cmdname: Command Name Resolver Library
Version 1

CML00076-01

Code Magus Limited (England reg. no. 4024745)
Number 6, 69 Woodstock Road
Oxford, OX2 6EY, United Kingdom
www.codemagus.com

December 15, 2020

Contents

1 Introduction

A command name file is a regular text file that holds commands to be executed by any Code Magus software that exposes a command interface to users in order to configure and/or manipulate the processing of that software. Using command name files allows users to write commands in a file so that they can all be executed as a group and possibly more than once. They are often used by servers at start up in order to configure the processing environment.

This document explains the rules of command name validation and then the interface to the Code Magus Limited `cmdname` library.

The rules (see section 2 on page 4) are documented for the benefit of the users of any Code Magus software that invokes the library services so that they can correctly name and place any command files in the file system of the machine running the Code Magus software.

The library application programming interface (API) (see section 3 on page 7) is documented for the benefit of the developers using the library services. Functions are supplied to validate and resolve a given command file name and path information into a fully qualified name (FQN), open the FQN for reading, read a record and close the FQN.

2 Command File Validation

Validation proceeds in two stages; first the path information (including each directory) is validated and then (possibly later and more than once) a command file name is validated using the stored path information.

For all validation note that an *Identifier* restricts both directory node names and file names. It is case sensitive and starts with a letter which can be followed by any number of letters, digits or the under-score character.

The following sub-sections explain the validation process.

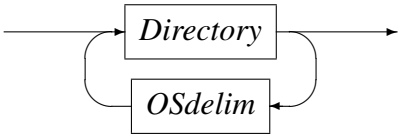
2.1 Path Information Validation

The path information only needs to be validated once, usually during initialisation of a utility or program requiring it. It must be supplied either by the program performing the validation or from the value of the environment variable `CODEMAGUS_COMMAND_PATH`. If no path information is supplied an error is returned.

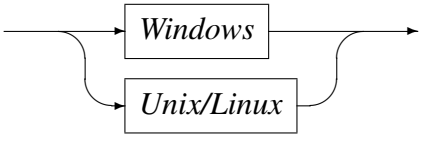
2.1.1 Path Information

The path information is a delimited list of directories. The delimiter depends on the operating system (OS) on which the software is running.

Path Information

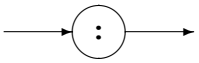


OSdelim



Windows

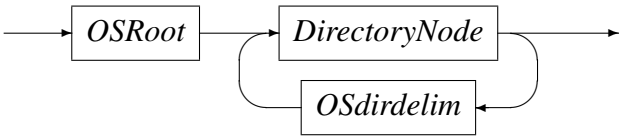
Unix/Linux



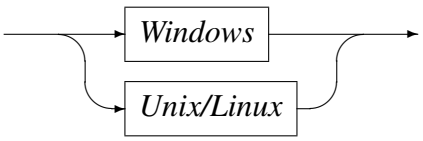
2.1.2 Directory Names

Each directory in the path information must be an absolute directory name, which means that it must start, for example, with '`c:\`' or '`d:/`' on Windows platforms and '/' on Linux and Unix platforms. Each directory must also exist within the file system. If either of these conditions are not met an error is returned.

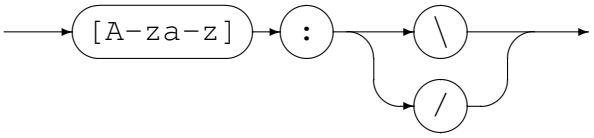
Directory



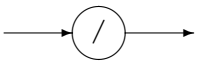
OSRoot



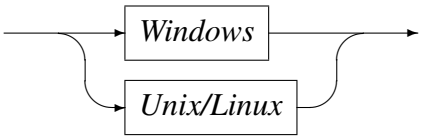
Windows



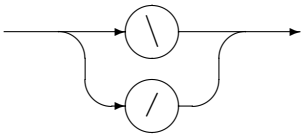
Unix/Linux



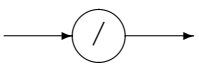
OSdirdelim



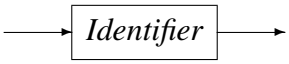
Windows



Unix/Linux



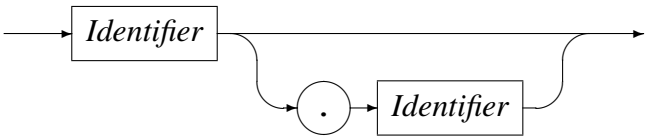
DirectoryNode



A directory node must be made up of an *Identifier*.

