2.26 CLPFLG_DLM

```
#define CLPFLG_DLM 0x02000000U
```

This flag ensures that fix size arrays has a empty (initialized) last element (max-1) as delimiter. Additional you enforce 0xFF at the and of a non fix size string array (size-1).

5.12.2.27 CLPFLG_UNS

```
#define CLPFLG_UNS 0x04000000U
```

Marks a number as unsigned (prevent negative values).

5.12.2.28 CLPFLG_XML

```
#define CLPFLG_XML 0x08000000U
```

Marks zero terminated string as XML path where '(' and ')' are used to replace environment variables.

5.12.2.29 CLPFLG_FIL

```
#define CLPFLG_FIL 0x10000000U
```

Marks zero terminated string as file and replace additional '∼' by HOME and corrects the prefix for different platforms.

5.12.2.30 CLPFLG_LAB

```
#define CLPFLG_LAB 0x20000000U
```

Marks zero terminated string as label and replace additional '∼' by USER, '^' by OWNER and '!' by ENVID .

5.12.2.31 CLPFLG_UPP

```
#define CLPFLG_UPP 0x40000000U
```

Converts zero terminated strings to upper case.

5.12.2.32 CLPFLG_LOW

```
#define CLPFLG_LOW 0x80000000U
```

Converts zero terminated strings to lower case.

5.13 CLP Argument Table

The structure and corresponding macros are used to define a entry for argument table.

Data Structures

- struct ClpArgument

  Table structure for arguments.

Macros

- #define CLPCONTAB_OPN(name) TsClpArgument name[ ]

  Starts a table with constant definitions.
- #define CLPCONTAB_NUMBER(kyw, dat, man, hlp) {CLPTYP_NUMBER,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON ,NULL,NULL,(man),(hlp),(dat), 0.0 ,NULL ,NULL},

  Defines a number literal with the command line keyword kyw and the value dat.
- #define CLPCONTAB_FLOATN(kyw, dat, man, hlp) {CLPTYP_FLOATN,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON ,NULL,NULL,(man),(hlp), 0 ,(dat),NULL ,NULL},

  Defines a floating point literal with the command line keyword kyw and the value dat.
- #define CLPCONTAB_STRING(kyw, dat, man, hlp) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON ,NULL,NULL,(man),(hlp), 0 , 0.0 ,(U08∗)(dat),NULL},

  Defines a default string literal with the command line keyword kyw and the value dat.
- #define CLPCONTAB_HEXSTR(kyw, dat, man, hlp) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON|CLPFLG_HEX, ULL,NULL,(man),(hlp), 0 , 0.0 ,(U08∗)(dat),NULL},

  Defines a hexadecimal string literal with the command line keyword kyw and the value dat.
- #define CLPCONTAB_ASCSTR(kyw, dat, man, hlp) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON|CLPFLG_ASC, ULL,NULL,(man),(hlp), 0 , 0.0 ,(U08∗)(dat),NULL},

  Defines a ASCII string literal with the command line keyword kyw and the value dat.
- #define CLPCONTAB_EBCSTR(kyw, dat, man, hlp) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON|CLPFLG_EBC, ULL,NULL,(man),(hlp), 0 , 0.0 ,(U08∗)(dat),NULL},

  Defines a EBCDIC string literal with the command line keyword kyw and the value dat.
- #define CLPCONTAB_BINARY(kyw, dat, siz, man, hlp) {CLPTYP_STRING,(kyw),NULL,0,0,(siz),0,0,CLPFLG_CON|CLPFLG_ ULL,NULL,(man),(hlp), 0 , 0.0 ,(U08∗)(dat),NULL},

  Defines a binary literal with the command line keyword kyw and the value dat.
- #define CLPCONTAB_CLS {CLPTYP_NON , NULL,NULL,0,0, 0 ,0,0,CLPFLG_NON ,NULL,NULL, NULL, NULL, 0 , 0.0 ,NULL ,NULL}
- #define CLPARGTAB_SKALAR(kyw, nam, typ, min, max, atyp, flg, oid, tab, dft, man, hlp) { atyp ,(kyw), NULL,(min), 1 ,sizeof(typ), offsetof(STRUCT_NAME,nam),(oid),(flg) ,(tab),(dft),(man),(hlp),0,0.0,NULL,#typ},

  Defines a scalar (single value) with the command line keyword kyw and the member name nam.
- #define CLPARGTAB_STRING(kyw, nam, siz, min, max, atyp, flg, oid, tab, dft, man, hlp) { CLPTYP_STRING,(kyw), NULL,(min),(max), (siz), offsetof(STRUCT_NAME,nam),(oid),(flg) ,(tab),(dft),(man),(hlp),0,0.↩ 0,NULL,NULL},

  Defines a string with the command line keyword kyw and the member name nam.
- #define CLPARGTAB_DYNSTR(kyw, nam, siz, min, max, atyp, flg, oid, tab, dft, man, hlp) { CLPTYP_STRING,(kyw), NULL,(min),(max), (siz), offsetof(STRUCT_NAME,nam),(oid),((flg)|CLPFLG_DYN),(tab),(dft),(man),(0,NULL,NULL},

  Defines a dynamic string with the command line keyword kyw and the member name nam (pointer to alloced memory, must be freed by the using application).
- #define CLPARGTAB_ARRAY(kyw, nam, typ, min, max, atyp, flg, oid, tab, dft, man, hlp) { atyp ,(kyw), NU↩ LL,(min),(max),sizeof(typ), offsetof(STRUCT_NAME,nam),(oid),(flg) ,(tab),(dft),(man),(hlp),0,0.0,NULL,#typ},

  Defines an array with the command line keyword kyw and the member name nam.
- #define CLPARGTAB_DYNARY(kyw, nam, typ, min, max, atyp, flg, oid, tab, dft, man, hlp) { atyp ,(kyw), NU↩ LL,(min),(max),sizeof(typ), offsetof(STRUCT_NAME,nam),(oid),((flg)|CLPFLG_DYN),(tab),(dft),(man),(hlp),0,0.↩ 0,NULL,#typ},

Defines an dynamic array with the command line keyword kyw and the member name nam (pointer to alloced memory, must be freed by the using application).

- #define CLPARGTAB_ALIAS(kyw, ali) { CLPTYP_XALIAS,(kyw),(ali), 0 , 0 , 0 , 0 , 0 , CLPFLG_ALI, NULL, NULL, NULL, NULL,0,0.0,NULL,NULL},

  Defines an alias name for another argument.
- #define CLPARGTAB_CLS { CLPTYP_NON , NULL, NULL, 0 , 0 , 0 , 0 , 0 , 0 , NULL, NULL, NULL, NULL,0,0.0,NULL,NULL}

  Will mark the end of an argument table.

Typedefs

- typedef struct ClpArgument TsClpArgument

  Table structure for arguments.

5.13.1 Detailed Description

The structure and corresponding macros are used to define a entry for argument table.

5.13.2 Macro Definition Documentation

5.13.2.1 CLPCONTAB_OPN

```
#define CLPCONTAB_OPN(
              name ) TsClpArgument name[]
```

Starts a table with constant definitions.

Parameters

in	*name*	Name of this table

5.13.2.2 CLPCONTAB_NUMBER

```
#define CLPCONTAB_NUMBER(
              kyw,
              dat,
              man,
              hlp ) {CLPTYP_NUMBER,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON ,NULL,NULL,(man),(hlp),(dat),
0.0 ,NULL ,NULL},
```

Defines a number literal with the command line keyword *kyw* and the value *dat*.

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*dat*	Pointer to target definition in case of reading.
in	*man*	Pointer to a null-terminated string for a detailed description of this constant (in ASCIIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs) Can be a NULL pointer or empty string to produce a bullet list. It is recommended to use a header file with a define for this long string.
in	*hlp*	Pointer to a null-terminated string for context sensitive help for this constant (also used as head line or in bullet list in documentation generation).

5.13.2.3 CLPCONTAB_FLOATN

```
#define CLPCONTAB_FLOATN(
              kyw,
              dat,
              man,
              hlp ) {CLPTYP_FLOATN,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON ,NULL,NULL,(man),(hlp), 0
,(dat),NULL ,NULL},
```

Defines a floating point literal with the command line keyword *kyw* and the value *dat*.

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*dat*	Pointer to target definition in case of reading.
in	*man*	Pointer to a null-terminated string for a detailed description of this constant (in ASCIIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs) Can be a NULL pointer or empty string to produce a bullet list. It is recommended to use a header file with a define for this long string.
in	*hlp*	Pointer to a null-terminated string for context sensitive help for this constant (also used as head line or in bullet list in documentation generation).

5.13.2.4 CLPCONTAB_STRING

```
#define CLPCONTAB_STRING(
              kyw,
              dat,
              man,
              hlp ) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON ,NULL,NULL,(man),(hlp), 0
, 0.0 ,(U08*)(dat),NULL},
```

Defines a default string literal with the command line keyword *kyw* and the value *dat*.

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*dat*	Pointer to target definition in case of reading.
in	*man*	Pointer to a null-terminated string for a detailed description of this constant (in ASCIIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs) Can be a NULL pointer or empty string to produce a bullet list. It is recommended to use a header file with a define for this long string.
in	*hlp*	Pointer to a null-terminated string for context sensitive help for this constant (also used as head line or in bullet list in documentation generation).

5.13.2.5 CLPCONTAB_HEXSTR

```
#define CLPCONTAB_HEXSTR(
              kyw,
              dat,
              man,
              hlp ) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON|CLPFLG_HEX,NULL,NULL,(man),(hlp),
0 , 0.0 ,(U08*)(dat),NULL},
```

Defines a hexadecimal string literal with the command line keyword *kyw* and the value *dat*.

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*dat*	Pointer to target definition in case of reading.
in	*man*	Pointer to a null-terminated string for a detailed description of this constant (in ASCIIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs) Can be a NULL pointer or empty string to produce a bullet list. It is recommended to use a header file with a define for this long string.
in	*hlp*	Pointer to a null-terminated string for context sensitive help for this constant (also used as head line or in bullet list in documentation generation).

5.13.2.6 CLPCONTAB_ASCSTR

```
#define CLPCONTAB_ASCSTR(
              kyw,
              dat,
              man,
              hlp ) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON|CLPFLG_ASC,NULL,NULL,(man),(hlp),
0 , 0.0 ,(U08*)(dat),NULL},
```

Defines a ASCII string literal with the command line keyword *kyw* and the value *dat*.

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*dat*	Pointer to target definition in case of reading.
in	*man*	Pointer to a null-terminated string for a detailed description of this constant (in ASCIIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs) Can be a NULL pointer or empty string to produce a bullet list. It is recommended to use a header file with a define for this long string.
in	*hlp*	Pointer to a null-terminated string for context sensitive help for this constant (also used as head line or in bullet list in documentation generation).

5.13.2.7 CLPCONTAB_EBCSTR

```
#define CLPCONTAB_EBCSTR(
              kyw,
              dat,
              man,
              hlp ) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON|CLPFLG_EBC,NULL,NULL,(man),(hlp),
0 , 0.0 ,(U08*)(dat),NULL},
```

Defines a EBCDIC string literal with the command line keyword *kyw* and the value *dat*.

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*dat*	Pointer to target definition in case of reading.
in	*man*	Pointer to a null-terminated string for a detailed description of this constant (in ASCIIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs) Can be a NULL pointer or empty string to produce a bullet list. It is recommended to use a header file with a define for this long string.
in	*hlp*	Pointer to a null-terminated string for context sensitive help for this constant (also used as head line or in bullet list in documentation generation).

5.13.2.8 CLPCONTAB_BINARY

```
#define CLPCONTAB_BINARY(
              kyw,
              dat,
              siz,
              man,
              hlp ) {CLPTYP_STRING,(kyw),NULL,0,0,(siz),0,0,CLPFLG_CON|CLPFLG_BIN,NULL,NU↩
LL,(man),(hlp), 0 , 0.0 ,(U08*)(dat),NULL},
```

Defines a binary literal with the command line keyword *kyw* and the value *dat*.

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*dat*	Pointer to target definition in case of reading.
in	*siz*	Size of the binary value.
in	*man*	Pointer to a null-terminated string for a detailed description of this constant (in ASCIIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs) Can be a NULL pointer or empty string to produce a bullet list. It is recommended to use a header file with a define for this long string.
in	*hlp*	Pointer to a null-terminated string for context sensitive help for this constant (also used as head line or in bullet list in documentation generation).

5.13.2.9 CLPCONTAB_CLS

```
#define CLPCONTAB_CLS {CLPTYP_NON , NULL,NULL,0,0, 0 ,0,0,CLPFLG_NON ,NULL,NULL, NULL, NULL, 0
, 0.0 ,NULL ,NULL}
```

Ends a table with constant definitions

5.13.2.10 CLPARGTAB_SKALAR

```
#define CLPARGTAB_SKALAR(
              kyw,
              nam,
              typ,
              min,
              max,
              atyp,
              flg,
              oid,
              tab,
              dft,
              man,
              hlp ) { atyp ,(kyw), NULL,(min), 1 ,sizeof(typ), offsetof(STRUCT_NAME,nam),(oid),(flg)
,(tab),(dft),(man),(hlp),0,0.0,NULL,#typ},
```

Defines a scalar (single value) with the command line keyword *kyw* and the member name *nam*.

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*nam*	Pointer to target definition in case of reading.
in	*typ*	is the C type of the member.
in	*min*	can be 0 or 1 for a scalar and determines if this argument is optional(0) or required(1).

Parameters

in	*max*	is unused for scalar values.
in	*atyp*	is one of the CLPTYP_∗ macros.
in	*flg*	is an OR-ed list of the flag macros CLPFLG_∗ which define various parsing options.
in	*oid*	is a unique id value used for this argument.
in	*tab*	is NULL or a pointer to another argument table describing the object, overlay or selection.
in	*dft*	is the hard coded default value of the argument if no input is given on the command line.
in	*man*	is a pointer to the long description of the argument.
in	*hlp*	is a pointer to the short description of the argument.

5.13.2.11 CLPARGTAB_STRING

```
#define CLPARGTAB_STRING(
              kyw,
              nam,
              siz,
              min,
              max,
              atyp,
              flg,
              oid,
              tab,
              dft,
              man,
              hlp ) { CLPTYP_STRING,(kyw), NULL,(min),(max), (siz), offsetof(STRUCT_NAME,nam),(oid),(flg)
,(tab),(dft),(man),(hlp),0,0.0,NULL,NULL},
```

Defines a string with the command line keyword *kyw* and the member name *nam*.

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*nam*	Pointer to target definition in case of reading.
in	*siz*	is the total available length of the string(s).
in	*min*	can be between 0 and *max* for a string and determines if this argument is optional(0) or required(min>=1).
in	*max*	defines the maximum number of strings accepted on the command line.
in	*atyp*	is unused and fixed to CLPTYP_STRING.
in	*flg*	is an OR-ed list of the flag macros CLPFLG_∗ which define various parsing options.
in	*oid*	is a unique id value used for this argument.
in	*tab*	is NULL or a pointer to another argument table describing a selection.
in	*dft*	is the hard coded default value of the argument if no input is given on the command line.
in	*man*	is a pointer to the long description of the argument.
in	*hlp*	is a pointer to the short description of the argument.

5.13.2.12 CLPARGTAB_DYNSTR

```
#define CLPARGTAB_DYNSTR(
              kyw,
              nam,
```

```
                siz,
                min,
                max,
                atyp,
                flg,
                oid,
                tab,
                dft,
                man,
                hlp ) { CLPTYP_STRING,(kyw), NULL,(min),(max), (siz), offsetof(STRUCT_NAME,nam),(oid),((flg)|CLPF
0,NULL,NULL},
```

Defines a dynamic string with the command line keyword *kyw* and the member name *nam* (pointer to alloced memory, must be freed by the using application).

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*nam*	Pointer to target definition in case of reading.
in	*siz*	is the total available length of the string(s).
in	*atyp*	is unused and fixed to CLPTYP_STRING.
in	*min*	can be between 0 and *max* for a string and determines if this argument is optional(0) or required(min>=1).
in	*max*	defines the maximum number of strings accepted on the command line.
in	*flg*	is an OR-ed list of the flag macros CLPFLG_* which define various parsing options.
in	*oid*	is a unique id value used for this argument.
in	*tab*	is NULL or a pointer to another argument table describing a selection.
in	*dft*	is the hard coded default value of the argument if no input is given on the command line.
in	*man*	is a pointer to the long description of the argument.
in	*hlp*	is a pointer to the short description of the argument.

5.13.2.13 CLPARGTAB_ARRAY

```
#define CLPARGTAB_ARRAY(
                kyw,
                nam,
                typ,
                min,
                max,
                atyp,
                flg,
                oid,
                tab,
                dft,
                man,
                hlp ) { atyp ,(kyw), NULL,(min),(max),sizeof(typ), offsetof(STRUCT_NAME,nam),(oid),(flg)
,(tab),(dft),(man),(hlp),0,0.0,NULL,#typ},
```

Defines an array with the command line keyword *kyw* and the member name *nam*.

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*nam*	Pointer to target definition in case of reading.
in	*typ*	is the C type of the member.
in	*min*	can be between 0 or *max* and determines if this argument is optional(0) or required(min>=1).

