

## FX ONE : DLL Manual

### Scope

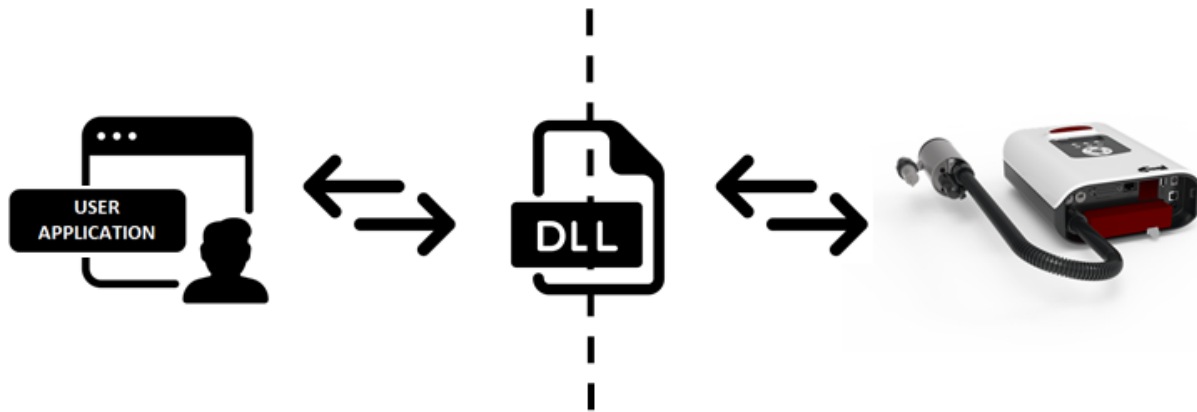
The purpose of this manual is to describe the operation of the DLL Interface for the Foenix FX ONE printers. A description of what the DLL is and how it works is provided. Example code fragments using the DLL functions are included. Two further examples describe how the DLL can be used with a database to extract information and send to the printer. Finally, a description of every function is provided at the end of this document.

### What is a DLL?

A DLL (Dynamic Link Library) is a set of software functions that can be used by a third party developer to incorporate into their own applications. The Foenix DLL described here provides a set of functions that allows a FX ONE printer to be controlled via the FXPro application or Foenix Touch Controller. The library makes the communication process to the printer much simpler for the developer as it takes care of the actual protocol used. The developer then uses a relatively simple mechanism to monitor printers, load messages and transmit data.

### How does a DLL work?

The DLL is used by the customers' application to communicate with the printer. The diagram below gives a basic overview showing how the DLL sits between the application and printer.



### Files to include

Within the third party application the following files are needed. These are supplied by Foenix Coding;

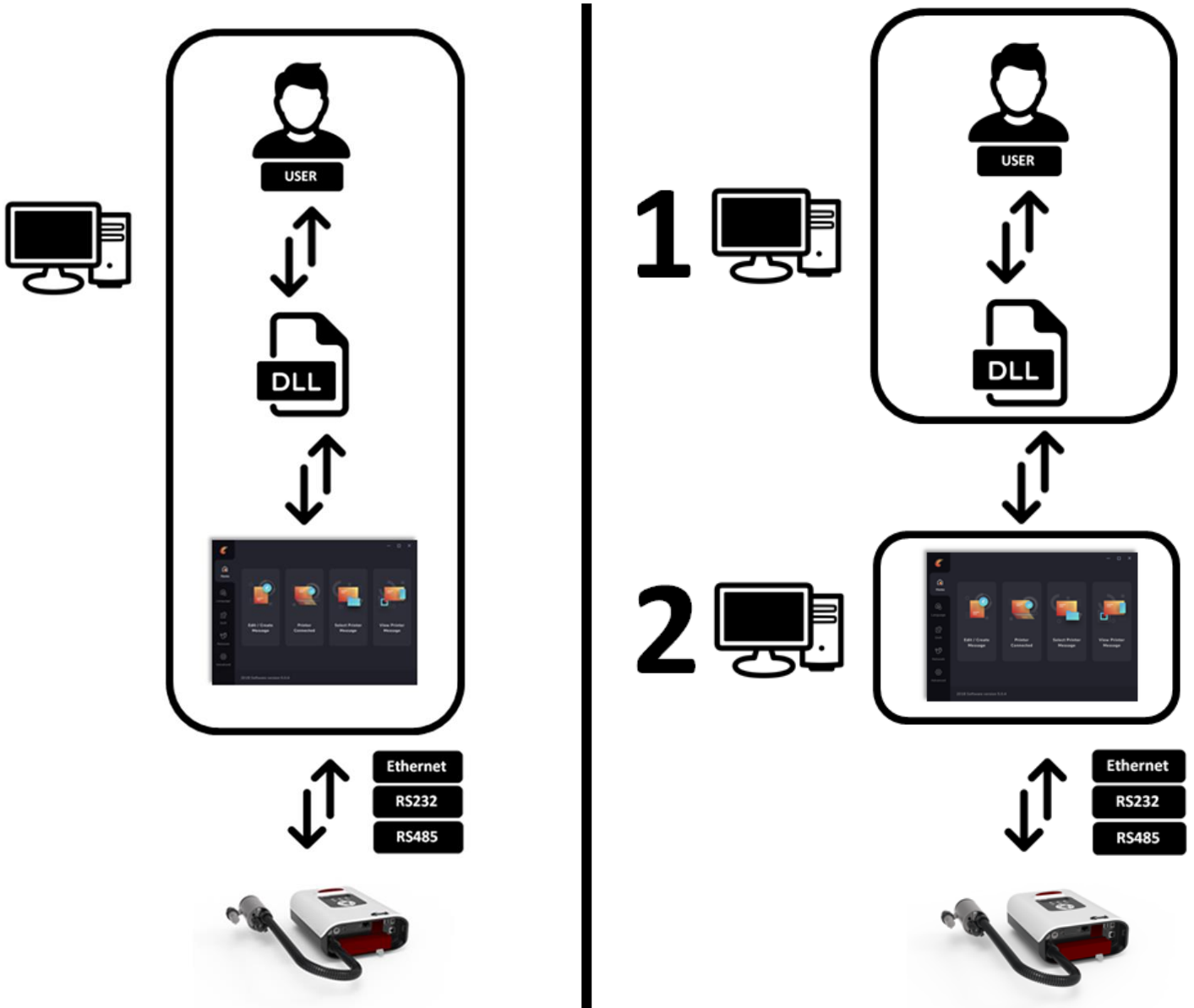
Printer DLL.h Header file that declares the functions available in the DLL

Printer DLL.lib Static library for incorporating into the third party application during building and also provides the function call interface to the actual functions

