FLUCUP-API

1

Generated by Doxygen 1.8.14

Contents

Chapter 1

FLUC Subprogram Interface

This interface provides all FLCL commands (`CONV`, `XCNV`, ...) as simple to use programming interface. It is based on the Frankenstein Limes Universal Converter (FLUC).

The programming interface can be used to integrate the source (file) to target (file) data conversion capabilities provided by FLUC into a custom program.

Additionally, it provides interactive help, documentation, syntax and other information to support the use of the commands.

This interface is also provided as load module interface for COBOL, PL1 and other languages. The corresponding load module interface is called FLUCUPLB.

1.1 Interface standards

For each command, there is a function that accepts a command line and property file string that follows the FLCL syntax and produces the same return/condition code as the FLCL. If the command string starts with an = sign, then a parameter filename must follow. The file is expected to contain the actual command string:

```
r=flucconv("=para.txt",NULL,":STDERR",NULL);
```

corresponds to:

```
FLCL CONV=para.txt
```

As second parameter you can provide a property file (NULL means no properties). The third parameter controls the print outs. If you define for a command, then no printouts are done. Normally ":STDERR" is used for printouts, but you can also define ":STDOUT" or a file name. The last parameter is optional and activates the trace output.

For output files you can define ":STDERR" or ":STDOUT" to assign the corresponding streams.

1.2 Properties

The property file string is parsed before the command line string. The property string may contain command properties in the property file syntax of FLCL. You can provide the contents of a property file from an arbitrary source via the property string. To generate a property file you can use:

```
FLCL GENPROP CONV=filename
(works for any command by replacing CONV with the right command)
```

You can also provide the filename of a property file by passing an = sign followed by the filename. Properties are then read from that file:

```
r=flucconv("=para.txt","=prop.txt",NULL,NULL);
```

To use FLCL property files, the program name must still be the default "flcl". If you write a program which is not compatible to FLCL, please set another program name (see next section). To manage the contents of a property file, the function flucpro() is provided.

1.2.1 Program and owner names

The program and owner name can be managed by the corresponding functions. The default program name is "flcl". The default owner name is "limes". The default values are used, if a NULL pointer or an empty string is passed. The owner id and program name are limited to 64 bytes, including the null termination. Longer names are ignored.

1.3 Environment variables

For all default character conversions, it is useful to set the environment variable LANG. Other used environment variables of FLAM can be found in the FLCL manual. With version 5.1.19 a new function (flucenv()) to load the FLAM environment was introduced. This function can be used in front of the first API call to establish the same environment used by FLAM utilities, subsystems and so on. This give the application developer the possibility to adjust the environment before the first real call is done. Till 5.1.18 each function has read the system variables on z/OS. This is now part of the flucenv() function to give complete control about the environment to the user of the API.

1.4 Condition codes

The return codes of the subprogram are called condition codes and are returned on subprogram termination.

- 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 of 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 (e.g. load of environment or open of file failed)
- 40 - access control or license error
- 44 - interface error (e.g. parameter pointer equals NULL)
- 48 - memory allocation failed (e.g. dynamic string handling)
- 64 - fatal error (basic things are damaged)
- >64 - for few special conditions codes (see FLCLBOOK)

Special condition codes can be found in the FLCL manual (FLCLBOOK).

1.5 Reason codes

The reason code (internal FLAM return code) can be determined by calling the function flucrsn(). A corresponding text message is returned by the function flucmsg(). All FLAM reason codes are listed in FLCL manual (FLCLBOOK).

1.6 Sample programs

Sample programs in C with name SCFCUCNV/GZP/LST/REM can be found as part of the installation package for mainframe systems in the library SRCLIBC(SCFCUxxx), with the corresponding compile and link step in JO↩ BLIB(SBUILD). For other platforms (Windows, UNIX) the sample program source of SCFCUCNV/GZP/LST/REM is located in the `sample` directory. The compile and link procedures can be found in the Makefile of the same directory.