Parameters

in	*max*	defines the size of the array.
in	*atyp*	is one of the CLPTYP_∗ macros.
in	*flg*	is an OR-ed list of the flag macros CLPFLG_∗ which define various parsing options.
in	*oid*	is a unique id value used for this argument.
in	*tab*	is NULL or a pointer to another argument table describing the object, overlay or selection.
in	*dft*	is the hard coded default value of the argument if no input is given on the command line.
in	*man*	is a pointer to the long description of the argument.
in	*hlp*	is a pointer to the short description of the argument.

5.13.2.14 CLPARGTAB_DYNARY

```
#define CLPARGTAB_DYNARY(
              kyw,
              nam,
              typ,
              min,
              max,
              atyp,
              flg,
              oid,
              tab,
              dft,
              man,
              hlp ) { atyp ,(kyw), NULL,(min),(max),sizeof(typ), offsetof(STRUCT_NAME,nam),(oid),((flg)|CLPFLG_
0,NULL,#typ},
```

Defines an dynamic array with the command line keyword *kyw* and the member name *nam* (pointer to alloced memory, must be freed by the using application).

Parameters

in	*kyw*	Pointer to command line keyword *kyw*.
in	*nam*	Pointer to target definition in case of reading.
in	*typ*	is the C type of the member.
in	*min*	can be between 0 or *max* and determines if this argument is optional(0) or required(min>=1).
in	*max*	defines the size of the array.
in	*atyp*	is one of the CLPTYP_∗ macros.
in	*flg*	is an OR-ed list of the flag macros CLPFLG_∗ which define various parsing options.
in	*oid*	is a unique id value used for this argument.
in	*tab*	is NULL or a pointer to another argument table describing the object, overlay or selection.
in	*dft*	is the hard coded default value of the argument if no input is given on the command line.
in	*man*	is a pointer to the long description of the argument.
in	*hlp*	is a pointer to the short description of the argument.

5.13.2.15 CLPARGTAB_ALIAS

```
#define CLPARGTAB_ALIAS(
              kyw,
              ali ) { CLPTYP_XALIAS,(kyw),(ali), 0 , 0 , 0 , 0 , 0 , CLPFLG_ALI, NULL, NULL,
```

NULL, NULL,0,0.0,NULL,NULL},
Defines an alias name for another argument.

Parameters

in	*kyw*	is the keyword accepted on the command line in place of the keyword given in *ali*.
in	*ali*	is the alternative keyword accepted on the command line in place of the keyword given in *kyw*.

5.13.2.16 CLPARGTAB_CLS

```
#define CLPARGTAB_CLS { CLPTYP_NON , NULL, NULL, 0 , 0 , 0 , 0 , 0 , 0 , NULL, NULL, NULL, N↩
ULL,0,0.0,NULL,NULL}
```
Will mark the end of an argument table.

5.13.3 Typedef Documentation

5.13.3.1 TsClpArgument

```
typedef struct ClpArgument TsClpArgument
```
Table structure for arguments.
To simplify the definition of the corresponding data structures and argument tables it is recommended to use the CLPARGTAB macros defined in CLPMAC.h or for constant definitions the CLPCONTAB macros below. With the CLPMAC.h you can generate the tables or the corresponding data structures, depending if DEFINE_STRUCT defined or not.
Example:
First you must define the table:

```
#define CLPINTFMTRED_TABLE\
 CLPARGTAB_SKALAR("BIN"          , stBin , TsClpIntRedBin , 0, 1, CLPTYP_OBJECT, CLPFLG_NON,
       INTCNV_FORMAT_BIN , asClpIntRedBin , NULL, MAN_INTRED_BIN, "Integer in binary format (two's
       complement)")\
 CLPARGTAB_SKALAR("BCD"          , stBcd , TsClpIntRedBcd , 0, 1, CLPTYP_OBJECT, CLPFLG_NON,
       INTCNV_FORMAT_BCD , asClpIntRedBcd , NULL, MAN_INTRED_BCD, "Binary coded decimal (BCD) number")\
 CLPARGTAB_SKALAR("STR"          , stStr , TsClpIntRedStr , 0, 1, CLPTYP_OBJECT, CLPFLG_NON,
       INTCNV_FORMAT_STR , asClpIntRedStr , NULL, MAN_INTRED_STR, "String representation of an integer")\
 CLPARGTAB_SKALAR("ENUM"         , stEnum, TsClpIntRedSel , 0, 1, CLPTYP_OBJECT, CLPFLG_NON,
       INTCNV_FORMAT_ENUM, asClpIntRedEnum, NULL, MAN_INTRED_ENUM,"Maps enumeration to an integer")\
 CLPARGTAB_CLS
```
Then you can use this table to define your structure or union, in our case we create a union:

```
#define DEFINE_STRUCT
#include "CLPMAC.h"
typedef union ClpIntFmtRed{
   CLPINTFMTRED_TABLE
}TuClpIntFmtRed;
```
And you use the same define to allocate the corresponding CLP table:

```
#undef DEFINE_STRUCT
#include "CLPMAC.h"
#undef  STRUCT_NAME
#define STRUCT_NAME TuClpIntFmtRed
TsClpArgument       asClpIntFmtRed[] = {
   CLPINTFMTRED_TABLE
};
```

5.14 CLP Function Pointer (call backs)

Function prototype definitions for call backs used by CLP.

Typedefs

- typedef int() TfF2S(void *pvGbl, void *pvHdl, const char *pcFil, char **ppBuf, int *piBuf, char *pcMsg, const int siMsg)

  Type definition for string to file call back function.
- typedef int() TfSaf(void *pvGbl, void *pvHdl, const char *pcVal)

  Type definition for resource access check.
- typedef int() TfClpPrintPage(void *pvHdl, const int siLev, const char *pcHdl, const char *pcPat, const char *pcFil, const char *pcOrg, const char *pcPge)

  Function 'prnHtmlDoc' of library 'libhtmldoc' called if built-in function HTMLDOC used.

5.14.1 Detailed Description

Function prototype definitions for call backs used by CLP.

5.14.2 Typedef Documentation

5.14.2.1 TfF2S

```
typedef int() TfF2S(void *pvGbl, void *pvHdl, const char *pcFil, char **ppBuf, int *piBuf, char *pcMsg, const int siMsg)
```

Type definition for string to file call back function.
Read a file using the specified filename and reads the whole content into the supplied buffer. The buffer is reallocated and buffer size updated, if necessary.

Parameters

in	*pvGbl*	Pointer to to the global handle as black box given with CleExecute
in	*pvHdl*	Pointer to a handle given for this callback
in	*pcFil*	File name to read
in,out	*ppBuf*	Pointer to a buffer pointer for reallocation
in,out	*piBuf*	Pointer to the buffer size (updated after reallocation)
out	*pcMsg*	Pointer to a buffer for the error message
in	*siMsg*	Size of the message buffer (should be 1024)

Returns

bytes read or negative value if error

5.14.2.2 TfSaf

```
typedef int() TfSaf(void *pvGbl, void *pvHdl, const char *pcVal)
```

Type definition for resource access check.
The function is called with the complete path and the standard lexeme as value in front of each wrte of data to the CLP structure.

Parameters

in	*pvGbl*	Pointer to to the global handle as black box given with CleExecute

Parameters

in	*pvHdl*	Pointer to a handle given for this callback
in	*pcVal*	Path=Value as resource

Returns

0 if write allowed else a authorization error

5.14.2.3 TfClpPrintPage

```
typedef int() TfClpPrintPage(void *pvHdl, const int siLev, const char *pcHdl, const char *pc↩
Pat, const char *pcFil, const char *pcOrg, const char *pcPge)
```

Function 'prnHtmlDoc' of library 'libhtmldoc' called if built-in function HTMLDOC used.
The built-in function HTMLDOC use a service provider interface to create the documentation using a callback function for each page/chapter. This is this callback function. This interface is used with the function siClpPrint↩ Docu() below. The function is called for each page/chapter and the pointer to the original manual page (pcOrg) can be used to determine if this page the first time printed or if the same page printed again. The HTML documentation can now use an link instead to print the same page again to reduce memory and redundancies in the document. The level can be used to insert Headlines to a dictionary, the path shows the real position and can be used to build files names. The path starts always with CLEP followed by COMMAND or OTHERCLP depending which tables are used. This prefix ensures uniqueness if it used for the file names. prepared page is in ASCIIDOC and must be converted to HTML. In this case the headline must be provided for the dictionary, the index terms for the index and other information can be used to build a very powerful HTML documentation.

Parameters

in,out	*pvHdl*	Handle for the print callback function (e.G. from opnHtmlDoc)
in	*siLev*	The hierarchical level for this page/chapter
in	*pcHdl*	The headline of the current chapter
in	*pcPat*	The path for the corresponding parameter (NULL if the page not inside a command or other CLP string)
in	*pcFil*	Unique hierarchical string (can be used as file name (no extension))
in	*pcOrg*	The pointer to the original manual page (can be used to determine duplicates and produce links)
in	*pcPge*	The prepared ASCIIDOC page for printing (must be converted to HTML)

Returns

Return code (0 is OK else error)

5.15 CLE Functions

The function provided by CLE.

Functions

- const char * pcCleVersion (const int l, const int s, char *b)

  Get CLE-version information.
- const char * pcCleAbout (const int l, const int s, char *b)

  Get about CLE-information.
- int siCleExecute (void *pvGbl, const TsCleCommand *psCmd, int argc, char *argv[ ], const char *pcOwn, const char *pcPgm, const char *pcAut, const char *pcAdr, const int isCas, const int isPfl, const int isRpl, const int isEnv, const int siMkl, FILE *pfOut, FILE *pfTrc, const char *pcDep, const char *pcOpt, const char *pcEnt, const char *pcLic, const char *pcBld, const char *pcVsn, const char *pcAbo, const char *pcHlp, const char *pcDef, TfMsg *pfMsg, const TsCleOtherClp *psOth, void *pvF2S, TfF2S *pfF2S, void *pvSaf, TfSaf *pfSaf, const char *pcDpa, const int siNoR, const TsCleDoc *psDoc)

  Execute CLE-command line.

5.15.1 Detailed Description

The function provided by CLE.

5.15.2 Function Documentation

5.15.2.1 pcCleVersion()

```
const char* pcCleVersion (
            const int l,
            const int s,
            char * b )
```

Get CLE-version information.
The function returns the version information for this library

Parameters

in	*l*	level of visible hierarchy in the first 2 numbers of the string the number can later be used to better visualize the hierarchy
in	*s*	size of the provided string buffer (including space for null termination)
in,out	*b*	buffer for the version string must contain a null-terminated string the version string will be concatenated the size including the 0-byte is the limit if (strlen(b)==s-1) then more space is required for the complete version string a good size for the version string is 256 byte

Returns

pointer to a null-terminated version string (return(b))

5.15.2.2 pcCleAbout()

```
const char* pcCleAbout (
            const int l,
            const int s,
            char * b )
```

Get about CLE-information.

The function returns the about information for this library

Parameters

l	level of visible hierarchy in the first 2 numbers of the string the number can later be used to better visualize the hierarchy
s	size of the provided string buffer (including space for null termination)
b	buffer for the about string must contain a null-terminated string the about string will be concatenated the size including the 0-byte is the limit if (strlen(b)==s-1) then more space is required for the complete about string a good size for the about string is 1024 byte

Returns

pointer to a null-terminated about string (return(b))

5.15.2.3 siCleExecute()

```
int siCleExecute (
            void * pvGbl,
            const TsCleCommand * psCmd,
            int argc,
            char * argv[],
            const char * pcOwn,
            const char * pcPgm,
            const char * pcAut,
            const char * pcAdr,
            const int isCas,
            const int isPfl,
            const int isRpl,
            const int isEnv,
            const int siMkl,
            FILE * pfOut,
            FILE * pfTrc,
            const char * pcDep,
            const char * pcOpt,
            const char * pcEnt,
            const char * pcLic,
            const char * pcBld,
            const char * pcVsn,
            const char * pcAbo,
            const char * pcHlp,
            const char * pcDef,
            TfMsg * pfMsg,
            const TsCleOtherClp * psOth,
            void * pvF2S,
            TfF2S * pfF2S,
            void * pvSaf,
            TfSaf * pfSaf,
            const char * pcDpa,
            const int siNoR,
            const TsCleDoc * psDoc )
```

Execute CLE-command line.
The function uses the command line parsers to execute different commands based on argc and argv given in the main function of a program and provides the additional built-in functions below:

- SYNTAX [command[.path] [DEPTH1 | ... | DEPTH9 | ALL]]
- HELP [command[.path] [DEPTH1 | ... | DEPTH9 | ALL]] [MAN]
- MANPAGE [function | command[.path][=filename]] | [filename]

- GENDOCU [command[.path]=]filename [NONBR] [SHORT]
- HTMLDOC [path] [NUMBERS]
- GENPROP [command=]filename
- SETPROP [command=]filename
- CHGPROP command [path[=value]]∗
- DELPROP [command]
- GETPROP [command[.path] [DEPTH1 | ... | DEPTH9 | DEPALL | DEFALL]]
- SETOWNER name
- GETOWNER
- SETENV variable=name
- GETENV
- DELENV variable
- TRACE ON | OFF | FILE=filename
- CONFIG [CLEAR]
- GRAMMAR
- LEXEMES
- LICENSE
- VERSION
- ABOUT
- ERRORS

Parameters

in	*pvGbl*	Pointer to a global handle given to called functions in the command table
in	*psCmd*	Pointer to the table which defines the commands
in	*argc*	Number of command line parameters (argc of main(int argc, char∗ argv[ ]))
in	*argv*	List of pointers to the command line parameters (argv of main(int argc, char∗ argv[ ]))
in	*pcOwn*	Default owner id (owner ids are used to identify properties and other things "com.company")
in	*pcPgm*	Logical program name (can be different from argv[0] and will be used in the root "com.company.program")
in	*pcAut*	Name of the author for ASCIIDOC header (required for header generation)
in	*pcAdr*	Mail address of the author for the ASCIIDOC header (optional)
in	*isCas*	Switch to enable case sensitive interpretation of the command line (recommended is FALSE)
in	*isPfl*	Switch to enable parameter file support for object, overlays and arrays (recommended is TRUE)
in	*isRpl*	Switch to enable replacement of environment variables (recommended is TRUE)
in	*isEnv*	Switch to load environment variables from default files (recommended is TRUE if no own load done else FALSE)
in	*siMkl*	Integer defining the minimal key word length (siMkl<=0 --> full length, no auto abbreviation)
in	*pfOut*	File pointer for help and error messages (if not defined stderr will be used)
in	*pfTrc*	Default trace file if no trace file is defined with the configuration data management (recommended: NULL, stdout or stderr)

Parameters

in	*pcDep*	String to visualize hierarchies (recommended: "--|" converted on EBCDIC systems (don't use S_IDT))
in	*pcOpt*	String to separate options (recommended: "/" converted on EBCDIC systems)
in	*pcEnt*	String to separate list entries (recommended: "," converted on EBCDIC systems)
in	*pcLic*	String containing the license information for this program (used by built-in function LICENSE - not converted on EBCDIC systems (don't use dia-critical characters))
in	*pcBld*	String containing the build number / raw version for this program (optional, can be NULL) used in final message and replacements - converted on EBCDIC systems
in	*pcVsn*	String containing the version information for this program (used by built-in function VERSION - not converted on EBCDIC systems (don't use dia-critical characters))
in	*pcAbo*	String containing the about message for this program (used by built-in function ABOUT - not converted on EBCDIC systems (don't use dia-critical characters))
in	*pcHlp*	Short help message for the whole program (converted on EBCDIC systems)
in	*pcDef*	Default command or built-in function, which is executed if the first keyword (argv[1]) don't match (if NULL then no default)
in	*pfMsg*	Pointer to a function which prints a message for an reason code (use to generate the corresponding appendix)
in	*psOth*	Pointer to the table with other CLP strings to print (optional could be NULL)
in	*pvF2S*	Pointer to a handle which can be used in file 2 string callback function (if not required then NULL)
in	*pfF2S*	Callback function which reads a file into a null-terminated string in memory (if NULL then default implementation is used)
in	*pvSaf*	Pointer to a handle which can be used in authorization callback function (if not required then NULL)
in	*pfSaf*	Callback function for additional authorization by CLP or NULL if no authorization is requested
in	*pcDpa*	Pointer to a file name for a default parameter file (e.g. "DD:FLAMPAR") or NULL/empty string for nothing, The file name is used if only a command without assignment or parameter is provided
in	*siNoR*	Define this reason code to the values the mapping function returns if no run is requested (0 is nothing)
in	*psDoc*	Table for documentation generation (must be defined)

Returns

signed integer with the condition codes below:
0 - command line, command syntax, mapping, execution and finish of the command was successful
1 - command line, command syntax, mapping, execution and finish of the command was successful but a warning can be found in the log
2 - command line, command syntax, mapping, execution was successful but cleanup of the command failed (may not happened)
4 - command line, command syntax and mapping was successful but execution of the command returns with a warning
8 - command line, command syntax and mapping was successful but execution of the command returns with an error
12 - command line and command syntax was OK but mapping failed
16 - command line was OK but command syntax was wrong
20 - command line was wrong (user error)
24 - initialization of parameter structure for the command failed (may not happened)
28 - configuration is wrong (user error)
32 - table error (something within the predefined tables is wrong)
36 - system error (mainly memory allocation or some thing like this failed)
40 - access control or license error

44 - interface error (parameter pointer equals to NULL or something like this)
48 - memory allocation failed (e.g. dynamic string handling)
64 - fatal error (basic things are damaged)
>64 - Special condition code for job control

5.16 CLP Functions

The function provided by CLP.