2.2 Command File Name Validation

A command name is a file name and must conform to the following:



The file name is searched for in each directory in the path information starting with the first directory and continuing in the order given. An error is returned if the file does not conform to the above rule or is not found in any of the directories.

3 API Reference

The API interface is defined in the header `cmdname.h`, see section A on page 13.

3.1 `cmdname_error()`

3.1.1 Synopsis

```
char *cmdname_error(void);
```

3.1.2 Description

This function returns a NULL terminated string describing the last error encountered by any call to the library.

3.1.3 Parameters

There are no parameters to this function.

3.1.4 Return Value

Returns a NULL terminated string describing the last error message encountered by any call to the library.

3.2 `cmdname_open()`

3.2.1 Synopsis

```
int cmdname_open(char *path);
```

3.2.2 Description

This function initialises the cmdname instance and validates the path information that is used for resolving the fully qualified command file names as per section 2 on page 4. The parameter path, if not NULL takes precedence over the value of the environment variable. It is an error if any of the rules for path information fails.

3.2.3 Parameters

- `path`
  This parameter contains the path information.

3.2.4 Return Value

On success zero is returned, otherwise -1 is returned and the error message can be retrieved, using the function `cmdname_error()`.

3.3 `cmdname_close()`

3.3.1 Synopsis

```
void cmdname_close(void);
```

3.3.2 Description

This function frees all resources used by the cmdname library.

3.3.3 Parameters

There are no parameters to this function.

3.3.4 Return Value

There are no return value from this function.

3.4 `cmdname_resolve()`

3.4.1 Synopsis

```
char *cmdname_resolve(char *fname);
```

3.4.2 Description

This function returns a validated fully qualified command file name for the requested command file name as per section 2 on page 4.

3.4.3 Parameters

- `fname`
  This parameter is the file name to resolve. It must adhere to the restrictions as described in section 2 on page 4.

3.4.4 Return Value

On success the fully qualified file name is returned, otherwise NULL is returned and the error message can be retrieved, using the function `cmdname_error()`.

3.5 `cmdname_open_file()`

3.5.1 Synopsis

```
cmdname_stream_t *cmdname_open_file(char *fname);
```

3.5.2 Description

This function initially calls `cmdname_resolve()` to resolve the fully qualified command file name and then opens the command file.

3.5.3 Parameters

- `fname`

  This parameter is the file name to resolve. It must adhere to the restrictions as described in section 2 on page 4.

3.5.4 Return Value

On success a `cmdname_stream_t` structure is returned which is to be used for reading and closing the command file using the relevant functions. On error NULL is returned and the error message can be retrieved, using the function `cmdname_error()`.

3.6 cmdname_open_output()

3.6.1 Synopsis

```
cmdname_stream_t *cmdname_open_output(char *fname);
```

3.6.2 Description

This function resolves the supplied file name by using the first path in the list of paths supplied on the `cmdname_open()` function and then opens the file for writing.

3.6.3 Parameters

- `fname`

  This parameter is the file name to resolve. It must adhere to the restrictions as described in section 2 on page 4.

3.6.4 Return Value

On success a `cmdname_stream_t` structure is returned which is to be used for writing and closing the command file using the relevant functions. On error NULL is returned and the error message can be retrieved, using the function `cmdname_error()`.

3.7 cmdname_close_file()

3.7.1 Synopsis

```
int cmdname_close_file(cmdname_stream_t *stream);
```

3.7.2 Description

This function closes the stream indicated by the stream handle supplied and frees all resources associated with the stream object.

3.7.3 Parameters

- `stream`

  Stream handle returned by the open of the file.

3.7.4 Return Value

On success zero is returned, otherwise -1 is returned and the error message can be retrieved, using the function `cmdname_error()`.

3.8 cmdname_read_file()

3.8.1 Synopsis

```
char *cmdname_read_file(cmdname_stream_t *stream, int *recno, int len,
      char *buf);
```

3.8.2 Description

This function reads the next text record from the input stream into the buffer `buf` for a maximum of `len` bytes. The text line delimiters ('\n' and/or '\r') are stripped from the end of the buffer and if this results in an empty buffer it will be ignored and the next record is processed automatically. If the last character (ignoring text line delimiters and spaces) of the currently read record in the buffer is a back slash ('\'), the next record will be appended to the buffer to form a single record. The buffer will always be NULL terminated.

