libQuickFT

Library for remote file operations.
Implementation of the QuickFT protocol.

Reference Manual

May 7th, 2016

Julián Bachiller
<julianbachiller@gmail.com>

ABOUT QUICKFT

The QuickFT protocol emerged from the need to transport files to and from embedded devices on a LAN and delete files remotely. The original version was written for Windows, aimed at circumventing the operating system's network file operations capabilities and implement its own.

Both the QuickFT protocol and the Linux port should be considered an excersise in protocol design and sockets programming in C language.

The library implementation allows programs to set up a server or a client very easily, asi shown in the program examples bundled with the library.

The server is designed in such a way that allows serving multiple simultaneous requests from file operations, up to a hard-coded max. of 256 (which could be parametrized easily if needed).

The library supports three types of operations, which from the client point of view would be: transfer a file to the server, transfer a file from the server and delete a device from the server.

The protocol is limited to single file operations, not permitting multiple files to be affected by a single action.

THE QUICKFT PROTOCOL

Below are the technichal details of the QuickFT Protocol.

The messages are composed of a header at the beginning and a variable length payload following the header.

The structure of the header part is the following:

QUIFT_MSG=V1.0=FILE_RCV=0000000B						
72 bits	8 bits	32 bits	8 bits	64 bits	8 bits	64 bits
Protocol specifitacion	Field separator	Protocol version	Field separator	Message type	Field separator	Variable part length (hex.)

Additionaly, following the header a colon ':' character is required as a separator between the header and the variable length part of the message.

It is valid for a message not to have a variable length part, ommiting also the separator between the header and the payload, thus being composed solely of a header.

The supported message types are the following:

FILE_SND	File send operation from client to server
FILE_RCV	File receive operation from server to client
FILE_DEL	File delete operation in server machine
ACK_____	Acknowledgment message

The protocol is based synchronous request-response message interchange.

Upon receiving a request for a file operation the server may send an acknowledgment message indicating that the request is being processed.

A response is mandatory indicating the result of the operation.

For the variable length part of the request the parameters suported depend on the type of operation being requested.

The parameters always start with an equal sign '=' followed by the parameter name and a colon separating the parameter name from the value.

The following are the supported parameters for the different message types:

FILE_SND	*Request*	=path:[path including filename of the file in destination] =length:[length of content] =content:[content of the file]
	Response	=result:[operation result code]
FILE_RCV	*Request*	=filename:[name of the file to being requested to receive]
	Response	=result:[operation result code] =length:[length of content] =content:[content of the file]
FILE_DEL	*Request*	=filename:[name of the file to delete in destination]

	Response	=result:[operation result code]

The acknowledgment message type does not support any parameters, and

The 'content' parameter used to transport the file itself typically will ve provided with a value already encoded to allow the string representation of bytes.

In the current library implementation all file content values are previously compressed and then base64-encoded.

File compression/decompression and encoding/decoding is already handled by the library.

Sample messages for the supported operations are presented below.

```
File Send

Request:
QUIFT_MSG=V1.0=FILE_SND=00000038:=path:\tmp\file.bin=length:14567=content:[file content]

Response:
QUIFT_MSG=V1.0=FILE_SND=00000020:=result:[operation result code]
```

```
File Receive

Request:
QUIFT_MSG=V1.0=FILE_RCV=0000001C:=filename:\tmp\testFile.xml

Response:

QUIFT_MSG=V1.0=FILE_RCV=0000004F:=result:[operation result code]=length:[content length]=content:[file
content]

QUIFT_MSG=V1.0=FILE_RCV=00000020:=result:[operation result code]
```

```
File Delete

Request:
QUIFT_MSG=V1.0=FILE_DEL=0000001C:=filename:\tmp\testFile.xml

Response:
QUIFT_MSG=V1.0=FILE_DEL=00000020:=result:[operation result code]
```

```
Acknowledgment

QUIFT_MSG=V1.0=ACK_____=00000000
```

LIBQUICKFT LIBRARY REFERENCE

The QuickFT library implements functions for both server and client programs.

Below are the function calls for the operations supported.