Chapter 2

Module Index

2.1 API definitions

Here is a list of the API parts:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Module Documentation

4.1 FLUC Function Codes

Macros

- #define FLUC_INFO 1
  INFO command.
- #define FLUC_CONV 2
  CONV command.
- #define FLUC_XCNV 3
  XCNV command.
- #define FLUC_ICNV 4
  ICNV command.
- #define FLUC_FLAM 5
  FLAM command.
- #define FLUC_XCHK 6
  XCHK command.
- #define FLUC_HASH 7
  HASH command.
- #define FLUC_UTIL 8
  UTIL command.
- #define FLUC_DIFF 9
  DIFF command.
- #define FLUC_KEY 10
  KEY command.

4.1.1 Detailed Description

Pass one of this to fluchlp(), flucsyn(), flucdoc() or flucpro() to get information about the corresponding command.

4.1.2 Macro Definition Documentation

4.1.2.1 FLUC_INFO

```
#define FLUC_INFO 1
```

INFO command.

4.1.2.2 FLUC_CONV

```
#define FLUC_CONV 2
```

CONV command.

4.1.2.3 FLUC_XCNV

```
#define FLUC_XCNV 3
```

XCNV command.

4.1.2.4 FLUC_ICNV

```
#define FLUC_ICNV 4
```

ICNV command.

4.1.2.5 FLUC_FLAM

```
#define FLUC_FLAM 5
```

FLAM command.

4.1.2.6 FLUC_XCHK

```
#define FLUC_XCHK 6
```

XCHK command.

4.1.2.7 FLUC_HASH

```
#define FLUC_HASH 7
```

HASH command.

4.1.2.8 FLUC_UTIL

```
#define FLUC_UTIL 8
```

UTIL command.

4.1.2.9 FLUC_DIFF

```
#define FLUC_DIFF 9
```

DIFF command.

4.1.2.10 FLUC_KEY

```
#define FLUC_KEY 10
```

KEY command.

4.2 Functions

Functions

- const char * flucvsn (void)

  Version.
- const char * flucabo (void)

  About.
- const char * fluclic (void)

  Get license text.
- const char * flucown (const char *own)

  Set/Get owner id.
- const char * flucpgm (const char *pgm)

  Set/Get program name.
- int flucenv (const int sys, const int std, const char *env)

  Load the FLAM environment.
- int flucinfo (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC INFO - Provides various information.
- int fluchash (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC HASH - Simplified Hash calculation.
- int flucconv (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC CONV - Simplified data conversion.
- int flucxcnv (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC XCNV - Extended data conversion.
- int flucxchk (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC XCHK - Extended data evalation.
- int flucicnv (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC ICNV - Character conversion like ICONV.
- int flucflam (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC FLAM - Support of FLAM4 command in FLUC.
- int flucutil (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC UTIL - Support of several utility function.
- int flucdiff (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC DIFF - Compares two data sources.
- int fluckey (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC KEY - Key management functions.
- int flucrsn (void)

  Get reason code.
- const char * flucmsg (const int rsn)

  Get a message for a reason code.
- void fluchlp (const int what, const int depth, const char *path, const char *out)

  Returns help information about the command string.
- void flucsyn (const int what, const int depth, const char *path, const char *out)

  Return syntax information about the command string.
- void flucdoc (const int what, const char *path, const char *out)

  Return documentation.
- void flucpro (const int what, const char *ipro, const char *prop, const char *opro, const char *out)

  Manage properties.

4.2.1 Detailed Description

4.2.2 Function Documentation

4.2.2.1 flucvsn()

```
const char* flucvsn (
            void  )
```

Version.

This function returns a string with version information for each used component. This should be used in a support case.

Returns

a pointer to a static area containing the zero-terminated version string

4.2.2.2 flucabo()

```
const char* flucabo (
            void  )
```

About.