Functions

- const char * pcClpVersion (const int l, const int s, char *b)

  Get version information.
- const char * pcClpAbout (const int l, const int s, char *b)

  Get about information.
- void * pvClpOpen (const int isCas, const int isPfl, const int isEnv, const int siMkl, const char *pcOwn, const char *pcPgm, const char *pcBld, const char *pcCmd, const char *pcMan, const char *pcHlp, const int isOvl, const TsClpArgument *psTab, void *pvDat, FILE *pfHlp, FILE *pfErr, FILE *pfSym, FILE *pfScn, FILE *pf↩ Prs, FILE *pfBld, const char *pcDep, const char *pcOpt, const char *pcEnt, TsClpError *psErr, void *pvGbl, void *pvF2S, TfF2S *pfF2S, void *pvSaf, TfSaf *pfSaf)

  Open command line parser.
- void vdClpReset (void *pvHdl)

  Reset command line parser.
- int siClpParsePro (void *pvHdl, const char *pcSrc, const char *pcPro, const int isChk, char **ppLst)

  Parse the property list.
- int siClpParseCmd (void *pvHdl, const char *pcSrc, const char *pcCmd, const int isChk, const int isPwd, int *piOid, char **ppLst)

  Parse the command line.
- int siClpSyntax (void *pvHdl, const int isSkr, const int isMin, const int siDep, const char *pcPat)

  Print command line syntax.
- const char * pcClpInfo (void *pvHdl, const char *pcPat)

  Give help message for given path.
- int siClpHelp (void *pvHdl, const int siDep, const char *pcPat, const int isAli, const int isMan)

  Print help for command line syntax.
- int siClpDocu (void *pvHdl, FILE *pfDoc, const char *pcPat, const char *pcNum, const char *pcKnd, const int isCmd, const int isDep, const int isMan, const int isAnc, const int isNbr, const int isIdt, const int isPat, const unsigned int uiLev)

  Generate documentation for command line syntax.
- int siClpPrint (void *pvHdl, const char *pcFil, const char *pcNum, const char *pcKnd, const int isCmd, const int isDep, const int isAnc, const int isNbr, const int isShl, const int isIdt, const int isPat, const unsigned int uiLev, const int siPs1, const int siPs2, const int siPr3, void *pvPrn, TfClpPrintPage *pfPrn)

  Generate documentation using a callback function.
- int siClpProperties (void *pvHdl, const int siMtd, const int siDep, const char *pcPat, FILE *pfOut)

  Generate properties.
- int siClpLexemes (void *pvHdl, FILE *pfOut)

  Print the lexems of the command line compiler.
- int siClpGrammar (void *pvHdl, FILE *pfOut)

  Print the grammar of the command line compiler.
- void vdClpClose (void *pvHdl, const int siMtd)

  Close the command line parser.
- void * pvClpAlloc (void *pvHdl, void *pvPtr, int siSiz, int *piInd)

  Allocate memory in CLP structure.
- char * pcClpError (int siErr)

  Provides error message.

5.16.1 Detailed Description

The function provided by CLP.

5.16.2 Function Documentation

5.16.2.1 pcClpVersion()

```
const char* pcClpVersion (
            const int l,
            const int s,
            char * b )
```

Get version information.
The function returns the version information for this library

Parameters

in	*l*	Level of visible hierarchy in the first 2 numbers of the string the number can later be used to better visualize the hierarchy
in	*s*	Size of the provided string buffer (including space for null termination)
in,out	*b*	Buffer for the version string. Must contain a null-terminated string. The version string will be concatenated. The size including the 0-byte is the limit. If (strlen(b)==s-1) then more space is required for the complete version string. A good size for the version string is 128 byte.

Returns

Pointer to a null-terminated version string (return(b))

5.16.2.2 pcClpAbout()

```
const char* pcClpAbout (
            const int l,
            const int s,
            char * b )
```

Get about information.
The function returns the about information for this library

Parameters

in	*l*	Level of visible hierarchy in the first 2 numbers of the string the number can later be used to better visualize the hierarchy
in	*s*	Size of the provided string buffer (including space for null termination)
in,out	*b*	Buffer for the about string. Must contain a null-terminated string. The about string will be concatenated. The size including the 0-byte is the limit. If (strlen(b)==s-1) then more space is required for the complete about string. A good size for the about string is 512 byte.

Returns

pointer to a null-terminated about string (return(b))

5.16.2.3 pvClpOpen()

```
void* pvClpOpen (
            const int isCas,
            const int isPfl,
            const int isEnv,
```

```
const int siMkl,
const char * pcOwn,
const char * pcPgm,
const char * pcBld,
const char * pcCmd,
const char * pcMan,
const char * pcHlp,
const int isOvl,
const TsClpArgument * psTab,
void * pvDat,
FILE * pfHlp,
FILE * pfErr,
FILE * pfSym,
FILE * pfScn,
FILE * pfPrs,
FILE * pfBld,
const char * pcDep,
const char * pcOpt,
const char * pcEnt,
TsClpError * psErr,
void * pvGbl,
void * pvF2S,
TfF2S * pfF2S,
void * pvSaf,
TfSaf * pfSaf )
```

Open command line parser.
The function uses the argument table and corresponding structure and creates the handle for the command line parser (FLAMCLP)

Parameters

in	*isCas*	Boolean to enable case sensitive parsing of keywords (recommended is FALSE)
in	*isPfl*	Boolean to enable parameter files per object and overlay (recommended is TRUE(1), if you provide 2 the parameter file is not parsed but the syntax is accepted)
in	*isEnv*	Boolean to enable replacement of environment variables (recommended is TRUE)
in	*siMkl*	Integer defining the minimal key word length (siMkl<=0 --> full length, no auto abbreviation)
in	*pcOwn*	String constant containing the owner name for the root in the path ("limes")
in	*pcPgm*	String constant containing the program name for the root in the path ("flcl")
in	*pcBld*	String constant containing the build/version string for replacement
in	*pcCmd*	String constant containing the command name for the root in the path ("CONV")
in	*pcMan*	String constant containing the manual page for this command (converted on EBCDIC systems)
in	*pcHlp*	String constant containing the help message for this command (converted on EBCDIC systems)
in	*isOvl*	Boolean if TRUE the main table (psTab) is a overlay else it will be interpreted as object
in	*psTab*	Pointer to the parameter table defining the semantic of the command line
out	*pvDat*	Pointer to the structure where the parsed values are stored (can be NULL if command line parsing not used)
in	*pfHlp*	Pointer to the file used for help messages (if not set then stderr)
in	*pfErr*	Pointer to the file used for error messages or NULL for no printing
in	*pfSym*	Pointer to the file used for symbol table trace or NULL for no printing
in	*pfScn*	Pointer to the file used for scanner trace or NULL for no printing
in	*pfPrs*	Pointer to the file used for parser trace or NULL for no printing
in	*pfBld*	Pointer to the file used for builder trace or NULL for no printing

Parameters

in	*pcDep*	String used for hierarchical print outs (help, errors, trace (recommended S_IDT="--|"))
in	*pcOpt*	String used to separate options (recommended "/")
in	*pcEnt*	String used to separate list entries (recommended ",")
out	*psErr*	Pointer to the error structure. If the pointer != NULL the structure is filled with pointers to certain error information in the CLP handle. If pfErr defined all error information are printed by CLP. In this case these structure is not required. If pfErr==NULL you can use these structure to gather all error information of CLP in memory. The pointer are only valid until vsClpClose().
in	*pvGbl*	Pointer to the global handle as black box given over CleExecute
in	*pvF2S*	Pointer to a handle which can be used in file 2 string callback function (if not required then NULL)
in	*pfF2S*	Callback function which reads a file into a variable null-terminated string in memory (if NULL then default implementation is used)
in	*pvSaf*	Pointer to a handle which can be used in authorization callback function (if not required then NULL)
in	*pfSaf*	Callback function to authorize each write to CLP structure or NULL for no additional check

Returns

void pointer to the memory containing the handle

5.16.2.4 vdClpReset()

```
void vdClpReset (
            void * pvHdl )
```

Reset command line parser.
Required after an error which was handled by the calling application to parse properties or commands correctly

Parameters

in,out	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*

5.16.2.5 siClpParsePro()

```
int siClpParsePro (
            void * pvHdl,
            const char * pcSrc,
            const char * pcPro,
            const int isChk,
            char ** ppLst )
```

Parse the property list.
The function parses the property list
Attention: Property parsing only effects the default values in the symbol table and don't write anything to the CLP structure. You must use the same CLP handle for property and command line parsing.

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*pcSrc*	Pointer to a null-terminated string containing the source name for the property list Property list are mainly taken from a file. It is useful to provide this file name for error printing

Parameters

in	*pcPro*	Pointer to a null-terminated string containing the property list for parsing
in	*isChk*	Boolean to enable (TRUE) or disable (FALSE) validation of the root in path (if FALSE then other properties are ignored, if TRUE then other properties are not possible)
out	*ppLst*	Pointer to the parsed parameter list (NULL = no list provided) in the CLP handle

Returns

signed integer with CLP_OK (0 - nothing parsed) or an error code (CLPERR_xxxxxx (<0)) or the amount of parsed entities (>0)

5.16.2.6 siClpParseCmd()

```
int siClpParseCmd (
            void * pvHdl,
            const char * pcSrc,
            const char * pcCmd,
            const int isChk,
            const int isPwd,
            int * piOid,
            char ** ppLst )
```

Parse the command line.
The function parses the command line and returns OK or the error code and error position (byte offset)

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*pcSrc*	Pointer to a null-terminated string containing the source name for the command line If the command line taken from a file it is useful to provide this file name for error printing else use NULL.
in	*pcCmd*	Pointer to a null-terminated string containing the command for parsing
in	*isChk*	Boolean to enable (TRUE) or disable (FALSE) validation of minimum number of entries
in	*isPwd*	Boolean to enable (TRUE) or disable (FALSE) '*** SECRET ***' replacement in parsed parameter list below
out	*piOid*	If this pointer is set and the main table is an overlay the corresponding object identifier is returned
out	*ppLst*	Pointer to the parsed parameter list (NULL = no list provided) in the CLP handle

Returns

signed integer with CLP_OK (0 - nothing parsed) or an error code (CLPERR_xxxxxx (<0)) or the amount of parsed entities (>0)

5.16.2.7 siClpSyntax()

```
int siClpSyntax (
            void * pvHdl,
            const int isSkr,
            const int isMin,
            const int siDep,
            const char * pcPat )
```

Print command line syntax.

The function prints the command line syntax

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*isSkr*	If true the syntax will be printed in structured form
in	*isMin*	If true each element will be prefixed with '!' for required and '?' for optional
in	*siDep*	Depth of next levels to display (0-Nothing, 1-One Level, 2-Two Level, ..., <9-All)
in	*pcPat*	Path (root.input...) to limit syntax to a certain level

Returns

signed integer with CLP_OK(0) or an error code (CLPERR_xxxxxx)

5.16.2.8 pcClpInfo()

```
const char* pcClpInfo (
            void * pvHdl,
            const char * pcPat )
```

Give help message for given path.

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*pcPat*	Path (root.input...) to limit help to a certain level

Returns

string to message or emty message if error

5.16.2.9 siClpHelp()

```
int siClpHelp (
            void * pvHdl,
            const int siDep,
            const char * pcPat,
            const int isAli,
            const int isMan )
```

Print help for command line syntax.
The function prints the help strings for the command line syntax

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*pcPat*	Path (root.input...) to limit help to a certain level
in	*siDep*	Depth of next levels to display (0-Manpage, 1-One Level, 2-Two Level, ..., <9-All)
in	*isAli*	Print also aliases (if FALSE help don't show aliases)
in	*isMan*	Print manpage if no further arguments are available

Returns

signed integer with CLP_OK(0) or an error code (CLPERR_xxxxxx)

5.16.2.10 siClpDocu()

```
int siClpDocu (
            void * pvHdl,
            FILE * pfDoc,
            const char * pcPat,
            const char * pcNum,
            const char * pcKnd,
            const int isCmd,
            const int isDep,
            const int isMan,
            const int isAnc,
            const int isNbr,
            const int isIdt,
            const int isPat,
            const unsigned int uiLev )
```

Generate documentation for command line syntax.
The function generates the documentation for a whole command or if a path is assigned a part of the command. The format will be ASCIIDOC. If one of the arguments has a manual page (detailed description) a headline with numbering dependent from the keyword is generated. After a generated synopsis including the help message, the path, the type and the syntax the detailed description (pcMan) is printed. If one of theses arguments has no manual page then a bullet list with the keyword, type, syntax and help message will be built.
The headline will be 'number ARGUMENT keyword'. For arguments the headline level 3 '~' is used. The same is valid for constant definitions, but in this case the headline level 4 '^' and the key word 'CONSTANT' in Headline or 'SELECTIONS' in bullet list is used.
Numbering of headlines for doc type book can be enabled or disabled. If a level given (>0) then the headlines are prefixed with this amount of '=' for arguments and with one '=' more for constants or selections. A valid minimum level would be 3 and the maximum should not greater then 5.
The manual page for the command is displayed with headline level 2 '-', if no manual page is available the message below is shown:

```
'No detailed description available for this command.'
```

The same is valid for objects and overlays, which means that at a minimum each command, overlay and object needs a detailed description of all normal arguments can be printed as bullet list.
There will be one level of headlines left for the CleExecute, where the program itself, the commands, built-in functions and the appendix are separated.
If only one level of depth is used for printing, then doc type book (isMan==FALSE) or manual page (isMan=TRUE) can be selected. With doc type book the corresponding section of the user manual is generated, with manual page option the asciidoc MANPAGE format is made.

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*pfDoc*	File handle used to write the documentation in ASCIIDOC format
in	*pcPat*	Path (root.input...) to limit documentation certain level
in	*pcNum*	Leading number for table of contents ("1.2." used or not used depend on isNbr, NULL pointer cause an error)
in	*pcKnd*	Qualifier for command/otherclp head line (Recommended: "COMMAND" or "STRING" NULL pointer cause an error)
in	*isCmd*	If TRUE for command and FALSE for other CLP strings (required for anchor generation).
in	*isDep*	If TRUE then all deeper parts are printed if FALSE then not.
in	*isMan*	If TRUE then doc type MANPAGE will be generated else doc type book will be used. (isMan==TRUE results automatically in isDep==FALSE)
in	*isAnc*	Boolean to enable write of generated anchors for each command (only for doc type book)
in	*isNbr*	Boolean to enable header numbering for generated documentation (only for doc type book)
in	*isIdt*	Boolean to enable printing of generated index terms (only for doc type book)
in	*isPat*	Boolean to enable printing of path as part of the synopsis (only for doc type book)

Parameters

in	*uiLev*	If > 0 then headlines are written with this amount of '=' in front instead of underlining (only for doc type book)

Returns

signed integer with CLP_OK(0) or an error code (CLPERR_xxxxxx)

5.16.2.11 siClpPrint()

```
int siClpPrint (
            void * pvHdl,
            const char * pcFil,
            const char * pcNum,
            const char * pcKnd,
            const int isCmd,
            const int isDep,
            const int isAnc,
            const int isNbr,
            const int isShl,
            const int isIdt,
            const int isPat,
            const unsigned int uiLev,
            const int siPs1,
            const int siPs2,
            const int siPr3,
            void * pvPrn,
            TfClpPrintPage * pfPrn )
```

Generate documentation using a callback function.
This function works like siClpDocu, but it gives each page to a callback function and don't print it to a certain file.