3.8.3 Parameters

- `stream`

  Stream handle returned by the open of the file.

- `recno`

  Updated by this function to the record number of the currently read record.

- `len`

  Size of the supplied buffer.

- `buf`

  Buffer for the record to be read.

3.8.4 Return Value

If the end of the stream is encountered during processing then NULL is returned, otherwise a pointer to the buffer is returned.

3.9 cmdname_write_file()

3.9.1 Synopsis

```
int cmdname_write_file(cmdname_stream_t *stream,char *buf);
```

3.9.2 Description

This function writes a text record to the output stream from the buffer `buf`. appending the appropriate line delimiters.

3.9.3 Parameters

- `stream`

  Stream handle returned by the open of the file.

- `buf`

  Buffer for the writing.

3.9.4 Return Value

On success the number of bytes written is returned, otherwise -1 is returned and the error message can be retrieved, using the function `cmdname_error()`.

A Header file cmdname.h

```
#ifndef CMDNAME_H
#define CMDNAME_H
  /* File cmdname.h
   *
   * Code Magus Limited command name library. This library resolves a path and a
   * command file name into a fully qualified name and, validates the file (e.g.
   * it exists as a regular file) and allows the caller to read the contents of
   * the file.
   * Functions are supplied to resolve the FQN, open the file for reading, read
   * a record and close the file. Both a PATH and the file name must conform to
   * a pattern defined by this library.
   *
   * Copyright (c) 2010 Code Magus Limited. All rights reserved.
   *
   */

  /* $Author: janvlok $
   * $Date: 2014/11/05 12:10:56 $
   * $Id: cmdname.h,v 1.9 2014/11/05 12:10:56 janvlok Exp $
   * $Name:  $
   * $Revision: 1.9 $
   * $State: Exp $
   *
   * $Log: cmdname.h,v $
   * Revision 1.9  2014/11/05 12:10:56  janvlok
   * Alow spaces in path name for windows
   *
   * Revision 1.8  2012/03/19 14:27:33  janvlok
   * Implemented write of a command file
   *
   * Revision 1.7  2011/06/21 10:54:26  hayward
   * Allow consecutive / or \ in the file names when checking
   * against the regex as these are valid in normal unix and
   * windows file system processing and allows a user to not
   * worry about wether the final slash on a directory name
   * must or must not be supplied.
   *
   * Revision 1.6  2010/12/22 11:16:43  hayward
   * Make '/' or '\' compulsory after the drive letter (c:) on Windows.
   *
   * Revision 1.5  2010/12/07 10:16:44  janvlok
   * Prefixed IDENTIFIER - clashesh with callers
   *
   * Revision 1.4  2010/12/03 10:28:45  hayward
   * Allow one or no suffix (including the dot).
   *
   * Revision 1.3  2010/12/02 11:57:41  hayward
   * Improve documentation.
   *
   * Revision 1.2  2010/11/30 17:09:35  janvlok
```

```
   * Working on Windows
   *
   * Revision 1.1.1.1  2010/11/30 14:56:42  janvlok
   * Take on
   *
   */
static char *cvs_cmdname_h =
   "$Id: cmdname.h,v 1.9 2014/11/05 12:10:56 janvlok Exp $";

#ifdef __cplusplus
extern "C"
{
#endif   /* C++ */

  /* Defines and constants:
   */

#define CMDPATH "CODEMAGUS_COMMAND_PATH"

  /* Regular expressions for path and file name validation::
   * Unix PATH       ^/\(\([A-Za-z][A-Za-z0-9_]*\)/\?\)*$
   * Win Path        ^[A-Za-z]:[/\]\(\([A-Za-z][A-Za-z0-9_]*\)[/\]\?\)*$
   * File name        ^\([A-Za-z][A-Za-z0-9_]*\)\.\([A-Za-z][A-Za-z0-9_]*\)$
   */

#define CM_IDENTIFIER "[A-Za-z][A-Za-z0-9_]*"
#ifdef WIN32
#define CM_PIDENTIFIER "[A-Za-z][A-Za-z0-9_ ]*"
#define PREGEX "^[A-Za-z]:[/\\]*\\(\\(" CM_PIDENTIFIER "\\)[/\\]*\\)*$"
#else
#define PREGEX "^/*\\(\\(" CM_IDENTIFIER "\\)/*\\)*$"
#endif

#define FNREGEX "^\\(" CM_IDENTIFIER "\\)\\(\\.\\(" CM_IDENTIFIER "\\)\\)\\?$"

  /* Structures:
   */

typedef struct cmdname_stream cmdname_stream_t;  /* command file stream */

  /* Exposed functions:
   */

  /* Function cmdname_error() will return a NULL terminated string describing
   * the last error message encountered by any call to the library.
   */

char *cmdname_error(void);

  /* Function cmdname_open() initialises the cmdname instance and validates the
   * path information that is used for resolving the fully qualified command
   * file names.
```

```