Printer DLL.dll Dynamic library that is loaded at third party application run

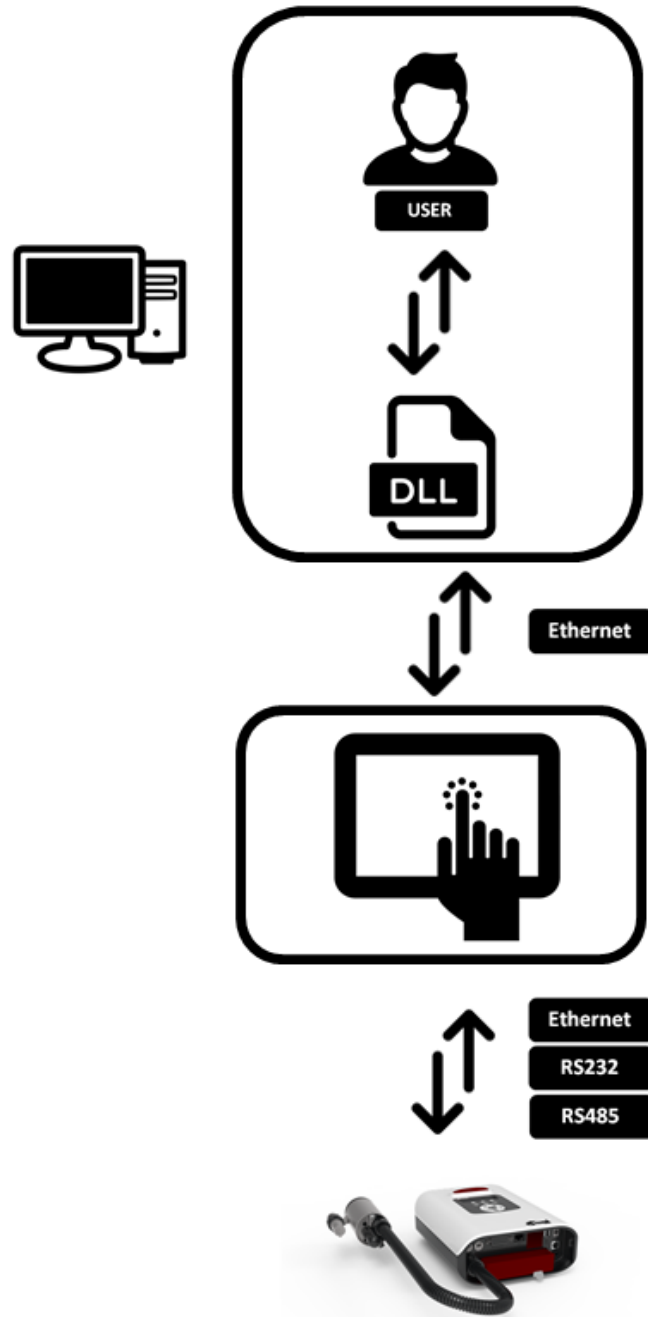
## Using the library

The third party application talks to a printer via a version of FXPro that is running on either the same PC as the application or one connected to it via Ethernet.



In this case, the PC running the FXPro application is called a **server**.

Alternatively, the third party application running on a PC can use the DLL to communicate with a printer via a Foenix Touch Controller over Ethernet.

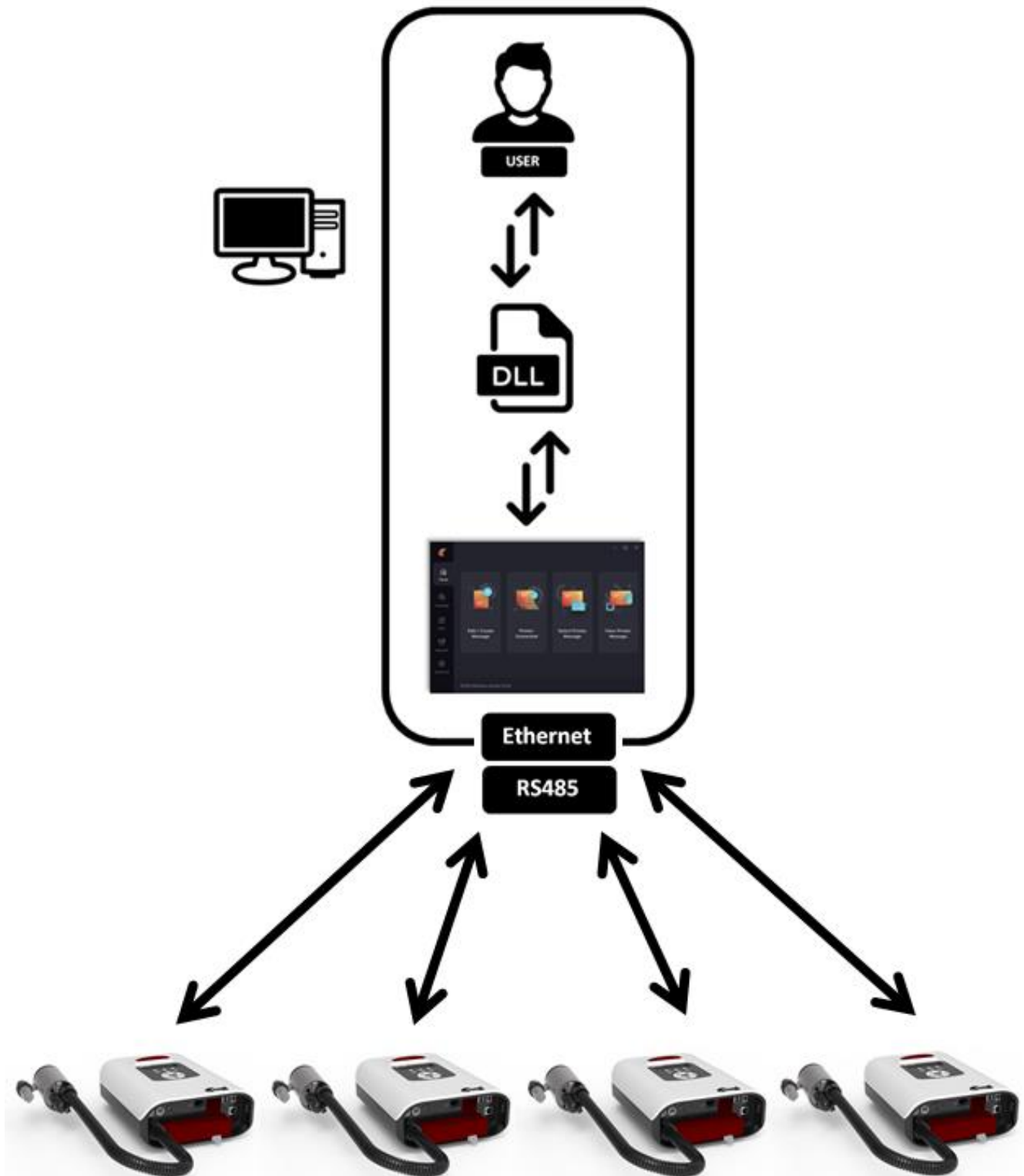


In this case, the Foenix Touch Controller is called a **server**.