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*pcFil*	Prefix for file name building
in	*pcNum*	Leading number for table of contents ("1.2." used or not used depend on isNbr, NULL pointer cause an error)
in	*pcKnd*	Qualifier for command/otherclp head line (Recommended: "COMMAND" or "STRING" NULL pointer cause an error)
in	*isCmd*	If TRUE for command and FALSE for other CLP strings (required for anchor generation).
in	*isDep*	If TRUE then all deeper parts are printed if FALSE then not.
in	*isAnc*	Boolean to enable write of generated anchors for each command (only for doc type book)
in	*isNbr*	Boolean to enable header numbering for generated documentation (only for doc type book)
in	*isShl*	Boolean to enable short headline without type specification (only for doc type book)
in	*isIdt*	Boolean to enable printing of generated index terms (only for doc type book)
in	*isPat*	Boolean to enable printing of path as part of the synopsis (only for doc type book)
in	*uiLev*	If > 0 then headlines are written with this amount of '=' in front instead of underlining (only for doc type book)
in	*siPs1*	Character to separate parts to build filename outside command path (only for doc type book)

Parameters

in	*siPs2*	Character to separate parts to build filename inside command path (only for doc type book)
in	*siPr3*	Character to replace non alpha-numerical characters in file names (only for doc type book)
in	*pvPrn*	Handle for the print callback function (created with TfCleOpenPrint (opnHtmlDoc))
in,out	*pfPrn*	Pointer to the callback function TfClpPrintPage (prnHtmlDoc)

Returns

signed integer with CLP_OK(0) or an error code (CLPERR_xxxxxx)

5.16.2.12 siClpProperties()

```
int siClpProperties (
            void * pvHdl,
            const int siMtd,
            const int siDep,
            const char * pcPat,
            FILE * pfOut )
```

Generate properties.
The function produces a property list with the current default values

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*siMtd*	Method for property printing (0-ALL, 1-Defined, 2-All but not defined as comment)
in	*siDep*	Depth of next levels to print (1-One Level, 2-Two Level, ..., <9-All)
in	*pcPat*	Path (root.input...) to limit the amount of properties
in	*pfOut*	File pointer to write the property list (if NULL then pfHlp of FLAMCLP is used)

Returns

signed integer with CLP_OK(0) or an error code (CLPERR_xxxxxx)

5.16.2.13 siClpLexemes()

```
int siClpLexemes (
            void * pvHdl,
            FILE * pfOut )
```

Print the lexems of the command line compiler.
The function prints the regular expressions of the command line compiler

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*pfOut*	Pointer to the file descriptor used to print the regular expressions

Returns

signed integer with CLP_OK(0) or an error code (CLPERR_xxxxxx)

5.16.2.14 siClpGrammar()

```
int siClpGrammar (
            void * pvHdl,
            FILE * pfOut )
```

Print the grammar of the command line compiler.
The function prints the context free grammar of the command line compiler

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*pfOut*	Pointer to the file descriptor used to print the grammar

Returns

signed integer with CLP_OK(0) or an error code (CLPERR_xxxxxx)

5.16.2.15 vdClpClose()

```
void vdClpClose (
            void * pvHdl,
            const int siMtd )
```

Close the command line parser.
The function releases the allocated resources in the handle. If dynamic allocation of data fields used in the CLP structure, you can close the CLP handle except the list of dynamic allocated pointer in the CLP structure. If the siMtd is set to CLPCLS_MTD_KEP the CLP handle is still open and can later be used to free the remaining pointers of the CLP structure. If the method EXC (except) used, then the CLP handle is closed and the application must free the allocated memory in the CLP structure. This is useful if you need the CLP no longer but the CLP data structure must be valid. You can later call the vdClpClose function again to release the dynamic allocated data fields of the CLP structure with method keep.

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*siMtd*	Define the close method (EXC/KEP/ALL)

5.16.2.16 pvClpAlloc()

```
void* pvClpAlloc (
            void * pvHdl,
            void * pvPtr,
            int siSiz,
            int * piInd )
```

Allocate memory in CLP structure.
This function allocates memory for the CLP structure and can be used to extend the structure where dynamic array or strings must be extended. If the pointer (pvPtr) not NULL and not known by CLP the function pvClpAlloc is called like pvPtr=NULL. This mechanism can be used to use a pointer to a literal or a static variable in the initialization phase

Parameters

in	*pvHdl*	Pointer to the corresponding handle created with *pvClpOpen*
in	*pvPtr*	Pointer of the dynamic allocated area or NULL if it is a new one
in	*siSiz*	Required size for the dynamic area
in,out	*piInd*	Pointer to the index of the memory area or NULL (improves performance)

Returns

Pointer to allocated and initialized memory or NULL if error

5.16.2.17 pcClpError()

```
char* pcClpError (
            int siErr )
```

Provides error message.
The function provides a error message for the corresponding error code

Parameters

in	*siErr*	Error code from parser

Chapter 6

Data Structure Documentation

6.1 CleCommand Struct Reference

CLE structure for command table.
`#include <CLEDEF.h>`

Data Fields

- const char * pcKyw

  Pointer to the key word for this command (:alpha:[:alnum:|'_']).*

- const TsClpArgument * psTab

  Pointer to the main argument table for this command (defines the semantic for the parser).

- void * pvClp

  Pointer to the corresponding argument structure (filled up by the parser).

- void * pvPar

  Pointer to the corresponding parameter structure (filled up by the mapping function).

- int * piOid

  Pointer to the object identifier for overlay commands (filled up by the parser, see siCleExecute()).

- TfIni * pfIni

  Pointer to the initialization function (initialize the argument structure in front of parsing).

- TfMap * pfMap

  Pointer to the mapping function (transfers the argument structure to the parameter structure).

- TfRun * pfRun

  Pointer to the executed function (use the mapped parameter structure to execute the command. (for logging the function name, original command line and parsed argument list and other values are also provided)).

- TfFin * pfFin

  Pointer to the finish function for cleanup (free memory, close files in parameter structure).

- int siFlg

  Flag to indicate a hidden (==0) or visible (!=0) command. For correct numbering, put hidden commands to the end of the table.

- const char * pcMan

  Pointer to a null-terminated string for a detailed description of this command (in ASCIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs plus .OPTIONS and/or.EXAMPLES). It is recommended to use a header file with a define for this long string. "&{OWN}" and "&{PGM}" are replaced with the current owner and program name. The resulting text is converted on EBCDIC systems).

- const char * pcHlp

  String for a short context sensitive help to this command (converted on EBCDIC systems).

6.1.1 Detailed Description

CLE structure for command table.
The command table defines all the commands which could be executed by CLE. To simplify the definition of command tables it is recommended to use the CLPTAB macros.

6.1.2 Field Documentation

6.1.2.1 pcKyw

`const char* CleCommand::pcKyw`
Pointer to the key word for this command (:alpha:[:alnum:|'_']*).

6.1.2.2 psTab

`const TsClpArgument* CleCommand::psTab`
Pointer to the main argument table for this command (defines the semantic for the parser).

6.1.2.3 pvClp

`void* CleCommand::pvClp`
Pointer to the corresponding argument structure (filled up by the parser).

6.1.2.4 pvPar

`void* CleCommand::pvPar`
Pointer to the corresponding parameter structure (filled up by the mapping function).

6.1.2.5 piOid

`int* CleCommand::piOid`
Pointer to the object identifier for overlay commands (filled up by the parser, see siCleExecute()).

6.1.2.6 pfIni

`TfIni* CleCommand::pfIni`
Pointer to the initialization function (initialize the argument structure in front of parsing).

6.1.2.7 pfMap

`TfMap* CleCommand::pfMap`
Pointer to the mapping function (transfers the argument structure to the parameter structure).

6.1.2.8 pfRun

`TfRun* CleCommand::pfRun`
Pointer to the executed function (use the mapped parameter structure to execute the command. (for logging the function name, original command line and parsed argument list and other values are also provided)).

6.1.2.9 pfFin

`TfFin* CleCommand::pfFin`
Pointer to the finish function for cleanup (free memory, close files in parameter structure).

6.1.2.10 siFlg

`int CleCommand::siFlg`
Flag to indicate a hidden (==0) or visible (!=0) command. For correct numbering, put hidden commands to the end of the table.

6.1.2.11 pcMan

`const char* CleCommand::pcMan`
Pointer to a null-terminated string for a detailed description of this command (in ASCIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs plus .OPTIONS and/or.EXAMPLES). It is recommended to use a header file with a define for this long string. "&{OWN}" and "&{PGM}" are replaced with the current owner and program name. The resulting text is converted on EBCDIC systems).

6.1.2.12 pcHlp

`const char* CleCommand::pcHlp`
String for a short context sensitive help to this command (converted on EBCDIC systems).
The documentation for this struct was generated from the following file:

- CLEDEF.h

6.2 CleDoc Struct Reference

CLE Structure for documentation table.
`#include <CLEDEF.h>`

Data Fields

- unsigned int uiTyp

  One of the documentation types above.
- unsigned int uiLev

  The level of the chapter in the document (cover page is 1 all other chapter > 1).
- const char * pcNum

  String for numbering or NULL for no number prefix.
- const char * pcKyw

  Optional ASCIIDOC key word (printed in front of headline in single square brackets).
- const char * pcAnc

  Optional anchor for this chapter (printed in front of headline in double square brackets).
- const char * pcHdl

  Headline for this chapter.
- const char * pcMan

  Optional or required manual page with the content of this chapter).
- const char * pcIdt

  Optional new line separated list of index term for this chapter (printed at the end (indexterm:[])).

6.2.1 Detailed Description

CLE Structure for documentation table.
This structure is used to build a table with the macros CLEDOC_OPN, CLETAB_DOC and CLEDOC_CLS. This table must be provided to the function siCleExecute for documentation generation using the built-in functions GENDOCU and HTMLDOC.

6.2.2 Field Documentation

6.2.2.1 uiTyp

`unsigned int CleDoc::uiTyp`
One of the documentation types above.

6.2.2.2 uiLev

`unsigned int CleDoc::uiLev`
The level of the chapter in the document (cover page is 1 all other chapter > 1).

6.2.2.3 pcNum

`const char* CleDoc::pcNum`
String for numbering or NULL for no number prefix.

6.2.2.4 pcKyw

`const char* CleDoc::pcKyw`
Optional ASCIIDOC key word (printed in front of headline in single square brackets).

6.2.2.5 pcAnc

`const char* CleDoc::pcAnc`
Optional anchor for this chapter (printed in front of headline in double square brackets).

6.2.2.6 pcHdl

`const char* CleDoc::pcHdl`
Headline for this chapter.

6.2.2.7 pcMan

`const char* CleDoc::pcMan`
Optional or required manual page with the content of this chapter).

6.2.2.8 pcIdt

`const char* CleDoc::pcIdt`
Optional new line separated list of index term for this chapter (printed at the end (indexterm:[])).
The documentation for this struct was generated from the following file:

- CLEDEF.h

6.3 CleOtherClp Struct Reference

CLE table structure for other CLP strings.
`#include <CLEDEF.h>`

Data Fields

- const char * pcRot

  Pointer to the program/root key word for this string (:alpha:[:alnum:|'_']).*
- const char * pcKyw

  Pointer to the key word for this string (:alpha:[:alnum:|'_']).*
- const TsClpArgument * psTab

  Pointer to the main argument table for this command (defines the semantic for the parser).
- const char * pcMan

  Pointer to a null-terminated string for a detailed description for this CLP string (in ASCIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs plus .OPTIONS and/or.EXAMPLES) It is recommended to use a header file with a define for this long string. "&{OWN}" and "&{PGM}" are replaced with the current owner and program name. The resulting text is converted on EBCDIC systems).
- const char * pcHlp

  String for a short context sensitive help for this CLP string (converted on EBCDIC systems).
- const int isOvl

  True if provided table must be interpreted as overlay else as object.

6.3.1 Detailed Description

CLE table structure for other CLP strings.
This structure is used to define a table of CLP strings in order to add as appendix to the generated documentation.

6.3.2 Field Documentation

6.3.2.1 pcRot

`const char* CleOtherClp::pcRot`
Pointer to the program/root key word for this string (:alpha:[:alnum:|'_']*).

6.3.2.2 pcKyw

`const char* CleOtherClp::pcKyw`
Pointer to the key word for this string (:alpha:[:alnum:|'_']*).

6.3.2.3 psTab

`const TsClpArgument* CleOtherClp::psTab`
Pointer to the main argument table for this command (defines the semantic for the parser).

6.3.2.4 pcMan

`const char* CleOtherClp::pcMan`
Pointer to a null-terminated string for a detailed description for this CLP string (in ASCIDOC format, content behind .DESCRIPTION, mainly simply some paragraphs plus .OPTIONS and/or.EXAMPLES) It is recommended to use a header file with a define for this long string. "&{OWN}" and "&{PGM}" are replaced with the current owner and program name. The resulting text is converted on EBCDIC systems).

6.3.2.5 pcHlp

`const char* CleOtherClp::pcHlp`
String for a short context sensitive help for this CLP string (converted on EBCDIC systems).

6.3.2.6 isOvl

`const int CleOtherClp::isOvl`
True if provided table must be interpreted as overlay else as object.
The documentation for this struct was generated from the following file:

- CLEDEF.h

6.4 ClpArgument Struct Reference

Table structure for arguments.
`#include <CLPDEF.h>`

Data Fields

- int siTyp

  Data type Type of this parameter (CLPTYP_xxxxxx). The type will be displayed in context sensitive help messages (TYPE: type_name).
- const char * pcKyw

  Keyword (Parameter name) Pointer to a null-terminated key word for this parameter (:alpha:[:alnum:|'_']).*
- const char * pcAli

  Alias (other name for a existing parameter) Pointer to another key word to define an alias (:alpha:[:alnum:|'_']).*
- int siMin

  Minimum amount of entries for this argument (0-optional n-required).
- int siMax

  Maximum amount of entries for this argument (1-scalar n-array (n=0 unlimited array, n>1 limited array)).
- int siSiz

  Data size If fixed size type (switch, number, float, object, overlay) then size of this type else (string) available size in memory. String type can be defined as FIX with CLPFLG_FIX but this requires a typedef for this string size. For dynamic strings an initial size for the first memory allocation can be defined.
- int siOfs

  Data Offset Offset of an argument in a structure used for address calculation (the offset of(t,m) instruction is used by the macros for it).
- int siOid

  Object identifier Unique integer value representing the argument (object identifier, used in overlays or for switches).
- unsigned int uiFlg

  Control Flag Flag value which can be assigned with CLPFLG_SEL/CON/FIX/CNT/SEN/ELN/TLN/OID/ALI to define different characteristics.
- struct ClpArgument * psTab

  Table for next level Pointer to another parameter table for CLPTYP_OBJECT and CLPTYP_OVRLAY describing these structures, for CLPTYP_NUMBER, CLPTYP_FLOATN or CLPTYP_STRING to define selections (constant definitions)
- const char * pcDft

  Default value.
- const char * pcMan

  Manual page.
- const char * pcHlp

  Help message.

6.4.1 Detailed Description

Table structure for arguments.
To simplify the definition of the corresponding data structures and argument tables it is recommended to use the CLPARGTAB macros defined in CLPMAC.h or for constant definitions the CLPCONTAB macros below. With the CLPMAC.h you can generate the tables or the corresponding data structures, depending if DEFINE_STRUCT defined or not.
Example:
First you must define the table:

```
#define CLPINTFMTRED_TABLE\
 CLPARGTAB_SKALAR("BIN"          , stBin , TsClpIntRedBin , 0, 1, CLPTYP_OBJECT, CLPFLG_NON,
       INTCNV_FORMAT_BIN , asClpIntRedBin , NULL, MAN_INTRED_BIN, "Integer in binary format (two's
       complement)")\
 CLPARGTAB_SKALAR("BCD"          , stBcd , TsClpIntRedBcd , 0, 1, CLPTYP_OBJECT, CLPFLG_NON,
       INTCNV_FORMAT_BCD , asClpIntRedBcd , NULL, MAN_INTRED_BCD, "Binary coded decimal (BCD) number")\
 CLPARGTAB_SKALAR("STR"          , stStr , TsClpIntRedStr , 0, 1, CLPTYP_OBJECT, CLPFLG_NON,
       INTCNV_FORMAT_STR , asClpIntRedStr , NULL, MAN_INTRED_STR, "String representation of an integer")\
 CLPARGTAB_SKALAR("ENUM"         , stEnum, TsClpIntRedSel , 0, 1, CLPTYP_OBJECT, CLPFLG_NON,
       INTCNV_FORMAT_ENUM, asClpIntRedEnum, NULL, MAN_INTRED_ENUM,"Maps enumeration to an integer")\
 CLPARGTAB_CLS
```

Then you can use this table to define your structure or union, in our case we create a union:

```
#define DEFINE_STRUCT
#include "CLPMAC.h"
typedef union ClpIntFmtRed{
   CLPINTFMTRED_TABLE
}TuClpIntFmtRed;
```

And you use the same define to allocate the corresponding CLP table:

```
#undef DEFINE_STRUCT
#include "CLPMAC.h"
#undef  STRUCT_NAME
#define STRUCT_NAME TuClpIntFmtRed
TsClpArgument       asClpIntFmtRed[] = {
   CLPINTFMTRED_TABLE
};
```

6.4.2 Field Documentation

6.4.2.1 siTyp

`int ClpArgument::siTyp`
Data type Type of this parameter (CLPTYP_xxxxxx). The type will be displayed in context sensitive help messages (TYPE: type_name).

6.4.2.2 pcKyw

`const char* ClpArgument::pcKyw`
Keyword (Parameter name) Pointer to a null-terminated key word for this parameter (:alpha:[:alnum:|'_']*).

6.4.2.3 pcAli

`const char* ClpArgument::pcAli`
Alias (other name for a existing parameter) Pointer to another key word to define an alias (:alpha:[:alnum:|'_']*).

6.4.2.4 siMin

`int ClpArgument::siMin`
Minimum amount of entries for this argument (0-optional n-required).

6.4.2.5 siMax

`int ClpArgument::siMax`
Maximum amount of entries for this argument (1-scalar n-array (n=0 unlimited array, n>1 limited array)).

6.4.2.6 siSiz

`int ClpArgument::siSiz`
Data size If fixed size type (switch, number, float, object, overlay) then size of this type else (string) available size in memory. String type can be defined as FIX with CLPFLG_FIX but this requires a typedef for this string size. For dynamic strings an initial size for the first memory allocation can be defined.

6.4.2.7 siOfs

`int ClpArgument::siOfs`
Data Offset Offset of an argument in a structure used for address calculation (the offset of(t,m) instruction is used by the macros for it).

6.4.2.8 siOid

`int ClpArgument::siOid`
Object identifier Unique integer value representing the argument (object identifier, used in overlays or for switches).