This function returns a string with about information for this library on multiple lines and license information for used external libraries.

Returns

a pointer to a static area containing the zero-terminated about string

4.2.2.3 fluclic()

```
const char* fluclic (
            void  )
```

Get license text.

This function can be used to get the current license text on multiple lines. The license text defines the permissible use of this library.

Returns

a pointer to a static area with a zero-terminated string containing the current license text.

4.2.2.4 flucown()

```
const char* flucown (
            const char * own )
```

Set/Get owner id.

This function can be used to get (pcOwn==NULL) or to set the current owner id. The owner id is a logical qualifier to separate different configurations for different clients in the same property file. If you want use a configuration for another owner as "limes", then you must define the owner id with this function first.

Parameters

in	*own*	Define owner name (DEFAULT: "limes")

Returns

a pointer to a static area with a zero-terminated string containing the current owner id.

4.2.2.5 flucpgm()

```
const char* flucpgm (
            const char * pgm )
```

Set/Get program name.

This function can be used to get (pcPgm==NULL) or to set ("mypg") the current program name. If you implement an own property management or a program which does not conform to the FLCL infrastructure components, it might be useful to change the program name.

Parameters

in	*pgm*	Define program name (DEFAULT: "flcl")

Returns

a pointer to a static area with a zero-terminated string containing the current program name.

4.2.2.6 flucenv()

```
int flucenv (
            const int sys,
            const int std,
            const char * env )
```

Load the FLAM environment.

This function load the FLAM system variables (only on z/OS or in EDZ environment), the STDENV definitions ('DD:STDENV' or '<SYSUID>.STDENV' on z/OS or '.stdenv' file in working or home directory on other system) and a optional list if variables to the environment. You can adjust the environment before the first call to a FLAM API function is done. This function gives the possibility to work with the same environment used by FLAM utilities, subsystems aso.

Parameters

in	*sys*	True to load system variables, false no system variables active
in	*std*	True to load standard environment, false no standard environment variables active
in	*env*	Optional list (could be NULL or empty) of environment variables (KEYWORD=VALUE) separated by new line ('\n') or semicolon (';')

Returns

negative (<0) error else (>=0) amount of environment variable load

4.2.2.7 flucinfo()

```
int flucinfo (
             const char * cmd,
             const char * pro,
             const char * out,
             const char * trc )
```

FLUC INFO - Provides various information.

This function calls the INFO command of FLUC

Examples: e=flucinfo("get.file='test.gz'",NULL,NULL,NULL);

Parameters

in	*cmd*	Command string with GET and LOG instructions (if starts with '=', a parameter file containing the command string must follow)
in	*pro*	Property string with GET and LOG instructions (can be empty or NULL)
in	*out*	Filename for printouts (if NULL or empty then stderr is used)
in	*trc*	Filename for tracing (if NULL or empty then no trace is written)

Returns

signed integer with condition code for the command

4.2.2.8 fluchash()

```
int fluchash (
             const char * cmd,
             const char * pro,
             const char * out,
             const char * trc )
```

FLUC HASH - Simplified Hash calculation.

This function calls the HASH command of FLUC

Examples: e=fluchash("file='test.gz' algo=md5",NULL,":STDERR",NULL);

Parameters

in	*cmd*	Command string with FILE and LOG instructions (if starts with '=', a parameter file containing the command string must follow)
in	*pro*	Property string with FILE and LOG instructions (can be empty or NULL)
in	*out*	Filename for printouts (if NULL or empty then is no printout produced)
in	*trc*	Filename for tracing (if NULL or empty then no trace is written)

Returns

signed integer with condition code for the command

4.2.2.9 flucconv()

```
int flucconv (
            const char * cmd,
            const char * pro,
            const char * out,
            const char * trc )
```

FLUC CONV - Simplified data conversion.