The library can be reused for multiple servers each connected to its own network of printers.

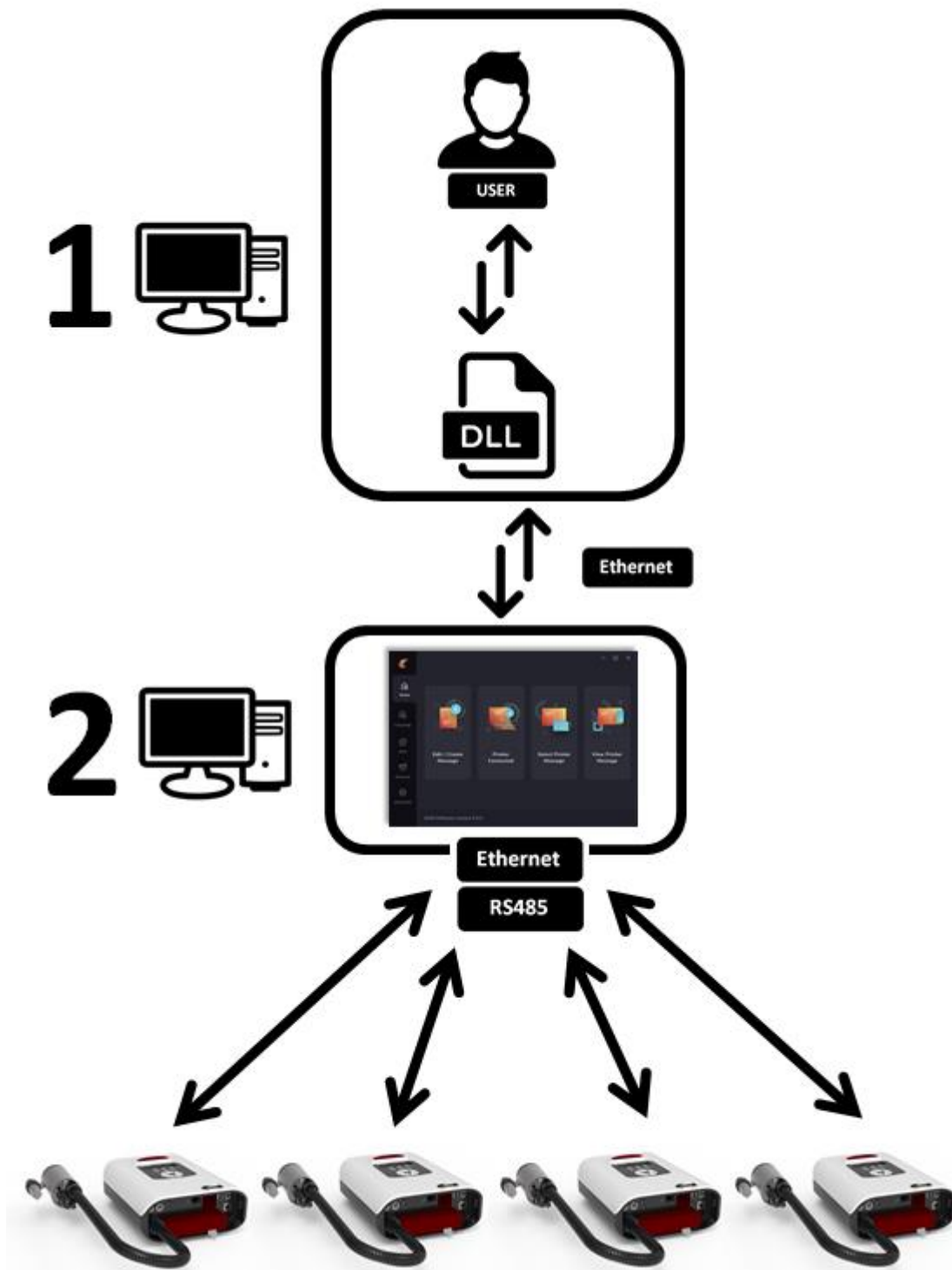
*Example Network Installation 1*

This setup has the user application and the FXPro application running on the same PC. Ethernet or USB to RS485 converter is used to connect between the PC and printers.



*Example Network Installation 2*

This setup has the user application and the FXPro application running on separate PCs. Ethernet or USB to RS485 converter is used to connect between the PC and printers.