6.4.2.9 uiFlg

`unsigned int ClpArgument::uiFlg`
Control Flag Flag value which can be assigned with CLPFLG_SEL/CON/FIX/CNT/SEN/ELN/TLN/OID/ALI to define different characteristics.

6.4.2.10 psTab

`struct ClpArgument* ClpArgument::psTab`
Table for next level Pointer to another parameter table for CLPTYP_OBJECT and CLPTYP_OVRLAY describing these structures, for CLPTYP_NUMBER, CLPTYP_FLOATN or CLPTYP_STRING to define selections (constant definitions)

6.4.2.11 pcDft

`const char* ClpArgument::pcDft`
Default value.
Pointer to a zero-terminated string to define the default values assigned if no argument was defined. If this pointer is NULL or empty ("") then no initialization is done.

- for switches a number literal or the special keywords ON/OFF can be defined
- for numbers a number literal or a key word for a constant definition can be defined
- for floats a floating point number literal or a key word for a constant definition can be defined
- for strings a string literal or a key word for a constant definition can be defined
- for objects the special keyword INIT must be defined to initialize the object
- for overlays the keyword of the assigning object must be defined to initialize the overlay

For arrays of these types a list of the corresponding values (literals or key words) can be defined The default values are displayed in context sensitive help messages (PROPERTY: [value_list]) This value can be override by corresponding environment variable or property definition.

6.4.2.12 pcMan

`const char* ClpArgument::pcMan`
Manual page.
Pointer to a zero-terminated string for a detailed description of this argument (in ASCIIDOC format, content behind. DESCRIPTION, mainly simply some paragraphs). Can be a NULL pointer or empty string for constant definition or simple arguments. It is recommended to use a header file with a define for this long string (required for objects and overlays). All occurrences of "&{OWN}" or "&{PGM}" (that all their case variations) are replaced with the current owner or program name, respectively. All other content between "&{" and "}" is ignored (comment). The resulting text is converted on EBCDIC systems).

6.4.2.13 pcHlp

`const char* ClpArgument::pcHlp`
Help message.
Pointer to a zero-terminated string for context sensitive help to this argument. Also used as headline in documentation generation. For this only alnum, blank, dot, comma, hyphen and parenthesis are used. At every other separator the headline will be cut, meaning it is possible to have more help information than head line. (converted on EBCDIC systems).
The documentation for this struct was generated from the following file:

- CLPDEF.h

6.5 ClpError Struct Reference

Defines a structure with error information.
`#include <CLPDEF.h>`

Data Fields

- const char ** ppMsg

  Points to the pointer of a zero-terminated string containing the current error message.
- const char ** ppSrc

  If a parameter file assigned and cause of the error pcSrc points to this file name.
- const int * piRow

  *Points to an integer containing the current row for the error in *pcSrc*e.*
- const int * piCol

  Points to an integer containing the current column for the error in pcSrc.

6.5.1 Detailed Description

Defines a structure with error information.
A pointer to this structure can be provided at siClpOpen() to have access to the error information managed in the CLP handle.
The pointers are set by CLP and valid until CLP is closed.

6.5.2 Field Documentation

6.5.2.1 ppMsg

`const char** ClpError::ppMsg`
Points to the pointer of a zero-terminated string containing the current error message.

6.5.2.2 ppSrc

`const char** ClpError::ppSrc`
If a parameter file assigned and cause of the error *pcSrc* points to this file name.
Points to the pointer of a zero-terminated string containing the current source. The initial source can be defined for command line or property file parsing. If the initial source is not defined the constant definitions below are used:

- for command line parsing ":command line:" see CLPSRC_CMD
- for property string parsing ":property list:" see CLPSRC_PRO

6.5.2.3 piRow

`const int* ClpError::piRow`
Points to an integer containing the current row for the error in ∗pcSrc∗e.

6.5.2.4 piCol

`const int* ClpError::piCol`
Points to an integer containing the current column for the error in *pcSrc*.
The documentation for this struct was generated from the following file:

- CLPDEF.h

6.6 DiaChr Struct Reference

`#include <CLEPUTL.h>`

Data Fields

- char exc [4]

  Reset list structure of environment variables
  .
- char hsh [4]
- char dlr [4]
- char ats [4]
- char sbo [4]
- char bsl [4]
- char sbc [4]
- char crt [4]
- char grv [4]
- char cbo [4]
- char vbr [4]
- char cbc [4]
- char tld [4]
- char svb [4]
- char sbs [4]
- char idt [4]

6.6.1 Field Documentation

6.6.1.1 exc

```
char DiaChr::exc[4]
```

Reset list structure of environment variables
.

6.6.1.2 hsh

```
char DiaChr::hsh[4]
```

6.6.1.3 dlr

```
char DiaChr::dlr[4]
```

6.6.1.4 ats

```
char DiaChr::ats[4]
```

6.6.1.5 sbo

```
char DiaChr::sbo[4]
```

6.6.1.6 bsl

```
char DiaChr::bsl[4]
```

6.6.1.7 sbc

```
char DiaChr::sbc[4]
```

6.6.1.8 crt

```
char DiaChr::crt[4]
```

6.6.1.9 grv

```
char DiaChr::grv[4]
```

6.6.1.10 cbo

```
char DiaChr::cbo[4]
```

6.6.1.11 vbr

```
char DiaChr::vbr[4]
```

6.6.1.12 cbc

```
char DiaChr::cbc[4]
```

6.6.1.13 tld

```
char DiaChr::tld[4]
```

6.6.1.14 svb

```
char DiaChr::svb[4]
```

6.6.1.15 sbs

```
char DiaChr::sbs[4]
```

6.6.1.16 idt

```
char DiaChr::idt[4]
```

The documentation for this struct was generated from the following file:

- CLEPUTL.h

6.7 EnVarList Struct Reference

```
#include <CLEPUTL.h>
```

Data Fields

- char * pcName

  Environment variable list
  .
- char * pcValue
- struct EnVarList * psNext

6.7.1 Field Documentation

6.7.1.1 pcName

```
char* EnVarList::pcName
```

Environment variable list
.

6.7.1.2 pcValue

```
char* EnVarList::pcValue
```

6.7.1.3 psNext

```
struct EnVarList* EnVarList::psNext
```

The documentation for this struct was generated from the following file:

- CLEPUTL.h

Chapter 7

File Documentation

7.1 CLEDEF.h File Reference

Definitions for **C**ommand **L**ine **E**xecution.

```
#include "stdio.h"
#include "CLPDEF.h"
```

Data Structures

- struct CleDoc

  CLE Structure for documentation table.
- struct CleCommand

  CLE structure for command table.
- struct CleOtherClp

  CLE table structure for other CLP strings.

Macros

- #define CLE_DOCTYP_COVER 1U

  Cover page (level must be 1).
- #define CLE_DOCTYP_CHAPTER 2U

  A chapter (level must $>$ 1 and $<$ 6).
- #define CLE_DOCTYP_PROGRAM 10U

  The main program chapter (like chapter but level must $<$ 5).
- #define CLE_DOCTYP_PGMSYNOPSIS 11U

  The program synopsis.
- #define CLE_DOCTYP_PGMSYNTAX 12U

  The program syntax.
- #define CLE_DOCTYP_PGMHELP 13U

  The program help.
- #define CLE_DOCTYP_COMMANDS 20U

  The commands part.
- #define CLE_DOCTYP_OTHERCLP 21U

  Other CLP strings.
- #define CLE_DOCTYP_BUILTIN 22U

  The built-in function section.
- #define CLE_DOCTYP_LEXEMES 30U

  The appendix which prints the lexemes.
- #define CLE_DOCTYP_GRAMMAR 31U

The appendix which prints the grammar.

- #define CLE_DOCTYP_VERSION 32U

  The appendix which prints the version (pcVsn must be given).
- #define CLE_DOCTYP_ABOUT 33U

  The appendix which prints the about (pcAbo must be given).
- #define CLE_DOCTYP_PROPREMAIN 41U

  The appendix which prints the remaining parameter documentation.
- #define CLE_DOCTYP_PROPDEFAULTS 42U

  The appendix which prints the default parameter documentation.
- #define CLE_DOCTYP_SPECIALCODES 51U

  The appendix which prints the special condition codes.
- #define CLE_DOCTYP_REASONCODES 52U

  The appendix which prints the reason codes (pfMsg must be provided).
- #define CLE_DOCKYW_PREFACE "preface"

  Mark level 2 chapter as preface.
- #define CLE_DOCKYW_APPENDIX "appendix"

  Mark level 2 chapter as appendix.
- #define CLE_DOCKYW_GLOSSARY "glossary"

  Mark level 2 chapter as glossary.
- #define CLE_DOCKYW_COLOPHON "colophon"

  Mark level 2 chapter as colophon.
- #define CLE_ANCHOR_BUILTIN_FUNCTIONS "CLEP.BUILTIN.FUNCTIONS"

  Chapter built-in functions.
- #define CLE_ANCHOR_APPENDIX_ABOUT "CLEP.APPENDIX.ABOUT"

  Appendix About.
- #define CLE_ANCHOR_APPENDIX_VERSION "CLEP.APPENDIX.VERSION"

  Appendix Version.
- #define CLE_ANCHOR_APPENDIX_LEXEMES "CLEP.APPENDIX.LEXEMES"

  Appendix Lexemes.
- #define CLE_ANCHOR_APPENDIX_GRAMMAR "CLEP.APPENDIX.GRAMMAR"

  Appendix Grammar.
- #define CLE_ANCHOR_APPENDIX_RETURNCODES "CLEP.APPENDIX.RETURNCODES"

  Appendix Return codes.
- #define CLE_ANCHOR_APPENDIX_REASONCODES "CLEP.APPENDIX.REASONCODES"

  Appendix Reason codes.
- #define CLE_ANCHOR_APPENDIX_PROPERTIES "CLEP.APPENDIX.PROPERTIES"

  Appendix Properties.
- #define CLEDOC_OPN(name) TsCleDoc name[ ]

  Starts the documentation generation table overview.
- #define CLETAB_DOC(typ, lev, num, kyw, anc, hdl, man, idt) {(typ),(lev),(num),(kyw),(anc),(hdl),(man),(idt)},

  Starts the documentation generation table.
- #define CLEDOC_CLS { 0 , 0 , NULL, NULL, NULL, NULL, NULL, NULL}

  Ends a table with constant definitions.
- #define CLECMD_OPN(name) TsCleCommand name[ ]

  Starts a table with command definitions.
- #define CLETAB_CMD(kyw, tab, clp, par, oid, ini, map, run, fin, flg, man, hlp) {(kyw),(tab),(clp),(par),(oid),(ini),(map),(run),(fin),(fl

  Defines a command with the command line keyword kyw.
- #define CLECMD_CLS { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, 0 , NULL, NULL}

  Ends a table with constant definitions.
- #define CLEOTH_OPN(name) TsCleOtherClp name[ ]

  Starts a table with other CLP strings.

- #define CLETAB_OTH(rot, kyw, tab, man, hlp, ovl) {(rot),(kyw),(tab),(man),(hlp),(ovl)},

  Defines a appendix for the object or overlay cmd of the root rot with the headline of hdl for a certain other CLP string. .
- #define CLEOTH_CLS { NULL, NULL, NULL, NULL, NULL, 0}

  Ends a table with other CLP strings.

Typedefs

- typedef struct CleDoc TsCleDoc

  CLE Structure for documentation table.
- typedef void ∗() TfCleOpenPrint(FILE ∗pfOut, FILE ∗pfErr, const char ∗pcPat, const char ∗pcOwn, const char ∗pcPgm, const char ∗pcBld, int ∗piHdr, int ∗piAnc, int ∗piIdt, int ∗piPat, int ∗psPs1, int ∗piPs2, int ∗piPr3)

  Function 'opnHtmlDoc' of library 'libhtmldoc' called if built-in function HTMLDOC used.
- typedef int() TfCleClosePrint(void ∗pvHdl)

  Function 'clsHtmlDoc' of library 'libhtmldoc' called if built-in function HTMLDOC used.
- typedef int() TfIni(void ∗pvHdl, FILE ∗pfOut, FILE ∗pfTrc, void ∗pvGbl, const char ∗pcOwn, const char ∗pc↩ Pgm, void ∗pvClp)

  Type definition for initialization FLAMCLE command structure.
- typedef int() TfMap(void ∗pvHdl, FILE ∗pfOut, FILE ∗pfTrc, void ∗pvGbl, int ∗piOid, void ∗pvClp, void ∗pvPar)

  Type definition for mapping parsed values from the FLAMCLP structure.
- typedef int() TfRun(void ∗pvHdl, FILE ∗pfOut, FILE ∗pfTrc, void ∗pvGbl, const char ∗pcOwn, const char ∗pc↩ Pgm, const char ∗pcVsn, const char ∗pcAbo, const char ∗pcLic, const char ∗pcFkt, const char ∗pcCmd, const char ∗pcLst, const void ∗pvPar, int ∗piWrn, int ∗piScc)

  Type definition for CLE run function.
- typedef int() TfFin(FILE ∗pfOut, FILE ∗pfTrc, void ∗pvGbl, void ∗pvPar)

  Type definition for the CLE fin function.
- typedef const char ∗() TfMsg(const int siRsn)

  Type definition for the CLE message function.
- typedef struct CleCommand TsCleCommand

  CLE structure for command table.
- typedef struct CleOtherClp TsCleOtherClp

  CLE table structure for other CLP strings.

7.1.1 Detailed Description

Definitions for **C**ommand **L**ine **E**xecution.

Author

limes datentechnik gmbh

Date

27.12.2019

Copyright

2. Altered source versions must be plainly marked as such, and must not be misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

If you need professional services or support for this library please contact support@flam.de.