Server: Initialization

```
/**
 * Initializes server for sending and receiving messages
 *
 * @param port                port to listen on
 * @param max_connections     max. connections supoerted by the server
 * @param timeout             timeout for messages, can be 0 for default
 * @param log_writer          a pointer to a callback logging function
 *                            that receives two parameters:
 *
 *                            function: name of the function in which
 *                                      logger_write is being called.
 *                            message:  body of the message.
 *
 * @return                    pointer to a SERVER_T type
 */
SERVER_T * server_initialize( int port,
                              int max_connections,
                              int timeout,
                              void (*log_writer) (const char* function, char* message) );
```

Server: Deinitialization

```
/**
 * Ends the connections and finalizes the server
 *
 * @param server              server data structure pointer by reference
 * @return                    TRUE or FALSE
 */
int server_finalize ( SERVER_T ** server );
```

Client: File Send Operation

```
/**
 * Performs a 'File Send' operation for the client
 *
 * @param remote_filename     filename in destination location
 * @param local_filename      filename in location of origin
 * @param addr                server IP
 * @param port                server port
 * @param timeout             timeout for messages, can be 0 for default
 * @param timeout_ack         timeout for ack messages, can be 0 for default
 * @param log_writer          a pointer to a callback logging function
 *                            that receives two parameters:
 *
 *                            function: name of the function in which
 *                                      logger_write is being called.
 *                            message:  body of the message.
 *
 * @return                    result code
 */
int client_file_send( char * remote_filename,
                      char * local_filename,
                      char * addr,
                      char * port,
                      int timeout,
                      int timeout_ack,
                      void (*log_writer) (const char* function, char* message) );
```

Client: File Receive Operation

```
/**
 * Performs a 'File Receive' operation for the client
```

```
 *
 * @param remote_filename     filename in location of origin
 * @param local_filename      filename in destination location
 * @param addr                server IP address
 * @param port                server Port, can be 0 for default
 * @param timeout             timeout for messages, can be 0 for default
 * @param timeout_ack         timeout for ack messages, can be 0 for default
 * @param log_writer          a pointer to a callback logging function
 *                            that receives two parameters:
 *
 *                            function: name of the function in which
 *                                      logger_write is being called.
 *                            message:  body of the message.
 *
 * @return                    result code
 */
int client_file_receive( char * remote_filename,
                         char * local_filename,
                         char * addr,
                         char * port,
                         int timeout,
                         int timeout_ack,
                         void (*log_writer) (const char* function, char* message) );
```

Client: File Delete Operation

```
/**
 * Performs a 'File Delete' operation for the client on the server
 *
 * @param remote_filename     filename in destination
 * @param addr                server IP
 * @param port                server port
 * @param timeout             timeout for messages, can be 0 for default
 * @param timeout_ack         timeout for ack messages, can be 0 for default
 * @param log_writer          a pointer to a callback logging function
 *                            that receives two parameters:
 *
 *                            function: name of the function in which
 *                                      logger_write is being called.
 *                            message:  body of the message.
 *
 * @return                    result code
 */
int client_file_delete( char * remote_filename,
                        char * addr,
                        char * port,
                        int timeout,
                        int timeout_ack,
                        void (*log_writer) (const char* function, char* message) );
```

The following macros are also defined in client.h for calling client functions:

```
#define QUICKFT_FILE_RECEIVE    client_file_receive
#define QUICKFT_FILE_SEND       client_file_send
#define QUICKFT_FILE_DELETE     client_file_delete
```

For client operations the operation result code definitions are the following.

Macro definition	Numeric Value	Description
RESULT_SUCCESS	0	Operation completed successfully
RESULT_CONNECTION_ERROR	-100	A connection error occurred
RESULT_UNDEFINED	-101	An unexpected error occurred
RESULT_CONFIG_ERROR	-102	An invalid configuration error occurred
RESULT_INVALID_REQUEST	-103	The request message is not valid
RESULT_INVALID_RESPONSE	-104	The response message is not valid
RESULT_FILE_ACCESS_ERROR	-105	The file could not be accessed
RESULT_FILE_NOT_FOUND	-106	A required file was not found during an operation
RESULT_FILE_WRITE_ERROR	-107	Error writing file
RESULT_FILE_READ_ERROR	-108	Error reading file

RESULT_FILE_COMPRESS_ERROR	-109	Error compressing file
RESULT_FILE_DECOMPRESS_ERROR	-110	Error decompressing file
RESULT_FILE_ENCODE_ERROR	-111	Error encoding file
RESULT_FILE_DECODE_ERROR	-112	Error decoding file
RESULT_FILE_DELETE_ERROR	-113	Error deleting a file
RESULT_INVALID_DESTINATION_DIRECTORY	-114	Destination directory is not valid
RESULT_COULD_NOT_CREATE_DESTINATION_DIRECTORY	-115	Destination directory could not be created