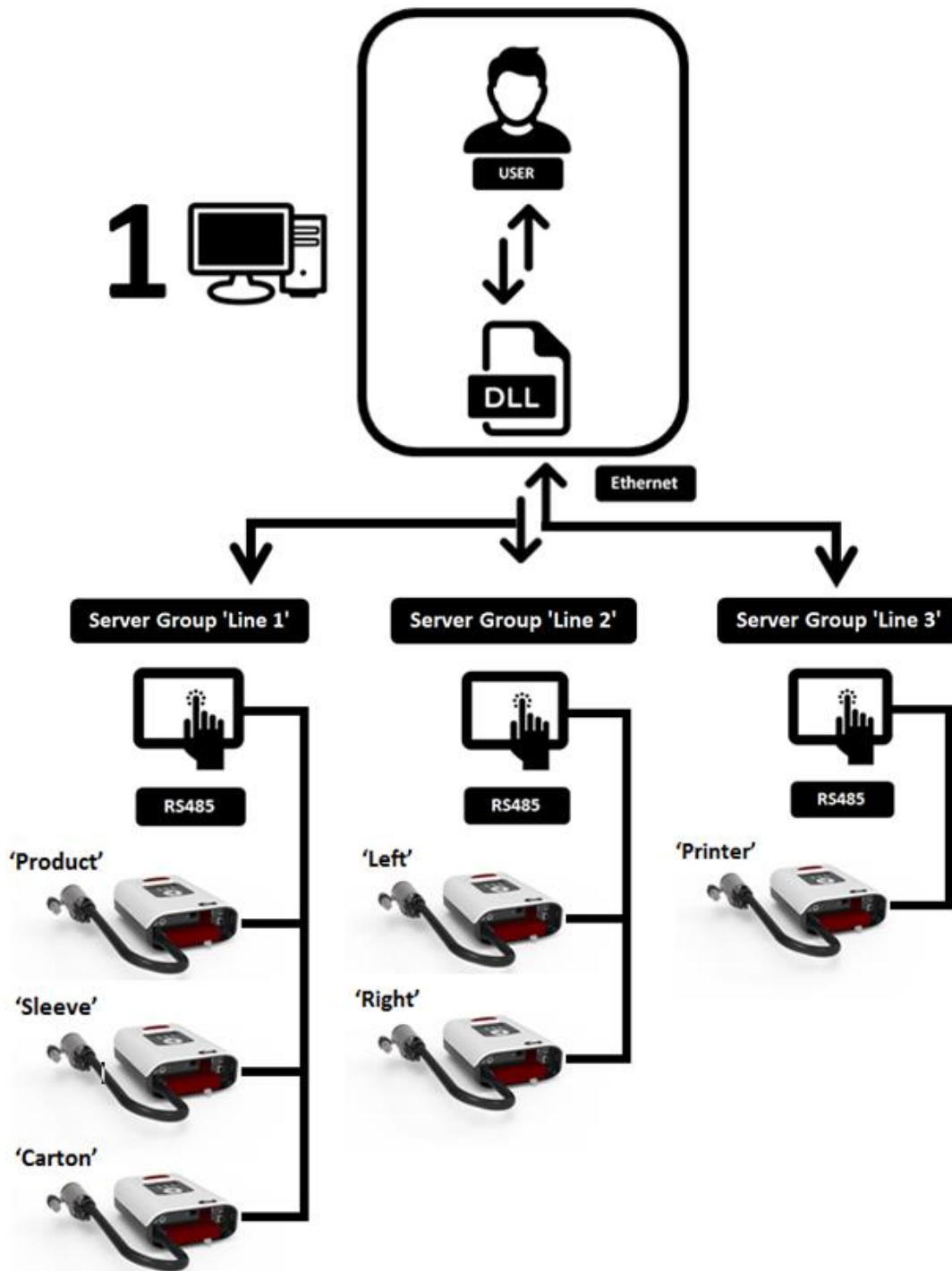


PC 1 could be in an office and PC 2 a low cost netbook located near to the production line.

**Note :** PC2 could be replaced by a Foenix touch controller.

### Example Network Installation 3

This setup has the user application running on a main PC and several touch controllers are used to create 'groups' of printers. Each touch controller is acting as a server. The DLL can be used to identify these groups and switch between them. The example below uses the touch controllers for three distinct groups 'Line 1', 'Line 2', and 'Line 3'.



- 1 Connect to a server e.g. 'Line 1'.
- 2 Select printer within that server e.g. 'Carton'.
- 3 Perform action on that printer e.g. poll status or change message.
- 4 Select another printer within that server (back to 2) or change to a new server (back to 1).

## Basic Test code

There are two example applications available complete with source code. These test different aspects of the DLL and can be used as a template for third party applications.

DLLTest.exe  
PrinterDLL.dll  
PrinterDLL.lib

DLLTest is a DOS based program. It will run from within Windows in a separate box.

Connect a Foenix Touch Controller on the network using an Ethernet cable or run FXPro software on this PC or any other PC connected to the network.

Double click on 'DLLTest.exe' and allow the program to run. After a short time searching for servers (FXPro or Touch Controllers), the test program will begin to get and send information using the DLL functions. In the third party application, the user is free to choose which functions they want to use. An example screen is shown below.

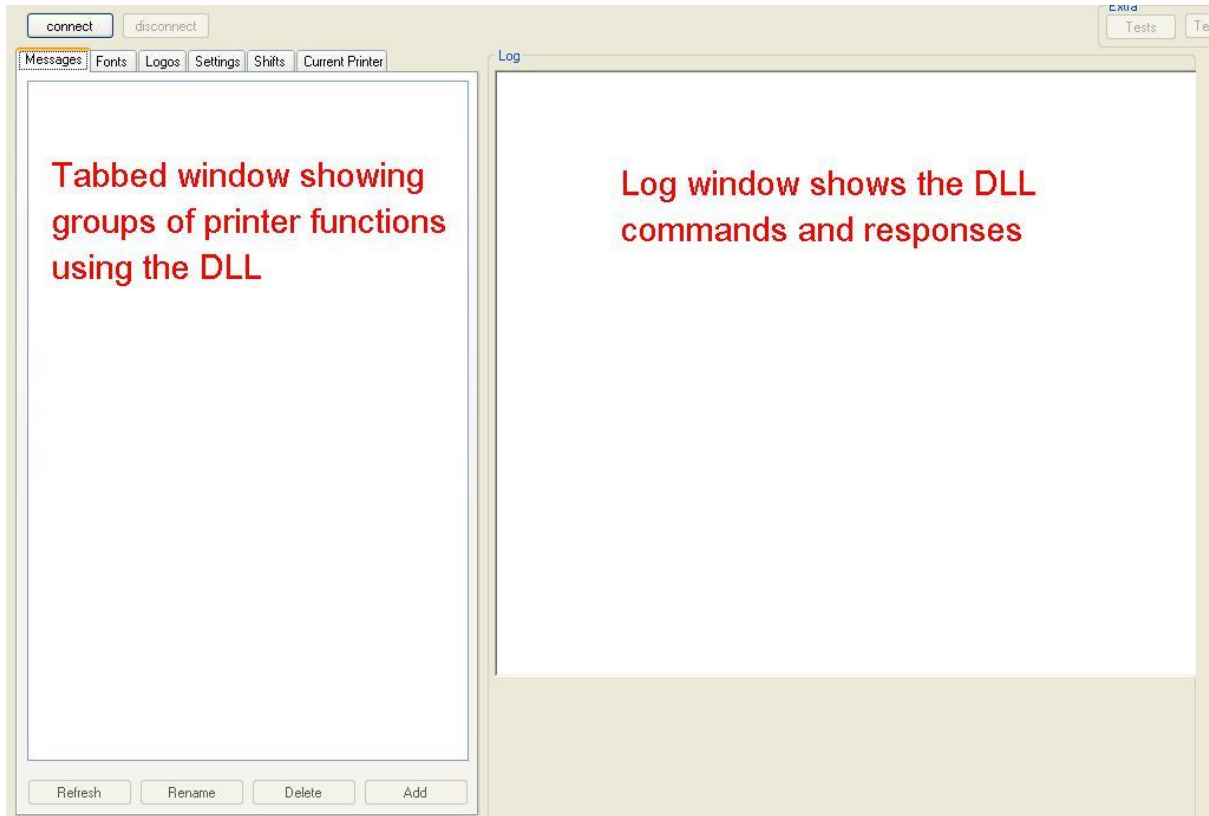
```
waiting until initial servers are found
Checking...
Checking...
found 1 servers
Connecting to server 0 : CAROLXPPC <PC>
Sending Send Message To Printer
-----
Sending Set Field Data
-----
Sending Get Field Data
-----
Sending Set Shift
-----
Sending Get Shift Settings
-----
start of day: 01:02
day 0 shift 0, 00:09, letter A
day 0 shift 1, 01:11, letter B
day 0 shift 2, 02:12, letter C
day 0 shift 3, 03:13, letter D
day 1 shift 0, 04:14, letter E
day 1 shift 1, 05:15, letter F
day 1 shift 2, 06:16, letter G
day 1 shift 3, 07:17, letter H
day 2 shift 0, 08:18, letter I
day 2 shift 1, 09:19, letter J
day 2 shift 2, 10:20, letter K
day 2 shift 3, 11:21, letter L
day 3 shift 0, 12:22, letter M
day 3 shift 1, 13:23, letter N
day 3 shift 2, 14:24, letter O
day 3 shift 3, 15:25, letter P
day 4 shift 0, 16:26, letter Q
day 4 shift 1, 17:27, letter R
day 4 shift 2, 18:28, letter S
day 4 shift 3, 19:29, letter T
day 5 shift 0, 20:30, letter U
day 5 shift 1, 21:31, letter U
day 5 shift 2, 22:32, letter W
day 5 shift 3, 23:33, letter X
day 6 shift 0, 00:34, letter Y
day 6 shift 1, 01:35, letter Z
day 6 shift 2, 02:36, letter 1
day 6 shift 3, 03:37, letter 2
Sending Set Shift
-----
```