7.2 CLEPUTL.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
```

Data Structures

- struct EnVarList
- struct DiaChr

Macros

- #define CLEP_DEFAULT_CCSID_ASCII 819
- #define CLEP_DEFAULT_CCSID_EBCDIC 1047
- #define SAFE_FREE(x) do { if ((x) != NULL) {free((void*)(x)); (x)=NULL;} } while(0)
- #define GETENV(name) getenv((name))
- #define SETENV(name, value) setenv((name), (value), 1)
- #define UNSETENV(name) unsetenv((name))
- #define isStr(c) (isprint(c) || (c)==C_TLD || (c)==C_DLR || (c)==C_ATS || (c)==C_BSL || (c)==C_CRT || (c)==C_EXC)
- #define isKyw(c) (isalnum(c) || (c)=='_')
- #define isCon(c) (isKyw(c) || (c)=='-' || (c)=='/')
- #define ISDDNAME(p) (strlen(p)>3 && toupper((p)[0])=='D' && toupper((p)[1])=='D' && (p)[2]==':')
- #define ISPATHNAME(p) (strchr((p),'/')!=NULL)
- #define ISDSNAME(p) (strlen(p)>2 && toupper((p)[0])=='/' && toupper((p)[1])=='/')
- #define ISGDGMBR(m) ((m)[0]=='0' || (m)[0]=='+' || (m)[0]=='-')
- #define ISDDN(c) (isalnum(c) || (c)==C_DLR || (c)==C_HSH || (c)==C_ATS)
- #define fopen_tmp() tmpfile()
- #define fclose_tmp(fp) fclose((fp))
- #define remove_hfq(n) remove(n)
- #define CLERTC_OK 0
- #define CLERTC_INF 1
- #define CLERTC_FIN 2
- #define CLERTC_WRN 4
- #define CLERTC_RUN 8
- #define CLERTC_MAP 12
- #define CLERTC_SYN 16
- #define CLERTC_CMD 20
- #define CLERTC_INI 24
- #define CLERTC_CFG 28
- #define CLERTC_TAB 32
- #define CLERTC_SYS 36
- #define CLERTC_ACS 40
- #define CLERTC_ITF 44
- #define CLERTC_MEM 48
- #define CLERTC_FAT 64
- #define CLERTC_MAX 64
- #define strncpy(...) Error: Do not use strncpy! Use strlcpy instead!

- #define CSTIME_BUFSIZ 24
- #define HSH_PBRK "#" /∗nodiac∗/
- #define ATS_PBRK "@" /∗nodiac∗/
- #define C_EXC '!' /∗nodiac∗/
- #define C_HSH '#' /∗nodiac∗/
- #define C_DLR '$' /∗nodiac∗/
- #define C_ATS '@' /∗nodiac∗/
- #define C_SBO '[' /∗nodiac∗/
- #define C_BSL '\\' /∗nodiac∗/
- #define C_SBC ']' /∗nodiac∗/
- #define C_CRT '^' /∗nodiac∗/
- #define C_GRV '`' /∗nodiac∗/
- #define C_CBO '{' /∗nodiac∗/
- #define C_VBR '|' /∗nodiac∗/
- #define C_CBC '}' /∗nodiac∗/
- #define C_TLD '∼' /∗nodiac∗/
- #define S_EXC "!" /∗nodiac∗/
- #define S_HSH "#" /∗nodiac∗/
- #define S_DLR "$" /∗nodiac∗/
- #define S_ATS "@" /∗nodiac∗/
- #define S_SBO "[" /∗nodiac∗/
- #define S_BSL "\\" /∗nodiac∗/
- #define S_SBC "]" /∗nodiac∗/
- #define S_CRT "^" /∗nodiac∗/
- #define S_GRV "`" /∗nodiac∗/
- #define S_CBO "{" /∗nodiac∗/
- #define S_VBR "|" /∗nodiac∗/
- #define S_CBC "}" /∗nodiac∗/
- #define S_TLD "∼" /∗nodiac∗/
- #define S_SVB "=|" /∗nodiac∗/
- #define S_SBS "/\\" /∗nodiac∗/
- #define S_IDT "--|" /∗nodiac∗/
- #define esrprintc srprintc
- #define esnprintf snprintf
- #define esprintf sprintf
- #define efprintf fprintf

Typedefs

- typedef struct EnVarList TsEnVarList
- typedef struct DiaChr TsDiaChr

Functions

- FILE ∗ fopen_hfq (const char ∗name, const char ∗mode)
- FILE ∗ fopen_hfq_nowarn (const char ∗name, const char ∗mode)
- FILE ∗ freopen_hfq (const char ∗name, const char ∗mode, FILE ∗stream)
- long long getFileSize (const char ∗name)
- char ∗ userid (const int size, char ∗buffer)
- char ∗ homedir (const int flag, const int size, char ∗buffer)
- char ∗ duserid (void)
- char ∗ dhomedir (const int flag)
- char ∗ safe_getenv (const char ∗name, char ∗buffer, size_t bufsiz)
- char ∗ unEscape (const char ∗input, char ∗output)
- char ∗ dynUnEscape (const char ∗input)

- int printd (const char *format,...) __PRINTF_CHECK__(1
- int int snprintc (char *buffer, const size_t size, const char *format,...) __PRINTF_CHECK__(3
- int int int srprintc (char **buffer, size_t *size, const size_t expansion, const char *format,...) __PRINTF_C↩ HECK__(4
- int int int int srprintf (char **buffer, size_t *size, const size_t expansion, const char *format,...) __PRINTF↩ _CHECK__(4
- int int int int void fprintm (FILE *file, const char *own, const char *pgm, const char *bld, const char *man, const int cnt)
- int snprintm (char *buffer, size_t size, const char *own, const char *pgm, const char *bld, const char *man, const int cnt)
- const char * prsdstr (const char **hdl, const char *str, int len)
- size_t strlcpy (char *dest, const char *src, size_t n)
- char * getenvar (const char *name, const size_t length, const size_t size, char *string)
- char * mapstr (char *string, int size)
- char * dmapstr (const char *string, int method)
- char * dmapxml (const char *string, int method)
- char * mapfil (char *file, int size)
- char * dmapfil (const char *file, int method)
- char * maplab (char *label, int size, int toUpper)
- char * dmaplab (const char *label, int method)
- char * cpmapfil (char *dest, int size, const char *source)
- char * dcpmapfil (const char *file)
- char * cpmaplab (char *label, int size, const char *templ, const char *values, int toUpper)
- char * dcpmaplab (const char *templ, const char *values, int method)
- unsigned int localccsid (void)
- const char * mapl2c (unsigned isEBCDIC)
- const char * lng2ccsd (const char *pcLang, unsigned isEbcdic)
- const char * mapccsid (const unsigned int uiCcsId)
- unsigned int mapcdstr (const char *p)
- unsigned int bin2hex (const unsigned char *bin, char *hex, const unsigned int len)
- unsigned int hex2bin (const char *hex, unsigned char *bin, const unsigned int len)
- unsigned int chr2asc (const char *chr, char *asc, const unsigned int len)
- unsigned int chr2ebc (const char *chr, char *ebc, const unsigned int len)
- unsigned int asc2chr (const char *asc, char *chr, const unsigned int len)
- void asc_chr (const char *asc, char *chr, const unsigned int len)
- void chr_asc (const char *chr, char *asc, const unsigned int len)
- unsigned int ebc2chr (const char *ebc, char *chr, const unsigned int len)
- void ebc_chr (const char *ebc, char *chr, const unsigned int len)
- void chr_ebc (const char *chr, char *ebc, const unsigned int len)
- int file2str (void *hdl, const char *filename, char **buf, int *bufsize, char *errmsg, const int msgsiz)
- int arry2str (char *array[ ], const int count, const char *separ, const int separLen, char **out, int *outlen)
- int strxcmp (const int ca, const char *s1, const char *s2, const int n, const int c, const int f)
- char * cstime (signed long long t, char *p)
- int loadEnvars (const unsigned int uiLen, const char *pcBuf, FILE *pfOut, FILE *pfErr, TsEnVarList **ppList)
- int readEnvars (const char *pcFil, FILE *pfOut, FILE *pfErr, TsEnVarList **ppList)
- int envarInsert (TsEnVarList **ppList, const char *pcName, const char *pcValue)
- int resetEnvars (TsEnVarList **ppList)
- void init_diachr (TsDiaChr *psDiaChr, const unsigned int uiCcsId)

7.2.1 Macro Definition Documentation

7.2.1.1 CLEP_DEFAULT_CCSID_ASCII

```
#define CLEP_DEFAULT_CCSID_ASCII 819
```

7.2.1.2 CLEP_DEFAULT_CCSID_EBCDIC

```
#define CLEP_DEFAULT_CCSID_EBCDIC 1047
```

7.2.1.3 SAFE_FREE

```
#define SAFE_FREE(
             x ) do { if ((x) != NULL) {free((void*)(x)); (x)=NULL;} } while(0)
```

Free memory space

7.2.1.4 GETENV

```
#define GETENV(
             name ) getenv((name))
```

7.2.1.5 SETENV

```
#define SETENV(
             name,
             value ) setenv((name), (value), 1)
```

7.2.1.6 UNSETENV

```
#define UNSETENV(
             name ) unsetenv((name))
```

7.2.1.7 isStr

```
#define isStr(
             c ) (isprint(c) || (c)==C_TLD || (c)==C_DLR || (c)==C_ATS || (c)==C_BSL || (c)==C_CRT
|| (c)==C_EXC)
```

7.2.1.8 isKyw

```
#define isKyw(
             c ) (isalnum(c) || (c)=='_')
```

7.2.1.9 isCon

```
#define isCon(
             c ) (isKyw(c) || (c)=='-' || (c)=='/')
```

7.2.1.10 ISDDNAME

```
#define ISDDNAME(
             p ) (strlen(p)>3 && toupper((p)[0])=='D' && toupper((p)[1])=='D' && (p)[2]==':')
```

7.2.1.11 ISPATHNAME

```
#define ISPATHNAME(
             p ) (strchr((p),'/')!=NULL)
```

7.2.1.12 ISDSNAME

```
#define ISDSNAME(
             p ) (strlen(p)>2 && toupper((p)[0])=='/' && toupper((p)[1])=='/')
```

7.2.1.13 ISGDGMBR

```
#define ISGDGMBR(
             m ) ((m)[0]=='0' || (m)[0]=='+' || (m)[0]=='-')
```

7.2.1.14 ISDDN

```
#define ISDDN(
             c ) (isalnum(c) || (c)==C_DLR || (c)==C_HSH || (c)==C_ATS)
```

7.2.1.15 fopen_tmp

```
#define fopen_tmp( ) tmpfile()
```

7.2.1.16 fclose_tmp

```
#define fclose_tmp(
             fp ) fclose((fp))
```

7.2.1.17 remove_hfq

```
#define remove_hfq(
             n ) remove(n)
```

7.2.1.18 CLERTC_OK

```
#define CLERTC_OK 0
```

0 - command line, command syntax, mapping, execution and finish of the command was successful

7.2.1.19 CLERTC_INF

```
#define CLERTC_INF 1
```

1 - command line, command syntax, mapping, execution and finish of the command was successful but a warning can be found in the log

7.2.1.20 CLERTC_FIN

```
#define CLERTC_FIN 2
```

2 - command line, command syntax, mapping, execution was successful but cleanup of the command failed (may not happened)

7.2.1.21 CLERTC_WRN

```
#define CLERTC_WRN 4
```

4 - command line, command syntax and mapping was successful but execution of the command returns with a warning

7.2.1.22 CLERTC_RUN

```
#define CLERTC_RUN 8
```

8 - command line, command syntax and mapping was successful but execution of the command returns with an error

7.2.1.23 CLERTC_MAP

```
#define CLERTC_MAP 12
```

12 - command line and command syntax was OK but mapping failed

7.2.1.24 CLERTC_SYN

```
#define CLERTC_SYN 16
```

16 - command line was OK but command syntax was wrong

7.2.1.25 CLERTC_CMD

```
#define CLERTC_CMD 20
```

20 - command line was wrong (user error)

7.2.1.26 CLERTC_INI

```
#define CLERTC_INI 24
```

24 - initialization of parameter structure for the command failed (may not happened)

7.2.1.27 CLERTC_CFG

```
#define CLERTC_CFG 28
```

28 - configuration is wrong (user error)

7.2.1.28 CLERTC_TAB

```
#define CLERTC_TAB 32
```

32 - table error (something within the predefined tables is wrong)

7.2.1.29 CLERTC_SYS

```
#define CLERTC_SYS 36
```

36 - system error (mainly memory allocation or some thing like this failed)

7.2.1.30 CLERTC_ACS

```
#define CLERTC_ACS 40
```

40 - access control or license error

7.2.1.31 CLERTC_ITF

```
#define CLERTC_ITF 44
```

44 - interface error (parameter pointer equals to NULL or something like this)

7.2.1.32 CLERTC_MEM

```
#define CLERTC_MEM 48
```

48 - memory allocation failed (e.g. dynamic string handling)

7.2.1.33 CLERTC_FAT

```
#define CLERTC_FAT 64
```

64 - fatal error (basic things are damaged)

7.2.1.34 CLERTC_MAX

```
#define CLERTC_MAX 64
```

maximal condition code value (greater condition codes are special return codes)

7.2.1.35 strncpy

```
#define strncpy(
              ...  ) Error:  Do not use strncpy!  Use strlcpy instead!
```

7.2.1.36 CSTIME_BUFSIZ

```
#define CSTIME_BUFSIZ 24
```

Build time string
Convert a time integer to a 20 byte time string of form YYYY-MM-DD HH:MM:SS.

Parameters

in	*t*	time in seconds since 1970 or 0 for current time
in	*p*	NULL to return a static variable or a pointer where the 20 bytes are copied in

Returns

pointer to the time string

7.2.1.37 HSH_PBRK

```
#define HSH_PBRK "#" /*nodiac*/
```

7.2.1.38 ATS_PBRK

```
#define ATS_PBRK "@" /*nodiac*/
```

7.2.1.39 C_EXC

```
#define C_EXC '!' /*nodiac*/
```

7.2.1.40 C_HSH

```
#define C_HSH '#' /*nodiac*/
```

7.2.1.41 C_DLR

```
#define C_DLR '$' /*nodiac*/
```

7.2.1.42 C_ATS

```
#define C_ATS '@' /*nodiac*/
```

7.2.1.43 C_SBO

```
#define C_SBO '[' /*nodiac*/
```

7.2.1.44 C_BSL

```
#define C_BSL '\\' /*nodiac*/
```

7.2.1.45 C_SBC

```
#define C_SBC ']' /*nodiac*/
```

7.2.1.46 C_CRT

```
#define C_CRT '^' /*nodiac*/
```

7.2.1.47 C_GRV

```
#define C_GRV '`' /*nodiac*/
```

7.2.1.48 C_CBO

```
#define C_CBO '{' /*nodiac*/
```

7.2.1.49 C_VBR

```
#define C_VBR '|' /*nodiac*/
```

7.2.1.50 C_CBC

```
#define C_CBC '}' /*nodiac*/
```

7.2.1.51 C_TLD

```
#define C_TLD '~' /*nodiac*/
```

7.2.1.52 S_EXC

```
#define S_EXC "!" /*nodiac*/
```

7.2.1.53 S_HSH

```
#define S_HSH "#" /*nodiac*/
```

7.2.1.54 S_DLR

```
#define S_DLR "$" /*nodiac*/
```

7.2.1.55 S_ATS

```
#define S_ATS "@" /*nodiac*/
```

7.2.1.56 S_SBO

```
#define S_SBO "[" /*nodiac*/
```

7.2.1.57 S_BSL

```
#define S_BSL "\\" /*nodiac*/
```

7.2.1.58 S_SBC

```
#define S_SBC "]" /*nodiac*/
```

7.2.1.59 S_CRT

```
#define S_CRT "^" /*nodiac*/
```

7.2.1.60 S_GRV

```
#define S_GRV "`" /*nodiac*/
```

7.2.1.61 S_CBO

```
#define S_CBO "{" /*nodiac*/
```

7.2.1.62 S_VBR

```
#define S_VBR "|" /*nodiac*/
```

7.2.1.63 S_CBC

```
#define S_CBC "}" /*nodiac*/
```

7.2.1.64 S_TLD

```
#define S_TLD "~" /*nodiac*/
```

7.2.1.65 S_SVB

```
#define S_SVB "=|" /*nodiac*/
```

7.2.1.66 S_SBS

```
#define S_SBS "/\\" /*nodiac*/
```

7.2.1.67 S_IDT

```
#define S_IDT "--|" /*nodiac*/
```

7.2.1.68 esrprintc

```
#define esrprintc srprintc
```

7.2.1.69 esnprintf

```
#define esnprintf snprintf
```

7.2.1.70 esprintf

```
#define esprintf sprintf
```

7.2.1.71 efprintf

```
#define efprintf fprintf
```

7.2.2 Typedef Documentation

7.2.2.1 TsEnVarList

```
typedef struct EnVarList TsEnVarList
```

7.2.2.2 TsDiaChr

```
typedef struct DiaChr TsDiaChr
```

7.2.3 Function Documentation

7.2.3.1 fopen_hfq()

```
FILE* fopen_hfq (
            const char * name,
            const char * mode )
```

7.2.3.2 fopen_hfq_nowarn()

```
FILE* fopen_hfq_nowarn (
            const char * name,
            const char * mode )
```

7.2.3.3 freopen_hfq()

```
FILE* freopen_hfq (
            const char * name,
            const char * mode,
            FILE * stream )
```

7.2.3.4 getFileSize()

```
long long getFileSize (
            const char * name )
```

7.2.3.5 userid()

```
char* userid (
            const int size,
            char * buffer )
```

Returns the current user id.

Parameters

size	size of the buffer
buffer	pointer to the buffer

Returns

pointer to the buffer containing the current user id (null-terminated)

7.2.3.6 homedir()

```
char* homedir (
            const int flag,
            const int size,
            char * buffer )
```

Returns the current home directory.

Parameters

flag	if true then slash/backslash are added
size	size of the string buffer
buffer	pointer to the buffer

Returns

pointer to the buffer containing the current home directory (null-terminated)

7.2.3.7 duserid()

```
char* duserid (
            void  )
```

Returns the current user id.

Returns

pointer to the buffer containing the current user id (null-terminated) must be freed by the caller

7.2.3.8 dhomedir()

```
char* dhomedir (
            const int flag )
```

Returns the current home directory.

Parameters

flag	if true then slash/backslash are added

Returns

pointer to the buffer containing the current home directory (null-terminated) must be freed by the caller

7.2.3.9 safe_getenv()

```
char* safe_getenv (
            const char * name,
            char * buffer,
            size_t bufsiz )
```

Gets an environment variable and stores it in the provided buffer. If the buffer is not large enough, the variable value is truncated.

Parameters

name	Name of the environment variable
buffer	Pointer to the buffer for the variable value
bufsiz	Size of the buffer

Returns

If bufsiz $>$ 0, returns the buffer pointer which contains a null-terminated string or NULL (variable does not exist). If bufsiz == 0, buffer is returned unmodified.

7.2.3.10 unEscape()

```
char* unEscape (
            const char * input,
            char * output )
```

Un-escape a string as part of special character support in EBCDIC codepages (static version).

Parameters

input	pointer to the input string containing the escape sequences
output	pointer to the output string for un-escaping (could be equal to the input pointer)

Returns

pointer to the un-escaped output or NULL if error

7.2.3.11 dynUnEscape()

```
char* dynUnEscape (
            const char * input )
```

Un-escape a string as part of special character support in EBCDIC codepages (dynamic version).

Parameters

input	pointer to the input string containing the escape sequences

Returns

pointer to the un-escaped output or NULL if error (calling application must free the memory)

7.2.3.12 printd()

```
int printd (
            const char * format,
             ...  )
```

Works like printf but print only in debug mode.

Parameters

format	format string

Returns

amount of characters printed (0 are mainly a error)

7.2.3.13 snprintc()

```
int int snprintc (
            char * buffer,
            const size_t size,
            const char * format,
             ...  )
```

Works like snprintf but concatenates the format string to the buffer.

Parameters

buffer	pointer to the string buffer
size	size of the string buffer
format	format string

Returns

amount of characters printed (0 are mainly a error)

7.2.3.14 srprintc()

```
int int int srprintc (
            char ** buffer,
            size_t * size,
            const size_t expansion,
            const char * format,
             ...  )
```

Works like snprintf but does reallocation of the buffer (maximal expansion of the format string can be specified).

Parameters

buffer	pointer to pointer to the string buffer (is updated, could be NULL at beginning)
size	pointer to size of the string buffer (is updated, could be 0 at beginning)
expansion	maximal expected expansion of the format string (size must be fit strlen(∗buffer)+strlen(format)+expansion+1)
format	format string

Returns

amount of characters printed (0 are mainly a error)

7.2.3.15 srprintf()