This function calls the CONV command of FLUC

Examples: e=flucconv("read.text(file='test.txt') write.record()",NULL,":STDERR",NULL);

Parameters

in	*cmd*	Command string with READ, WRITE and LOG instructions (if starts with '=', a parameter file containing the command string must follow)
in	*pro*	Property string with READ, WRITE and LOG instructions (can be empty or NULL)
in	*out*	Filename for printouts (if NULL or empty then is no printout produced)
in	*trc*	Filename for tracing (if NULL or empty then no trace is written)

Returns

signed integer with condition code for the command

4.2.2.10 flucxcnv()

```
int flucxcnv (
            const char * cmd,
            const char * pro,
```

```
            const char * out,
            const char * trc )
```

FLUC XCNV - Extended data conversion.

This function calls the XCNV command of FLUC

Examples: e=flucxcnv("inp(sav.fil(fio.blk(name='test.txt') cnv.chr(from='IBM-1141' to='UTF-8') fmt.txt())) out(sav.↩ fil(fio.blk(name='test.out')))",NULL,":STDERR",NULL);

Parameters

in	*cmd*	Command string with INP, OUT and LOG instructions (if starts with '=', a parameter file containing the command string must follow)
in	*pro*	Property string with INP, OUT and LOG instructions (can be empty or NULL)
in	*out*	Filename for printouts (if NULL or empty then is no printout produced)
in	*trc*	Filename for tracing (if NULL or empty then no trace is written)

Returns

signed integer with condition code for the command

4.2.2.11 flucxchk()

```
int flucxchk (
            const char * cmd,
            const char * pro,
            const char * out,
            const char * trc )
```

FLUC XCHK - Extended data evalation.

This function calls the XCHK command of FLUC

Examples: e=flucxchk("inp(sav.fil(fio.blk(name='test.txt') cnv.hsh()))",NULL,":STDERR",NULL);

Parameters

in	*cmd*	Command string with INP and LOG instructions (if starts with '=', a parameter file containing the command string must follow)
in	*pro*	Property string with INP and LOG instructions (can be empty or NULL)
in	*out*	Filename for printouts (if NULL or empty then is no printout produced)
in	*trc*	Filename for tracing (if NULL or empty then no trace is written)

Returns

signed integer with condition code for the command

4.2.2.12 flucicnv()

```
int flucicnv (
            const char * cmd,
            const char * pro,
            const char * out,
            const char * trc )
```

FLUC ICNV - Character conversion like ICONV.

This function calls the ICNV command of FLUC

Examples: e=flucicnv("in='test.txt' from='IBM-1141' to='UTF-8' ENL2LF out='test.out'",NULL,":STDERR",NULL);

Parameters

in	*cmd*	Command string with ICONV parameter (if starts with '=', a parameter file containing the command string must follow)
in	*pro*	Property string with ICONV parameter (can be empty or NULL)
in	*out*	Filename for printouts (if NULL or empty then is no printout produced)
in	*trc*	Filename for tracing (if NULL or empty then no trace is written)

Returns

signed integer with condition code for the command

4.2.2.13 flucflam()

```
int flucflam (
            const char * cmd,
            const char * pro,
            const char * out,
            const char * trc )
```

FLUC FLAM - Support of FLAM4 command in FLUC.

This function calls the FLAM command of FLUC

Examples: e=flucflam("comp flamin='test.txt' flamfile='test.fla' inrecformat=undefined",NULL,":STDERR",NULL);

Parameters

in	*cmd*	Command string with FLAM parameter (if starts with '=', a parameter file containing the command string must follow)
in	*pro*	Property string with FLAM parameter (can be empty or NULL)
in	*out*	Filename for printouts (if NULL or empty then is no printout produced)
in	*trc*	Filename for tracing (if NULL or empty then no trace is written)

Returns

signed integer with condition code for the command

4.2.2.14 flucutil()

```
int flucutil (
             const char * cmd,
             const char * pro,
             const char * out,
             const char * trc )
```

FLUC UTIL - Support of several utility function.