The C++ source code is freely distributed. A more comprehensive test application is also provided.

## Functional Test Code

This is a small application that allows the developer to explore the library functions. It is provided as is with no warranty. The source code is also provided which can be modified or used as a basis for third party applications.

To launch the executable application double click on foenixDLLNET.exe. The following window will be displayed.



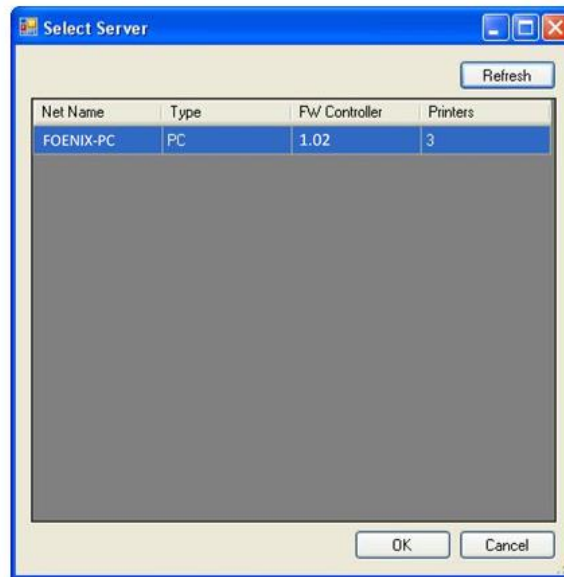
To connect to a Touch controller or PC running FXPro, click on .

The application will use the DLL to search for local and remote servers (FXPro) running on PCs and controllers over Ethernet. The log window will show the results of the search as each unit is found.





A summary of the devices found is then displayed which allows the user to select which unit to interrogate further. The summary below shows a single server has been located running on a PC. The FXPro version on this PC is 1.02 and 3 printers are connected to the server.



Once a unit is selected, the program obtains all the settings, message, font and logo names from the device. The log window shows the commands being sent and received. An example is shown below:


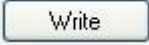
```

Connection to Linker... OK
***** refreshList *****
found 1 servers
0 TCP_GetPrinterType[0]: type:FX01E, name: Touch
0 TCP_GetPrinterFirmware[0]: FX01E.102
0 TCP_GetControllerFirmware[0]: 0.8
*****
TCP_GetPrinterStatus: status: PrinterOFF num: 0 err: 0
TCP_GetCOMPortSetting:
  baud:br115200
  Data:Data8bits8
  Parity:ParityNone
  Stop:Stop8bits1
TCP_GetPrinterSettings:
  Continuous:True
  Direction:DirLeft
  Encoder:True
  InkSaving:True
  InkType:InkTypeSolvent
  Orientation:OrientationInverted
  ShutterType:ShutterTypeNone
  Name:Touch
  Spit:235
  Tickle:120
  Close:0
  Oper:0
  Time:9
TCP_GetLanguage: 9
TCP_GetCalendarType: Gregorian
TCP_GetCartridgeCost: 0.0
TCP_GetScreenBrightness: 100
TCP_GetBeeper: False
TCP_GetDHCPState: DHCP
TCP_GetIPAddress: 1392552128
TCP_GetSubnetMask: 16777215
TCP_GetShiftSettings:
startHour :0
startMinute:0
  in 0/0/011
  
```

The output from the printer is split into a set of tabbed screens.



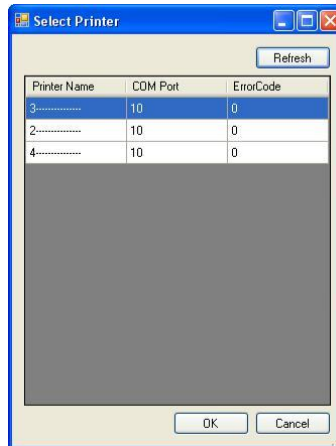
- Messages shows a list of messages on the unit. A message can be selected for printing, renamed, deleted or a new message added remotely.
- Fonts shows a list of fonts on the unit. A font can be deleted or a new one added.
- Logos shows a list of logos on the unit. A logo can be deleted or a new one added.
- Settings shows the basic settings such as message configuration options and the com port setting used for external communication.
- Shifts shows the start of day. In addition, four shift patterns per day can be defined.
- Current Printer shows the settings and message for the currently selected printer.

The contents of these tabs are automatically populated when the touch controller or PC software that has previously been identified is selected. Any changes to the items within the tabs that have been set directly on the connected unit can be updated within the tabs by pressing . In addition, pressing  will send any changes to the printer. In this way, it is possible to make multiple changes and send the settings over in one go.

### Selecting Printer

Printer Name:

If there are multiple printers on a network, pressing  will allow the current one to be changed.



Select the name of the printer to select.

The program will disconnect from the current printer and select the new one. Click on  to obtain the new parameters for the printer.

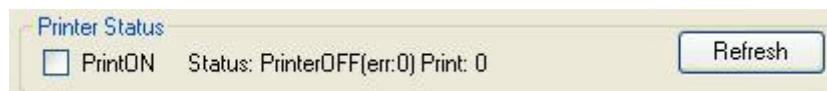
### Changing Message To Print On Printer

Name:

The currently selected message to print can be changed by pressing . Select the new message and click .