```
int int int int srprintf (
            char ** buffer,
            size_t * size,
            const size_t expansion,
            const char * format,
             ...  )
```

Works like snprintc but does reallocation of the buffer (maximal expansion of the format string can be specified).

Parameters

buffer	pointer to pointer to the string buffer (is updated, could be NULL at beginning)
size	pointer to size of the string buffer (is updated, could be 0 at beginning)
expansion	maximal expected expansion of the format string (size must be fit strlen(format)+expansion+1)
format	format string

Returns

amount of characters printed (0 are mainly a error)

7.2.3.16 fprintm()

```
int int int int void fprintm (
            FILE * file,
            const char * own,
            const char * pgm,
            const char * bld,
            const char * man,
            const int cnt )
```

Prints man pages to a file, inserting owner, program name, build number, state and date into placeholders

Parameters

file	pointer to the file
own	owner name for replacement (&{OWN})
pgm	program name for replacement (&{PGM})
bld	build/version string for replacement (&{BLD})
man	manpage to print, which can contain &{PGM}, &{OWN}, &{BLD}, &{DATE} and &{STATE}
cnt	amount of ' ' added to man page (0,1,2 (>2=2))

7.2.3.17 snprintm()

```
int snprintm (
            char * buffer,
            size_t size,
            const char * own,
            const char * pgm,
            const char * bld,
            const char * man,
            const int cnt )
```

Prints man pages to a buffer, inserting owner, program name, build number, state and date into placeholders

Parameters

buffer	pointer to the buffer
size	size of the buffer
own	owner name for replacement (&{OWN})
pgm	program name for replacement (&{PGM})
bld	build/version string for replacement (&{BLD})
man	manpage to print, which can contain &{PGM}, &{OWN}, &{BLD}, &{DATE} and &{STATE}
cnt	amount of ' ' added to man page (0,1,2 (>2=2))

Returns

same as snprintf

7.2.3.18 prsdstr()

```
const char* prsdstr (
            const char ** hdl,
            const char * str,
            int len )
```

This function parses a zero terminated string array of a certain length or terminated with 0xFF. Such a string array is the result of a variable length array of strings provided by CLP. If you don't know the length, please provide a negative number to look for 0xFF termination. For 0xFF termination you must define the CLPFLG_DLM. This is useful if there is no capability to use the ELN link to determine the length of the string array. Each call returns the pointer to the next string in the array, if no string is found anymore NULL is returned.

Parameters

hdl	pointer of pointer to string initialized with NULL at beginning

Parameters

str	pointer to the string arrays from CLP
len	-1 for 0xFF delimiter parsing or the ELN of the string array

Returns

pointer to one string or NULL if no more string

7.2.3.19 strlcpy()

```
size_t strlcpy (
            char * dest,
            const char * src,
            size_t n )
```

Works like strncpy but ensures null-termination.

Parameters

dest	pointer to destination string
src	pointer to source string
n	size of memory available for buffer

Returns

number of bytes actually copied (excludes NUL-termination)

7.2.3.20 getenvar()

```
char* getenvar (
            const char * name,
            const size_t length,
            const size_t size,
            char * string )
```

Get environment variable and handle HOME, USER, CUSEr, Cuser, cuser, OWNER, ENVID if not defined

Parameters

name	environment variable name
length	optional length of the name if no zero termination (0 if zero termination)
size	size of string
string	containing the value for the corresponding environment variable

Returns

pointer to string

7.2.3.21 mapstr()

```
char* mapstr (
            char * string,
            int size )
```

Replace all environment variables enclosed with '<' and '>' to build a string

Parameters

string	string for replacement
size	size of replacement string

Returns

pointer to string

7.2.3.22 dmapstr()

```
char* dmapstr (
            const char * string,
            int method )
```

Replace all environment variables enclosed with '<' and '>' to build a dynamic string

Parameters

string	string for replacement
method	conversion method (1 - to upper, 2 - to lower, else nothing)

Returns

pointer to the new allocated string or NULL if error

7.2.3.23 dmapxml()

```
char* dmapxml (
            const char * string,
            int method )
```

Replace all environment variables enclosed with '(' and ')' to build a dynamic string

Parameters

string	string for replacement
method	conversion method (1 - to upper, 2 - to lower, else nothing)

Returns

pointer to the new allocated string or NULL if error

7.2.3.24 mapfil()

```
char* mapfil (
            char * file,
            int size )
```

Replace '~' with "<HOME>" and all environment variables enclosed with '<' and '>' to build a file name

Parameters

file	string for replacement
size	size of replacement string

Returns

pointer to file

7.2.3.25 dmapfil()

```
char* dmapfil (
            const char * file,
            int method )
```

Replace '∼' with "<HOME>" and all environment variables enclosed with '<' and '>' to build a dynamic file name

Parameters

file	string for replacement
method	conversion method (1 - to upper, 2 - to lower, else nothing)

Returns

pointer to the new allocated string or NULL if error

7.2.3.26 maplab()

```
char* maplab (
            char * label,
            int size,
            int toUpper )
```

Replace '!' with ENVID, '∼' with "<SYSUID>", '^' with "<OWNERID>" and all environment variables enclosed with '<' and '> to build a key label'

Parameters

label	string for replacement
size	size of replacement string
toUpper	for mapping file to upper

Returns

pointer to label

7.2.3.27 dmaplab()

```
char* dmaplab (
            const char * label,
            int method )
```

Replace '!' with ENVID, '∼' with "<SYSUID>", '^' with "<OWNERID>" and all environment variables enclosed with '<' and '> to build a dynamic key label'

Parameters

label	string for replacement
method	conversion method (1 - to upper, 2 - to lower, else nothing)

Returns

pointer to new allocated string or NULL if error

7.2.3.28 cpmapfil()

```
char* cpmapfil (
            char * dest,
            int size,
            const char * source )
```

Replace '∼' with "<HOME>" and all environment variables enclosed with '<' and '>'

Parameters

dest	string for replacement
size	size of replacement string
source	original string

Returns

pointer to dest

7.2.3.29 dcpmapfil()

```
char* dcpmapfil (
            const char * file )
```

Replace '∼' with "<HOME>" and all environment variables enclosed with '<' and '>' and returns a dynamic string

Parameters

file	string for replacement

Returns

pointer to dynamic allocated string

7.2.3.30 cpmaplab()

```
char* cpmaplab (
            char * label,
            int size,
            const char * templ,
            const char * values,
            int toUpper )
```

Use rpltpl() and maplab() to build key label names, based on key label templates

Parameters

label	string for replacement
size	size of replacement string
templ	key label template (with x)
values	value string for replacement (x:s)
toUpper	for mapping label to upper

Returns

pointer to label

7.2.3.31 dcpmaplab()

```
char* dcpmaplab (
            const char * templ,
            const char * values,
            int method )
```

Use drpltpl() and dmaplab() to build key label names, based on key label templates in dynamic form

Parameters

templ	key label template (with x)
values	value string for replacement (x:s)
method	conversion method (1 - to upper, 2 - to lower, else nothing)

Returns

pointer to dynamic allocated string

7.2.3.32 localccsid()

```
unsigned int localccsid (
            void  )
```

Determines the local CCSID by querying nl_langinfo() (POSIX) or GetCPInfoEx() (Windows). If none of both are available or no valid CCSID can be determined, mapl2c() is called. If it also fails to determine the system default CSSID, the CCSID for ASCII (ISO8859-1) or IBM-1047 (EBCDIC platforms) is returned.

Returns

A supported CCSID > 0

7.2.3.33 mapl2c()

```
const char* mapl2c (
            unsigned isEBCDIC )
```

Map environment variable LANG to CCSID

Parameters

isEBCDIC	if true returns EBCDIC code pages else ASCII

Returns

NULL in case of an error or pointer to a static string containing the CCSID

7.2.3.34 lng2ccsd()

```
const char* lng2ccsd (
            const char * pcLang,
            unsigned isEbcdic )
```

Map environment variable LANG to CCSID

Parameters

pcLang	string containing the value of the environment variable LANG
isEbcdic	if true returns EBCDIC code pages else ASCII

Returns

NULL in case of an error or pointer to a static string containing the CCSID

7.2.3.35 mapccsid()

```
const char* mapccsid (
            const unsigned int uiCcsId )
```

Map CCSID in encoding string

Parameters

ui↩ CcsId	CCSID

Returns

encoding string

7.2.3.36 mapcdstr()

```
unsigned int mapcdstr (
            const char * p )
```

Map encoding string in CCSID

Parameters

p	encoding string

Returns

CCSID

7.2.3.37 bin2hex()

```
unsigned int bin2hex (
```

```
            const unsigned char * bin,
            char * hex,
            const unsigned int len )
```

Convert binary to hex

Parameters

bin	binary blob
hex	hex string
len	length of binary blob

Returns

amount of converted bytes

7.2.3.38 hex2bin()

```
unsigned int hex2bin (
            const char * hex,
            unsigned char * bin,
            const unsigned int len )
```

Convert from hex to binary

Parameters

hex	hex string
bin	binary string
len	length of hex string

Returns

amount of converted bytes

7.2.3.39 chr2asc()

```
unsigned int chr2asc (
            const char * chr,
            char * asc,
            const unsigned int len )
```

Convert character string to US-ASCII(UTF-8) and stops at not convertible chars

Parameters

chr	character string
asc	ASCII string
len	length

Returns

amount of converted bytes

7.2.3.40 chr2ebc()

```
unsigned int chr2ebc (
            const char * chr,
            char * ebc,
            const unsigned int len )
```

Convert character string to EBCDIC (only non variant characters) and stops at not convertible chars

Parameters

chr	character string
ebc	EBCDIC string
len	length

Returns

amount of converted bytes

7.2.3.41 asc2chr()

```
unsigned int asc2chr (
            const char * asc,
            char * chr,
            const unsigned int len )
```

Convert ASCII to character string (only non variant characters) and stops at not convertible chars

Parameters

asc	ASCII string
chr	character string
len	length

Returns

amount of converted bytes

7.2.3.42 asc_chr()

```
void asc_chr (
            const char * asc,
            char * chr,
            const unsigned int len )
```

Convert ASCII to character string (only non variant characters) and replace not convertible chars with '_'

Parameters

asc	ASCII string
chr	character string
len	length

7.2.3.43 chr_asc()

```
void chr_asc (
            const char * chr,
            char * asc,
            const unsigned int len )
```

Convert character string to US-ASCII(UTF-8) and replace not convertible chars with '_'

Parameters

chr	character string
asc	ASCII string
len	length

7.2.3.44 ebc2chr()

```
unsigned int ebc2chr (
            const char * ebc,
            char * chr,
            const unsigned int len )
```

Convert EBCDIC to character string (only non variant characters) and stops at not convertible chars

Parameters

ebc	EBCDIC string
chr	character string
len	length

Returns

amount of converted bytes

7.2.3.45 ebc_chr()

```
void ebc_chr (
            const char * ebc,
            char * chr,
            const unsigned int len )
```

Convert EBCDIC to character string (only non variant characters) and replace not convertible chars with '_'

Parameters

ebc	EBCDIC string
chr	character string
len	length

7.2.3.46 chr_ebc()

```
void chr_ebc (
            const char * chr,
            char * ebc,
```

```
            const unsigned int len )
```

Convert character string to EBCDIC (only non variant characters) and replace not convertible chars with '_'

Parameters

chr	character string
ebc	EBCDIC string
len	length

7.2.3.47 file2str()

```
int file2str (
            void * hdl,
            const char * filename,
            char ** buf,
            int * bufsize,
            char * errmsg,
            const int msgsiz )
```

Read a file using the specified filename and reads the whole content into the supplied buffer. The buffer is reallocated and bufsize updated, if necessary.
This is the default implementation if no file to string function for CLEP provided

Parameters

hdl	is ignored and not used (required for default implementation of call back function)
filename	The path and name of the file to read
buf	A pointer to a pointer to a buffer, may be a pointer to NULL for allocation else reallocation
bufsize	A pointer to the size of buf, is updated ater the call
errmsg	Pointer to a provided buffer for the error message (optional (can be NULL), result is null terminated)
msgsiz	The size of the buffer for the error message (optional (can be 0))

Returns

A positive value indicates the number of bytes read and copied into buf. A negative value indicates an error, in which case the content of buf is undefined. Error codes:

- -1: invalid arguments
- -2: fopen() failed
- -3: integer overflow, file too big
- -4: realloc() failed
- -5: file read error

7.2.3.48 arry2str()

```
int arry2str (
            char * array[],
            const int count,
            const char * separ,
            const int separLen,
            char ** out,
            int * outlen )
```

Takes an array of null-terminated strings and concatenates all strings into one single string separated by the specified separator. The resulting string is put into the out buffer which may be reallocated if necessary. If the buffer already contain a string the remaining strings are concatenated.

Parameters

array	Input array of null-terminated strings.
count	Number of string in array
separ	Separator of arbitrary length (may be NULL if separLen=0)
separLen	length of separator
out	Pointer to an output buffer (may be reallocated)
outlen	Size of output buffer

Returns

Error codes:

- 0: success
- -1: invalid arguments
- -2: realloc() failed

7.2.3.49 strxcmp()

```
int strxcmp (
            const int ca,
            const char * s1,
            const char * s2,
            const int n,
            const int c,
            const int f )
```

Compare of two string
The procedure combines strcmp, stricmp, strncmp and strchr in one function.

Parameters

in	*ca*	Flag if case sensitiv (TRUE) or not (FALSE)
in	*s1*	String 1 to compare
in	*s2*	string 2 to compare
in	*n*	If c!=0 then minimum else maximum amount of character to compare (0=disabled)
in	*c*	Character where the compare stops or -1 for keyword syntax
in	*f*	If true only compare up to null termination or stop char if false (normal compare) including null termination or stop char

Returns

signed integer with 0 for equal and !=0 for different

7.2.3.50 cstime()

```
char* cstime (
            signed long long t,
            char * p )
```

7.2.3.51 loadEnvars()

```
int loadEnvars (
            const unsigned int uiLen,
            const char * pcBuf,
            FILE * pfOut,
            FILE * pfErr,
            TsEnVarList ** ppList )
```

Load environment variables from buffer

Parameters

in	*uiLen*	Length of the buffer with environment variables
in	*pcBuf*	Buffer containing list of environment variables (ASCII or EBCDIC separated by new line or semicolon)
in	*pfOut*	File pointer for output messages
in	*pfErr*	File pointer for error messages
out	*ppList*	Pointer to an optional envar list for reset (the list and each string must freed by caller)

Returns

>=0 amount of successful defined environment variables else -1∗CLERTCs

7.2.3.52 readEnvars()

```
int readEnvars (
            const char * pcFil,
            FILE * pfOut,
            FILE * pfErr,
            TsEnVarList ** ppList )
```

Read and set environment variables from file

Parameters

in	*pcFil*	Filename, if pcFil==NULL use "DD:STDENV" instead
in	*pfOut*	File pointer for output messages
in	*pfErr*	File pointer for error messages
out	*ppList*	Pointer to an optional envar list for reset (the list and each string must freed by caller)

Returns

>=0 amount of successful defined environment variables else -1∗CLERTCs

7.2.3.53 envarInsert()

```
int envarInsert (
            TsEnVarList ** ppList,
            const char * pcName,
            const char * pcValue )
```

Store envars in a list for reset

Parameters

out	*ppList*	Pointer to envar list for reset

Parameters

in	*pcName*	Name of the environment variable
in	*pcValue*	Value of the environment variable (NULL for unset)

7.2.3.54 resetEnvars()

```
int resetEnvars (
            TsEnVarList ** ppList )
```

Reset list of environment variables

Parameters

in	*ppList*	Pointer to envar lisl

Returns

amount of envars reset or -1∗CLERTCs

7.2.3.55 init_diachr()