   * Default path information for the command files may be specified in the
   * environment variable CODEMAGUS_COMMAND_PATH. The parameter path, if not
   * NULL takes precedence over the value of the environment variable. It is an
   * error if both the environment variable and the path parameter are empty.
   * Multiple paths may be specified, delimited by ':' (';' for Windows).
   * Note the paths must be absolute, and can not be relative.
   * On success zero is returned, otherwise -1 is returned and the error
   * message can be retrieved, using the function cmdname_error().
   */

int cmdname_open(char *path);

  /* Function cmdname_close() frees all resources used by the cmdname library.
   * There is no return value from this function.
   */

void cmdname_close(void);

  /* Function cmdname_resolve() returns the validated fully qualified file name
   * for the requested file name. The file is searched for in the list of paths
   * supplied on the cmdname_open() function starting with the first one and
   * working down the list. Validation of the fully qualified name ensures that
   * the file exists as a regular file.
   * Note the file name must not include the leading directory components.
   * On error NULL is returned and the error message can be retrieved, using
   * the function cmdname_error().
   */

char *cmdname_resolve(char *fname);

  /* Function cmdname_open_file() initially calls cmdname_resolve() to resolve
   * the fully qualified command file name and then opens the command file,
   * returning a cmdname_stream_t structure which is to be used for reading and
   * closing the command file using the relevant functions.
   * Note the file name must not include the leading directory components.
   * On error NULL is returned and the error message can be retrieved, using
   * the function cmdname_error().
   */

cmdname_stream_t *cmdname_open_file(char *fname);

  /* Function cmdname_open_output() resolves the supplied file name by using the
   * first path in the list of paths supplied on the cmdname_open() function and
   * then opens the file for writing
   * Returns a cmdname_stream_t structure which is to be used for writing and
   * closing the command file using the relevant functions.
   * Note the file name must not include the leading directory components.
   * On error NULL is returned and the error message can be retrieved, using
   * the function cmdname_error().
   */

cmdname_stream_t *cmdname_open_output(char *fname);
```

```
  /* Function cmdname_close_file() closes the stream indicated by the stream
   * handle supplied and frees all resources associated with the stream object.
   * On success zero is returned, otherwise -1 is returned and the error
   * message can be retrieved, using the function cmdname_error().
   */

int cmdname_close_file(cmdname_stream_t *stream);

  /* Function cmdname_read_file() reads the next text record from the input
   * stream into the buffer 'buf' for a maximum of 'len' bytes.
   * If the end of the stream is encountered during processing then the function
   * returns NULL. 'recno' is set to the record number of the currently read
   * record.
   * The text line delimiters (\n and/or \r) are stripped from the end of the
   * buffer 'buf' and if this results in an empty buffer it will be ignored and
   * the next record is processed automatically.
   * If the last character (ignoring text line delimiters and spaces) of the
   * currently read record in 'buf' is a back slash ('\'), the next record will
   * be appended to the buffer 'buf' to form a single record.
   * The buffer 'buf' will always be NULL terminated.
   */

char *cmdname_read_file(cmdname_stream_t *stream, int *recno, int len,
      char *buf);

  /* Function cmdname_write_file() writes a text record to the output
   * stream from the buffer 'buf'. Note the appropiate line delimeters will
   * be appended to the buffer before writing.
   * On success zero is returned, otherwise -1 is returned and the error
   * message can be retrieved, using the function cmdname_error().
   */

int cmdname_write_file(cmdname_stream_t *stream,char *buf);

#ifdef __cplusplus
}
#endif   /* C++ */

#endif /* CMDNAME_H */
```