### Printer Status

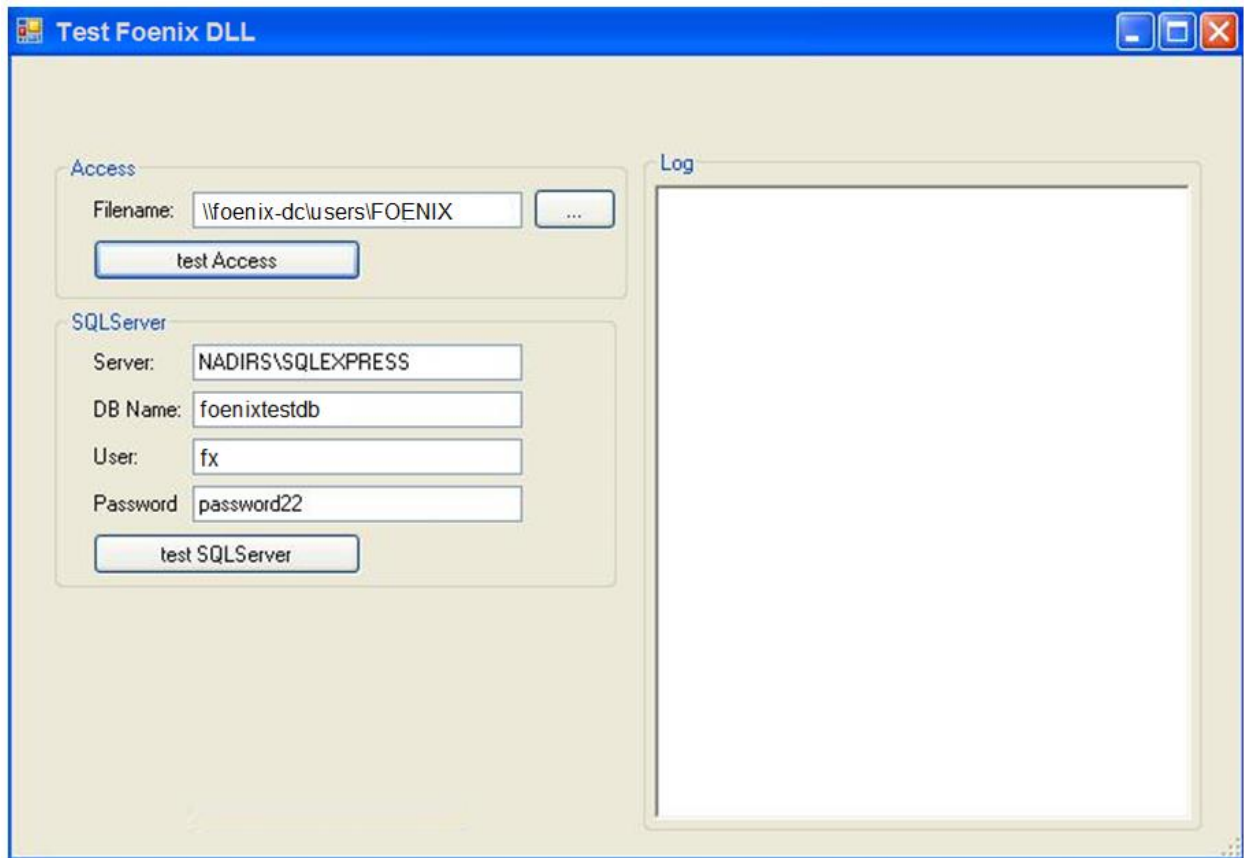
The current printer status is displayed below the printer name.



Enabling or disabling printON check box will turn printing on or off. The current print status can be retrieved using the  button. Note: print status is retrieved from the printer back to the controller or FXPro application on a periodic basis of about 5 seconds.

## Database Integration

An example Access database is included which allows data to be extracted. A test message is loaded into the printer and the field data populated from the database.



This demonstration uses;

test.mdb      Access database  
testdb.svn    message file with a field

Click on the Test Access button to begin the process.

The program opens the Access database and performs a query operation to read content from the database.

```
*****onBtnTestAccess*****  
*****  
Database Opened  
Execute query SELECT Col2 FROM Table1;  
Example 1  
Example 2  
Example 3  
Example 4
```

The program then deletes and then loads the 'testdb.svn' message to the printer server. This step is not always necessary but is included here in case there is a previous message called 'testdb'.

```
TCP_DeleteMessage 'testdb':  
TCP_AddMessage 'testdb':  
TCP_LoadMessage 'testdb':  
TCP_PrintMessage:  
TCP_GetCurrentMessageName: testdb
```

The program then determines if there are any fields or editable counters in the message. This message only has fields so the counter response comes back negative.

```
TCP_GetFieldNamesInCurrentMessage:  
names: System.Collections.Generic.List`1[System.String]  
TCP_GetCounterListInMessage (NOTOK): data:
```

The program then retrieves the current field data. In this case the field is blank.

```
TCP_GetFieldDataInCurrentMessage [TESTDB]:  
fieldData?????????  
len 10  
match 1  
type 2
```

The program populates the field and re-reads the data. The field now has the new information.

```
TCP_SetFieldDataInMessage [TESTDB]: data:Example 1  
TCP_GetFieldDataInCurrentMessage [TESTDB]:  
fieldDataExample 1  
len 10  
match 1  
type 2
```

The SQL server option requires a SQL database to be configured, server running and username/password assigned to allow access. The source code associated with the 'testSQLServer' button can be used for reference.

## Example DLL usage

### *Application Launch*

This process is performed once on application launch and generates a handle for all further DLL calls.

```
void* UDPMan = CreateUDPManager();
```

### *Application Exit*

```
DestroyUDPManager(UDPMan);
```

### *Connection to servers*

This process is performed once the UDP Manager has been created and begins the search for all servers connected via Ethernet. This may be a mixture of FXPro (PC) and Touch Controller.

- 1 Search for external servers
- 2 Continue searching for external servers
- 3 Connect to all the servers that were found

```
1 while (!UDP_FoundInitialServers(UDPMan))  
    {  
        Sleep(4000);  
    }
```

```
2 while (UDP_GetServerListSize(UDPMan) < 1)  
    {  
        UDP_Refresh(UDPMan);  
        Sleep(5000);  
    }
```

```
3 std::vector<void*> clients;  
    int iMax = UDP_GetServerListSize(UDPMan);  
    for (int i = 0; i < iMax; ++i)  
    {  
        clients.push_back(ConnectToServer(UDPMan,i));  
    }
```

At this point a list of servers has been created and connected.

### *Identify the servers*

Each server can identify itself by name and type (PC or Touch Controller). Cycle through the server list to obtain these details. These names can then be used to pick and choose which server to use.

```
int iMax = UDP_GetServerListSize(UDPMan);
for (int i = 0; i < iMax; ++i)
{
    printf("Server %d : %ls (%s)\n", i ,UDP_GetName(UDPMan,i), (UDP_IsPC(UDPMan,i)
? "PC" : "Touch Controller"));
}
```

### *Get the number of printers connected to the server*

```
const ID91_GETNETWORKLISTPRINTERCOUNT_SERVER* a =
TCP_GetNetworkListPrinterCount(*it);
```

a->count returns the number of printers in the group network.