This function calls the UTIL command of FLUC

Examples: e=flucutil("run.remove='∼.test.∗∗'",NULL,NULL,NULL);

Parameters

in	*cmd*	Command string with UTIL parameter (if starts with '=', a parameter file containing the command string must follow)
in	*pro*	Property string with UTIL parameter (can be empty or NULL)
in	*out*	Filename for printouts (if NULL or empty then is no printout produced)
in	*trc*	Filename for tracing (if NULL or empty then no trace is written)

Returns

signed integer with condition code for the command

4.2.2.15 flucdiff()

```
int flucdiff (
             const char * cmd,
             const char * pro,
             const char * out,
             const char * trc )
```

FLUC DIFF - Compares two data sources.

This function calls the DIFF command of FLUC

Examples: e=flucdiff("read.file'./test.txt.gz' compare.file='∼.test.fba'",NULL,":STDERR",NULL);

Parameters

in	*cmd*	Command string with DIFF parameter (if starts with '=', a parameter file containing the command string must follow)
in	*pro*	Property string with DIFF parameter (can be empty or NULL)
in	*out*	Filename for printouts (if NULL or empty then is no printout produced)
in	*trc*	Filename for tracing (if NULL or empty then no trace is written)

Returns

signed integer with condition code for the command

4.2.2.16 fluckey()

```
int fluckey (
            const char * cmd,
            const char * pro,
            const char * out,
            const char * trc )
```

FLUC KEY - Key management functions.

This function calls the KEY command of FLUC

Examples: e=fluckey("import.pgp(file='keyFile.gpg')",NULL,":STDERR",NULL);

Parameters

in	*cmd*	Command string with KEY parameter (if starts with '=', a parameter file containing the command string must follow)
in	*pro*	Property string with KEY parameter (can be empty or NULL)
in	*out*	Filename for printouts (if NULL or empty then is no printout produced)
in	*trc*	Filename for tracing (if NULL or empty then no trace is written)

Returns

signed integer with condition code for the command

4.2.2.17 flucrsn()

```
int flucrsn (
            void  )
```

Get reason code.

This function return the reason code (internal FLAM error code) if a call to a FLUC command fails.

Returns

signed integer with internal FLAM reason code

4.2.2.18 flucmsg()

```
const char* flucmsg (
            const int rsn )
```

Get a message for a reason code.

Parameters

in	*rsn*	Signed integer containing the reason code

Returns

a pointer to a static area with a zero-terminated string containing the error message for that reason code.

4.2.2.19 fluchlp()

```
void fluchlp (
            const int what,
            const int depth,
            const char * path,
            const char * out )
```

Returns help information about the command string.

This function can be used to get all help messages or man pages for each parameter in the command string. If depth 0 or greater then 9 aliases are provided. In all other cases aliases are suppressed. Please use flucsyn() to see possible aliases for the same argument.

Parameters

in	*what*	Integer constant for the corresponding command string (FLUC_CONV or FLUC_XCNV or ...)
in	*depth*	Number of levels to display (0-Man page, 1-One Level, 2-Two Level, ..., <9-All)
in	*path*	Path (e.g. conv.read.text.ccsid) to limit help to the (sub)tree of a certain object
in	*out*	Filename for printouts (if NULL or empty then stderr is used)

4.2.2.20 flucsyn()

```
void flucsyn (
            const int what,
```

```
            const int depth,
            const char * path,
            const char * out )
```

Return syntax information about the command string.

This function can be used to determine the complete syntax of the command strings.

Parameters

in	*what*	Integer constant for the corresponding command string (FLUC_CONV or FLUC_XCNV or ...)
in	*depth*	Number of levels to display (0-Man page, 1-One Level, 2-Two Level, ..., <9-All)
in	*path*	Path (e.g. conv.read.text.ccsid) to limit syntax to the (sub)tree of a certain object
in	*out*	Filename for printouts (if NULL or empty then stderr is used)

4.2.2.21 flucdoc()

```
void flucdoc (
            const int what,
            const char * path,
            const char * out )
```

Return documentation.