```
void init_diachr (
            TsDiaChr * psDiaChr,
            const unsigned int uiCcsId )
```

7.3 CLPDEF.h File Reference

Definitions for **C**ommand **L**ine **P**arsing.

Data Structures

- struct ClpError

  Defines a structure with error information.
- struct ClpArgument

  Table structure for arguments.

Macros

- #define CLP_OK 0

  Return code for a successful parsing: 0, otherwize > 0.
- #define CLPERR_LEX -1

  Lexical error (determined by scanner).
- #define CLPERR_SYN -2

  Syntax error (determined by parser).
- #define CLPERR_SEM -3

  Semantic error (determined by builder).
- #define CLPERR_TYP -4

  Type error (internal error with argument types).
- #define CLPERR_TAB -5

  Table error (internal error with argument tables).
- #define CLPERR_SIZ -6

Size error (internal error with argument tables and data structures).

- #define CLPERR_PAR -7

  Parameter error (internal error with argument tables and data structures).
- #define CLPERR_MEM -8

  Memory error (internal error with argument tables and data structures).
- #define CLPERR_INT -9

  Internal error (internal error with argument tables and data structures).
- #define CLPERR_SYS -10

  System error (internal error with argument tables and data structures).
- #define CLPERR_AUT -11

  Authorization request failed.
- #define CLPSRC_CMD ":command line:"

  From command line.
- #define CLPSRC_PRO ":property list:"

  From property list.
- #define CLPSRC_DEF ":default value:"

  From default value.
- #define CLPSRC_ENV ":environment variable:"

  From environment variable.
- #define CLPSRC_PRF ":property file:"

  From property file.
- #define CLPSRC_CMF ":command file:"

  From command file.
- #define CLPSRC_PAF ":parameter file:"

  Parameter file.
- #define CLPSRC_SRF ":string file:"

  String file.
- #define CLPTYP_NON 0

  No type - Mark the end of an argument table.
- #define CLPTYP_SWITCH 1

  Switch (single keyword representing a number (OID)).
- #define CLPTYP_NUMBER 2

  Signed or unsigned integer number (8, 16, 32 or 64 bit).
- #define CLPTYP_FLOATN 3

  Floating point number (32 or 64 bit).
- #define CLPTYP_STRING 4

  String literal (binary (HEX, ASCII, EBCDIC, CHARS) or null-terminated (default)).
- #define CLPTYP_OBJECT 5

  Object (KEYWORD(parameter_list)) can contain arbitrary list of other types.
- #define CLPTYP_OVRLAY 6

  Overlay (KEYWORD.KEYWORD...) contains one of its list as in a C union.
- #define CLPTYP_XALIAS -1

  For alias definition (used in the corresponding table macro)
- #define CLPCLS_MTD_ALL 1

  Complete close, free anything including the dynamic allocated buffers in the CLP structure.
- #define CLPCLS_MTD_KEP 0

  Free anything except the allocated memory in CLP structure and keep the handle open to close it later with method ALL.
- #define CLPCLS_MTD_EXC 2

  Free anything including the handle except the allocated memory in the CLP structure, the application must free the dynamic allocated buffers in the CLP structure it self.

- #define CLPPRO_MTD_ALL 0

  All properties are printed (manual pages added as comment).
- #define CLPPRO_MTD_SET 1

  Only defined properties are printed (no manual pages used).
- #define CLPPRO_MTD_CMT 2

  All properties are printed, but not defined properties are line comments .
- #define CLPPRO_MTD_DOC 3

  All property only parameter are printed as documentation.
- #define CLPFLG_NON 0x00000000U

  To define no special flags.
- #define CLPFLG_ALI 0x00000001U

  This parameter is an alias for another argument (set by macros).
- #define CLPFLG_CON 0x00000002U

  This parameter is a constant definition (no argument, no link, no alias (set by macros)).
- #define CLPFLG_CMD 0x00000004U

  If set the parameter is only used within the command line (command line only).
- #define CLPFLG_PRO 0x00000008U

  If set the parameter is only used within the property file (property file only).
- #define CLPFLG_SEL 0x00000010U

  If set only the predefined constants over the corresponding key words can be selected (useful to define selections).
- #define CLPFLG_FIX 0x00000020U

  This argument has a fixed length (only useful for strings if a typedef defines a fixed length per element, else set internally).
- #define CLPFLG_BIN 0x00000040U

  This argument can contain binary data without null termination (length must be known or determined with a link).
- #define CLPFLG_DMY 0x00000080U

  If set the parameter is not put in the symbol table, meaning it is only a peace of memory in the CLP structure.
- #define CLPFLG_CNT 0x00000100U

  This link will be filled by the calculated amount of elements (useful for arrays).
- #define CLPFLG_OID 0x00000200U

  This link will be filled by the object identifier (OID) of the chosen argument (useful for overlays).
- #define CLPFLG_IND 0x00000400U

  This link will be filled with the index (position) in the CLP string (byte offset of the current key word).
- #define CLPFLG_HID 0x00000800U

  If set the parameter is not visible, meaning it is a hidden parameter.
- #define CLPFLG_ELN 0x00001000U

  This link will be filled by the calculated length of an element (fixed types == data size, packed types == data length).
- #define CLPFLG_SLN 0x00002000U

  This link will be filled by the calculated string length for an element (only for null-terminated strings).
- #define CLPFLG_TLN 0x00004000U

  This link will be filled by the calculated total length for the argument (sum of all element lengths).
- #define CLPFLG_DEF 0x00010000U

  This flag enables to use the OID as default for numbers if no value is assigned (only the keyword is used (syntax extension)).
- #define CLPFLG_CHR 0x00020000U

  This flag will set the default method of interpretation of a binary string to local character string (DEFAULT).
- #define CLPFLG_ASC 0x00040000U

  This flag will set the default method of interpretation of a binary string to ASCII.
- #define CLPFLG_EBC 0x00080000U

  This flag will set the default method of interpretation of a binary string to EBCDIC.
- #define CLPFLG_HEX 0x00100000U

This flag will set the default method of interpretation of a binary string to hexadecimal.

- #define CLPFLG_PDF 0x00200000U

  This flag will be set if a property value was defined from outside, it will be FALSE if the property value was hard coded in the tables.

- #define CLPFLG_TIM 0x00400000U

  This flag mark a number as time value (only used to print out the corressponing time stamp).

- #define CLPFLG_DYN 0x00800000U

  This flag mark a string or array as dynamic (only a pointer to allocated memory is used and must be freeed by the user).

- #define CLPFLG_PWD 0x01000000U

  This flag will ensure that the clear value is only put into the data structure but not traced, logged or given away elsewhere.

- #define CLPFLG_DLM 0x02000000U

  This flag ensures that fix size arrays has a empty (initialized) last element (max-1) as delimiter. Additional you enforce 0xFF at the and of a non fix size string array (size-1).

- #define CLPFLG_UNS 0x04000000U

  Marks a number as unsigned (prevent negative values).

- #define CLPFLG_XML 0x08000000U

  Marks zero terminated string as XML path where '(' and ')' are used to replace environment variables.

- #define CLPFLG_FIL 0x10000000U

  Marks zero terminated string as file and replace additional '∼' by HOME and corrects the prefix for different platforms.

- #define CLPFLG_LAB 0x20000000U

  Marks zero terminated string as label and replace additional '∼' by USER, '^' by OWNER and '!' by ENVID .

- #define CLPFLG_UPP 0x40000000U

  Converts zero terminated strings to upper case.

- #define CLPFLG_LOW 0x80000000U

  Converts zero terminated strings to lower case.

- #define CLPCONTAB_OPN(name) TsClpArgument name[ ]

  Starts a table with constant definitions.

- #define CLPCONTAB_NUMBER(kyw, dat, man, hlp) {CLPTYP_NUMBER,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON ,NULL,NULL,(man),(hlp),(dat), 0.0 ,NULL ,NULL},

  Defines a number literal with the command line keyword kyw and the value dat.

- #define CLPCONTAB_FLOATN(kyw, dat, man, hlp) {CLPTYP_FLOATN,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON ,NULL,NULL,(man),(hlp), 0 ,(dat),NULL ,NULL},

  Defines a floating point literal with the command line keyword kyw and the value dat.

- #define CLPCONTAB_STRING(kyw, dat, man, hlp) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON ,NULL,NULL,(man),(hlp), 0 , 0.0 ,(U08∗)(dat),NULL},

  Defines a default string literal with the command line keyword kyw and the value dat.

- #define CLPCONTAB_HEXSTR(kyw, dat, man, hlp) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON|CLPFLG_HEX, ULL,NULL,(man),(hlp), 0 , 0.0 ,(U08∗)(dat),NULL},

  Defines a hexadecimal string literal with the command line keyword kyw and the value dat.

- #define CLPCONTAB_ASCSTR(kyw, dat, man, hlp) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON|CLPFLG_ASC, ULL,NULL,(man),(hlp), 0 , 0.0 ,(U08∗)(dat),NULL},

  Defines a ASCII string literal with the command line keyword kyw and the value dat.

- #define CLPCONTAB_EBCSTR(kyw, dat, man, hlp) {CLPTYP_STRING,(kyw),NULL,0,0, 0 ,0,0,CLPFLG_CON|CLPFLG_EBC, ULL,NULL,(man),(hlp), 0 , 0.0 ,(U08∗)(dat),NULL},

  Defines a EBCDIC string literal with the command line keyword kyw and the value dat.

- #define CLPCONTAB_BINARY(kyw, dat, siz, man, hlp) {CLPTYP_STRING,(kyw),NULL,0,0,(siz),0,0,CLPFLG_CON|CLPFLG_ ULL,NULL,(man),(hlp), 0 , 0.0 ,(U08∗)(dat),NULL},

  Defines a binary literal with the command line keyword kyw and the value dat.

- #define CLPCONTAB_CLS {CLPTYP_NON , NULL,NULL,0,0, 0 ,0,0,CLPFLG_NON ,NULL,NULL, NULL, NULL, 0 , 0.0 ,NULL ,NULL}

Typedefs

- typedef struct ClpError TsClpError

  Defines a structure with error information.
- typedef struct ClpArgument TsClpArgument

  Table structure for arguments.
- typedef int() TfF2S(void *pvGbl, void *pvHdl, const char *pcFil, char **ppBuf, int *piBuf, char *pcMsg, const int siMsg)

  Type definition for string to file call back function.
- typedef int() TfSaf(void *pvGbl, void *pvHdl, const char *pcVal)

  Type definition for resource access check.
- typedef int() TfClpPrintPage(void *pvHdl, const int siLev, const char *pcHdl, const char *pcPat, const char *pcFil, const char *pcOrg, const char *pcPge)

  Function 'prnHtmlDoc' of library 'libhtmldoc' called if built-in function HTMLDOC used.

7.3.1 Detailed Description

Definitions for **C**ommand **L**ine **P**arsing.

Author

limes datentechnik gmbh

Date

27.12.2019

Copyright

If you need professional services or support for this library please contact support@flam.de.

7.4 CLPMAC.h File Reference

Macros for single definition of C struct and argument table for **C**ommand **L**ine **P**arsing.
`#include <stddef.h>`

Macros

- #define CLPARGTAB_SKALAR(kyw, nam, typ, min, max, atyp, flg, oid, tab, dft, man, hlp) { atyp ,(kyw), NULL,(min), 1 ,sizeof(typ), offsetof(STRUCT_NAME,nam),(oid),(flg) ,(tab),(dft),(man),(hlp),0,0.0,NULL,#typ},

  Defines a scalar (single value) with the command line keyword kyw and the member name nam.

- #define CLPARGTAB_STRING(kyw, nam, siz, min, max, atyp, flg, oid, tab, dft, man, hlp) { CLPTYP_STRING,(kyw), NULL,(min),(max), (siz), offsetof(STRUCT_NAME,nam),(oid),(flg) ,(tab),(dft),(man),(hlp),0,0.↩ 0,NULL,NULL},

  Defines a string with the command line keyword kyw and the member name nam.
- #define CLPARGTAB_DYNSTR(kyw, nam, siz, min, max, atyp, flg, oid, tab, dft, man, hlp) { CLPTYP_STRING,(kyw), NULL,(min),(max), (siz), offsetof(STRUCT_NAME,nam),(oid),((flg)|CLPFLG_DYN),(tab),(dft),(man),(0,NULL,NULL},

  Defines a dynamic string with the command line keyword kyw and the member name nam (pointer to alloced memory, must be freed by the using application).
- #define CLPARGTAB_ARRAY(kyw, nam, typ, min, max, atyp, flg, oid, tab, dft, man, hlp) { atyp ,(kyw), NU↩ LL,(min),(max),sizeof(typ), offsetof(STRUCT_NAME,nam),(oid),(flg) ,(tab),(dft),(man),(hlp),0,0.0,NULL,#typ},

  Defines an array with the command line keyword kyw and the member name nam.
- #define CLPARGTAB_DYNARY(kyw, nam, typ, min, max, atyp, flg, oid, tab, dft, man, hlp) { atyp ,(kyw), NU↩ LL,(min),(max),sizeof(typ), offsetof(STRUCT_NAME,nam),(oid),((flg)|CLPFLG_DYN),(tab),(dft),(man),(hlp),0,0.↩ 0,NULL,#typ},

  Defines an dynamic array with the command line keyword kyw and the member name nam (pointer to alloced memory, must be freed by the using application).
- #define CLPARGTAB_ALIAS(kyw, ali) { CLPTYP_XALIAS,(kyw),(ali), 0 , 0 , 0 , 0 , 0 , CLPFLG_ALI, NULL, NULL, NULL, NULL,0,0.0,NULL,NULL},

  Defines an alias name for another argument.
- #define CLPARGTAB_CLS { CLPTYP_NON , NULL, NULL, 0 , 0 , 0 , 0 , 0 , 0 , NULL, NULL, NULL, NU↩ LL,0,0.0,NULL,NULL}

  Will mark the end of an argument table.

7.4.1 Detailed Description

Macros for single definition of C struct and argument table for **C**ommand **L**ine **P**arsing.

Author

Falk Reichbott

Date

20.10.2012

Copyright

7.4.2 CLP table macros

This file is commonly included 2 times in a C file where the command line arguments of a program are defined. The first inclusion is used to define the C struct types where the parsed argument values will be stored. The second inclusion is used to define a table which describes the available command line arguments. With this method one source of description is used to define the argument table as well as the C struct where the parsed values will be stored.
DEFINE_STRUCT acts as a switch to the CLPARGTAB_∗ macros and determines if they define members of a C struct or entries of the argument table.
If defined the macros define struct members. Otherwise they define entries of the argument table.

7.5 FLAMCLE.h File Reference

Definitions for **C**ommand **L**ine **E**xecution.
`#include "CLEDEF.h"`

Functions

- const char * pcCleVersion (const int l, const int s, char *b)

  Get CLE-version information.
- const char * pcCleAbout (const int l, const int s, char *b)

  Get about CLE-information.
- int siCleExecute (void *pvGbl, const TsCleCommand *psCmd, int argc, char *argv[], const char *pcOwn, const char *pcPgm, const char *pcAut, const char *pcAdr, const int isCas, const int isPfl, const int isRpl, const int isEnv, const int siMkl, FILE *pfOut, FILE *pfTrc, const char *pcDep, const char *pcOpt, const char *pcEnt, const char *pcLic, const char *pcBld, const char *pcVsn, const char *pcAbo, const char *pcHlp, const char *pcDef, TfMsg *pfMsg, const TsCleOtherClp *psOth, void *pvF2S, TfF2S *pfF2S, void *pvSaf, TfSaf *pfSaf, const char *pcDpa, const int siNoR, const TsCleDoc *psDoc)

  Execute CLE-command line.

7.5.1 Detailed Description

Definitions for **C**ommand **L**ine **E**xecution.

Author

limes datentechnik gmbh

Date

06.03.2015

Copyright

7.6 FLAMCLP.h File Reference

```
#include <stdio.h>
#include "CLPDEF.h"
```

Functions

- const char * pcClpVersion (const int l, const int s, char *b)

  Get version information.
- const char * pcClpAbout (const int l, const int s, char *b)

  Get about information.

- void * pvClpOpen (const int isCas, const int isPfl, const int isEnv, const int siMkl, const char *pcOwn, const char *pcPgm, const char *pcBld, const char *pcCmd, const char *pcMan, const char *pcHlp, const int isOvl, const TsClpArgument *psTab, void *pvDat, FILE *pfHlp, FILE *pfErr, FILE *pfSym, FILE *pfScn, FILE *pf↩ Prs, FILE *pfBld, const char *pcDep, const char *pcOpt, const char *pcEnt, TsClpError *psErr, void *pvGbl, void *pvF2S, TfF2S *pfF2S, void *pvSaf, TfSaf *pfSaf)

  Open command line parser.
- void vdClpReset (void *pvHdl)

  Reset command line parser.
- int siClpParsePro (void *pvHdl, const char *pcSrc, const char *pcPro, const int isChk, char **ppLst)

  Parse the property list.
- int siClpParseCmd (void *pvHdl, const char *pcSrc, const char *pcCmd, const int isChk, const int isPwd, int *piOid, char **ppLst)

  Parse the command line.
- int siClpSyntax (void *pvHdl, const int isSkr, const int isMin, const int siDep, const char *pcPat)

  Print command line syntax.
- const char * pcClpInfo (void *pvHdl, const char *pcPat)

  Give help message for given path.
- int siClpHelp (void *pvHdl, const int siDep, const char *pcPat, const int isAli, const int isMan)

  Print help for command line syntax.
- int siClpDocu (void *pvHdl, FILE *pfDoc, const char *pcPat, const char *pcNum, const char *pcKnd, const int isCmd, const int isDep, const int isMan, const int isAnc, const int isNbr, const int isIdt, const int isPat, const unsigned int uiLev)

  Generate documentation for command line syntax.
- int siClpPrint (void *pvHdl, const char *pcFil, const char *pcNum, const char *pcKnd, const int isCmd, const int isDep, const int isAnc, const int isNbr, const int isShl, const int isIdt, const int isPat, const unsigned int uiLev, const int siPs1, const int siPs2, const int siPr3, void *pvPrn, TfClpPrintPage *pfPrn)

  Generate documentation using a callback function.
- int siClpProperties (void *pvHdl, const int siMtd, const int siDep, const char *pcPat, FILE *pfOut)

  Generate properties.
- int siClpLexemes (void *pvHdl, FILE *pfOut)

  Print the lexems of the command line compiler.
- int siClpGrammar (void *pvHdl, FILE *pfOut)

  Print the grammar of the command line compiler.
- void vdClpClose (void *pvHdl, const int siMtd)

  Close the command line parser.
- void * pvClpAlloc (void *pvHdl, void *pvPtr, int siSiz, int *piInd)

  Allocate memory in CLP structure.
- char * pcClpError (int siErr)

  Provides error message.

Index