### *Get the list of printer names connected to the server*

```
const ID90_GETNETWORKLISTPRINTERLIST_SERVER* a = TCP_GetNetworkListPrinterList(*it);
```

a->count returns the number of printers in the group network.  
a->printers[].comPort returns the port used to connect the printer to the group.  
a->printers[].errorCode returns the error code for the printer.  
a->printers[].name returns a string to the printer name.

### *Switch to a printer within the group*

Switches to a printer called '2-----'.

```
const ID82_SWITCHTOPRINTERBYNAME_SERVER* a = TCP_SwitchToPrinterByName(*it, L"2-
-----");
```

This printer will now be the active one. Subsequent printer commands will act on this printer.

### *Determine printer type*

Assuming a connection to a list of servers has already been made, the type of printer connected to the server can be determined with:

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
const ID40_GETPRINTERTYPE_SERVER* a = TCP_GetPrinterType(*it);
printf("ok? %d, type:%d, name: %S\n",a->ok,a->type,a->name);
```

If no printer is found then ok=0. Otherwise, the type of printer (0:FXONE-S, 1:FXONE-R, 2:FXONE-P) can be determined along with the name of the printer connected as a 17 byte string.

### Determine printer firmware

Assuming a connection to a list of servers has already been made, the firmware version of the printer connected to the server can be determined with:

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
const ID41_GETPRINTERFIRMWARE_SERVER* a = TCP_GetPrinterFirmware(*it);
printf("type: %d, version: %02d.%02d\n",a->type,a->major,a->minor);
```

Type is 0: FXONE-S, 1: FXONE-R, 2:FXONE-P printer model. The major and minor digits are recombined to give a firmware version 'major.minor'.

### Determine printer status

Assuming a connection to a list of servers has already been made, the status of the printer connected to the server can be determined with:

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
const ID42_GETPRINTERSTATUS_SERVER* a = TCP_GetPrinterStatus(*it);
printf("ok? %d, lastError: %d, status: %d, no. of prints: %d\n",a->ok,a->errCode,a->status, a->numPrints);
```

'ok' indicates if the command was successful (0: false, 1: true). The printer error code and current printing status is returned. The error code list is shown below:

Enumeration	Error Code
0	ERROR_NO_ERROR
1	ERROR_LOW_INK
2	ERROR_SHUTTER_FAULT
3	ERROR_NO_FILE_ON_USB_STICK
4	ERROR_TOO_MUCH_DATA_USB_STICK
5	ERROR_DATA_INVALID_CHECKSUM_USB_STICK
6	ERROR_DATA_NO_STX_USB_STICK
7	ERROR_DATA_NO_ETX_USB_STICK
8	ERROR_DATA_SIZE_MISMATCH_USB_STICK
9	ERROR_DATA_INVALID_TYPE_USB_STICK
10	ERROR_TOO_MUCH_DATA_DIRECT_CONNECTION
11	ERROR_DATA_INVALID_CHECKSUM_DIRECT_CONNECTION
12	ERROR_DATA_NO_STX_DIRECT_CONNECTION
13	ERROR_DATA_NO_ETX_DIRECT_CONNECTION
14	ERROR_DATA_SIZE_MISMATCH_DIRECT_CONNECTION
15	ERROR_DATA_INVALID_TYPE_DIRECT_CONNECTION
16	ERROR_DATA_INVALID_ELEMENT_TYPE
17	ERROR_DATA_INVALID_TIME
18	ERROR_DATA_INVALID_DATE
19	ERROR_DATA_INVALID_SHIFT



20	ERROR_DATA_INVALID_COUNTER
21	ERROR_DATA_INVALID_BARCODE
22	ERROR_DATA_INVALID_SPIT
23	ERROR_DATA_INVALID_TICKLE
24	ERROR_DATA_INVALID_SHUTTER_OPEN_CALIBRATION
25	ERROR_DATA_INVALID_SHUTTER_CLOSE_CALIBRATION
26	ERROR_DATA_INVALID_SHUTTER_OPEN_TIME
27	ERROR_DATA_INVALID_PRINT_PARAMETERS
28	ERROR_DATA_INVALID_BACKGROUND
29	ERROR_DATA_TOO_MANY_DYNAMIC_ELEMENTS
30	ERROR_DATA_TOO_MANY_FONT_COMBINATIONS
31	ERROR_DATA_FONT_TABLE
32	ERROR_INVALID_SYMBOL
33	ERROR_PRINTER_BUSY
34	ERROR_POWER_FAULT_14V
35	ERROR_POWER_FAULT_24V
36	ERROR_POWER_FAULT_36V
37	ERROR_CORRUPTED_PACKET

The printer status is shown below:

Enumeration	Status Code
0	PRINTER_ERROR
1	PRINTER_NO_MESSAGE
2	PRINTER_MESSAGE_PRINT_OFF
3	PRINTER_MESSAGE_PRINT_ON
4	PRINTER_PRINTING
5	PRINTER_CONFIGURATION

The number of prints completed is also provided by this function.

#### *Turning printing on/off*

Assuming a connection to a list of servers has already been made, the print status of the printer connected to the server can be controlled with:

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
TCP_SetPrinterStatus(*it,PrintState);
```

PrintState – 0: Printing turned off, 1: Printing turned on.

### *Get number of installed messages*

Assuming a connection to a list of servers has already been made, the number of messages installed on the server can be determined with:

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
const ID1_GETNUMBEROFINSTALLEDMESSAGES_SERVER* a
TCP_GetNumberOfInstalledMessages(*it,NULL);
```

The number of messages located at the optional path (PC) or NULL (Touch Controller) will be returned.

### *Get list of installed message*

Assuming a connection to a list of servers has already been made, a list of messages installed on the server can be determined with:

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
const ID2_GETINSTALLEDMESSAGELIST_SERVER* a =
TCP_GetInstalledMessageList(*it,NULL);
```

The string returned contains a list of all the messages installed that are located at the optional path (PC) or NULL (Touch Controller).

### *Add a message to the server*

Assuming a connection to a list of servers has already been made, a SVN message file on a connected PC can be copied to the server:

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
TCP_AddMessage(*it,L"testop",L"C:\\",NULL);
```

The message name (without extension SVN) is specified. The source path is specified and a destination path is required for the PC version. Note, when specifying the path, the string parameter may require the use of '\\\' to represent \' due to programming language restrictions.

### *Delete a message from the server*

Assuming a connection to a list of servers has already been made, a SVN message file on the server can be removed:

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
TCP_DeleteMessage(*it,L"testop",NULL);
```

The message name (without extension SVN) is specified. The source path is required for the PC version. Note, when specifying the path, the string parameter may require the use of '\\\' to represent \' due to programming language restrictions.