This function can be used to determine the complete documentation of the corresponding command.

Parameters

in	*what*	Integer constant for the corresponding command string (FLUC_CONV or FLUC_XCNV or ...)
in	*path*	Path (e.g. conv.read.text.ccsid) to limit docu to the (sub)tree of a certain object
in	*out*	Filename for printouts (if NULL or empty then stderr is used)

4.2.2.22 flucpro()

```
void flucpro (
            const int what,
            const char * ipro,
            const char * prop,
            const char * opro,
            const char * out )
```

Manage properties.

This function can be used to manage a property file for one of the supported commands. You can generate a property file if 'ipro' NULL and 'opro' defined. You can printout the properties if 'ipro' defined and 'opro' is NULL. You can validate and copy property files if 'ipro' and 'opro' defined. You can adjust a property file if 'ipro' defined, 'prop' are used and 'opro' set.

Parameters

in	*what*	Integer constant for the corresponding command (FLUC_CONV or FLUC_XCNV or ...)
in	*ipro*	Property file to read (if NULL or empty then then no property file is read)
in	*prop*	Property string to manipulate the current properties (if NULL or empty then no update is done)
in	*opro*	Property file to write (if NULL or empty then stdout is used)
in	*out*	Filename for printouts (if NULL or empty then is no printout produced)

Chapter 5

File Documentation

5.1 FLUCUP.h File Reference

Specification for the FLUC subprogram interface.

Macros

- #define FLUC_INFO 1

  INFO command.
- #define FLUC_CONV 2

  CONV command.
- #define FLUC_XCNV 3

  XCNV command.
- #define FLUC_ICNV 4

  ICNV command.
- #define FLUC_FLAM 5

  FLAM command.
- #define FLUC_XCHK 6

  XCHK command.
- #define FLUC_HASH 7

  HASH command.
- #define FLUC_UTIL 8

  UTIL command.
- #define FLUC_DIFF 9

  DIFF command.
- #define FLUC_KEY 10

  KEY command.

Functions

- const char * flucvsn (void)

  Version.
- const char * flucabo (void)

  About.
- const char * fluclic (void)

  Get license text.
- const char * flucown (const char *own)

  Set/Get owner id.
- const char * flucpgm (const char *pgm)

  Set/Get program name.
- int flucenv (const int sys, const int std, const char *env)

  Load the FLAM environment.
- int flucinfo (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC INFO - Provides various information.
- int fluchash (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC HASH - Simplified Hash calculation.
- int flucconv (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC CONV - Simplified data conversion.
- int flucxcnv (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC XCNV - Extended data conversion.
- int flucxchk (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC XCHK - Extended data evalation.
- int flucicnv (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC ICNV - Character conversion like ICONV.
- int flucflam (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC FLAM - Support of FLAM4 command in FLUC.
- int flucutil (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC UTIL - Support of several utility function.
- int flucdiff (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC DIFF - Compares two data sources.
- int fluckey (const char *cmd, const char *pro, const char *out, const char *trc)

  FLUC KEY - Key management functions.
- int flucrsn (void)

  Get reason code.
- const char * flucmsg (const int rsn)

  Get a message for a reason code.
- void fluchlp (const int what, const int depth, const char *path, const char *out)

  Returns help information about the command string.
- void flucsyn (const int what, const int depth, const char *path, const char *out)

  Return syntax information about the command string.
- void flucdoc (const int what, const char *path, const char *out)

  Return documentation.
- void flucpro (const int what, const char *ipro, const char *prop, const char *opro, const char *out)

  Manage properties.

5.1.1 Detailed Description

Specification for the FLUC subprogram interface.

Author

limes datentechnik gmbh

Date

18.02.2015

Copyright

Index