### *Rename a message on the server*

Assuming a connection to a list of servers has already been made, a SVN message file on the server can be renamed:

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
TCP_RenameMessage(*it,L"testop",L"testop2",NULL);
```

The original and new message names (without extension SVN) are specified. The path is required for the PC version. Note, when specifying the path, the string parameter may require the use of '\\\' to represent '\\' due to programming language restrictions.

### *Get current message set to print*

Assuming a connection to a list of servers has already been made, the name of the current message selected to print can be retrieved:

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
const ID7_GETCURRENTMESSAGENAME_SERVER* a = TCP_GetCurrentMessageName(*it);
```

A string containing the name is returned.

### *Set a message to print*

Assuming a connection to a list of servers has already been made, a message can be sent to the printer for printing:

The first step is to load the message into the print server.

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
TCP_LoadMessage (*it,L"time1",NULL);
```

At this point, any dynamic data can be updated as required.

Now, the message can be despatched to the printer.

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
TCP_PrintMessage (*it);
```

The path is required for the PC version. Note, when specifying the path, the string parameter may require the use of '\\\' to represent '\\' due to programming language restrictions. Printing can be enabled by the TCP\_SetPrinterStatus command. Printer monitoring can be performed using the TCP\_GetPrinterStatus command.

## Updating fields

Assuming a connection to a list of servers has already been made, fields within a message can be updated:

- 1 Determine if there are any fields within the message, if so proceed with other steps
- 2 Get a list of any field names within the message
- 3 Get the field data for a specified name otherwise abort
- 4 Present the list to the operator, send the updated data back

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
```

**1** `const ID8_GETANYFIELDDATAINCURRENTMESSAGE_SERVER* a = TCP_GetAnyFieldDataInCurrentMessage(*it);`

Returns 0: no fields present in message, 1: fields present in message

**2** `const ID15_GETFIELDNAMESINCURRENTMESSAGE_SERVER* a = TCP_GetFieldNamesInCurrentMessage(*it);`

This returns a string containing field names that are available to be updated.

**3** `const ID9_GETFIELDDATAINCURRENTMESSAGE_SERVER* result = TCP_GetFieldDataInCurrentMessage(*it,L"MYFIELD");`

This returns a structure that contains all the field data for a field called 'MYFIELD'.

**4** `const ID10_SETFIELDDATAINMESSAGE_SERVER* res = TCP_SetFieldDataInMessage(*it,L"MYFIELD",L"123");`

This copies '123' string to a field called 'MYFIELD'.

## Updating counters

Assuming a connection to a list of servers has already been made, counters within a message can be updated:

- 1 Get a list of any counters within the message
- 2 Get the counter data for a specified index
- 3 Present the list to the operator, send the updated data back

```
std::vector<void*> clients; // List of clients already created
std::vector<void*>::iterator it = clients.begin() // Start of list
```

**1** `const ID11_GETCOUNTERLISTINMESSAGE_SERVER* a = TCP_GetCounterListInMessage(*it);`

Returns a structure which indicates if there are any counters in the message, the number of counters and also a set of indices to reference the individual counters.

**2** `TCP_GetCounterValueInMessage(*it,index);`

Returns a structure for the specified counter index. This structure contains all the data needed to be able to offer the counter to the user including rollover value, increment, current value, number of digits.

**3** `TCP_SetCounterValueInMessage(*it,index,count, batch);`

For the specified index, the count and batch values can be updated. For a regular counter that does not need batches, set the batch parameter to 1.

## *Using a database*

Data that is sent to the Foenix printers can be obtained from a variety of sources; manually entered, barcode scanner, checkweigher or automatically from a database. This section gives two examples on how to extract data from existing databases, one created in Access and one in SQL. The user is free to copy and reuse this code as required.

Setting up the database is beyond the scope of this section.

### *Access*

A database called 'My Database' which has no password.

```
String filenameedb="My Database";
String myConnectionString = @"Driver={Microsoft Access Driver (*.mdb)};" + "Dbq="+
filenameedb +";";// +";Uid=Admin;Pwd=;

// Open database
OdbcConnection myConnection = new OdbcConnection();
myConnection.ConnectionString = myConnectionString;

try
{
    myConnection.Open();
}catch(Exception){
    // Unable to open DB – add exception logic as required
    return;
}

// Database opened - get data from database - see Support functions below
String valuefromdb=executeQuery(ref myConnection);

// Close database
myConnection.Close();

// Prepare message
int ret=setCurrentMessage(); // See Support functions below
if (ret == 0)
{
    //Set value
    ret = setFieldValue(valuefromdb); // See Support functions below
}
```

## SQL

A database called 'My Database' which has a username 'ABC' and a password '123'. This database is managed by a SQL server called 'Server'.

```
String namedb=" My Database";
String pwd='123';
String nameserver='Server';
String username='ABC';

String myConnectionString = @"Driver={SQL Server};Server=" + nameserver + ";Database=" +
namedb + ";Uid="+username+";Pwd="+pwd+"";

//Open database
OdbcConnection myConnection = new OdbcConnection();
myConnection.ConnectionString = myConnectionString;
try
{
myConnection.Open();
}catch(Exception){
    // Unable to open DB – add exception logic as required
    return;
}

// Database opened - get data from database - see Support functions below
String valuefromdb=executeQuery(ref myConnection);

// Close database
myConnection.Close();

// Prepare message
int ret=setCurrentMessage(); // See Support functions below
if (ret == 0)
{
    //Set value
    ret = setFieldValue(valuefromdb); // See Support functions below
}
```

The commands for both types of database are identical. They use standard SQL functions to connect to a database and extract data. The only difference between the two types is the initial connection method.

### *Support functions used*

**executeQuery** uses the previously established connection to extract all the row data and insert into a list. The list is then returned. A third party application may choose to run a specific query based on the data required. In which case the 'sql' command string will be modified accordingly.

```
protected String executeQuery(ref OdbcConnection myConnection)
{
    // Read all the data in the database into a list
    // Could replace this with a single read operation instead

    // The database is called 'Table1'
    // The field item in the database is located at 'Col2'
    String table = "Table1";
    String field = "Col2";
    String sql = "SELECT " + field + " FROM " + table + ";";

    OdbcCommand com = new OdbcCommand(sql, myConnection);
    List<String> list=new List<String>();
    OdbcDataReader reader = com.ExecuteReader();

    // Get all the data
    while (reader.Read())
    {
        string word = reader.GetString(0);
        list.Add(word);
    }
    if (list.Count > 0)
        return list[0];
    return "";
}
```

**setCurrentMessage** loads the message called 'testdb' to the printer. The example includes a path 'C:\\' which indicates the current printer server is a PC running FXPro. The path is NULL for the Touch Controller version.

```
private int setCurrentMessage() {
    // Set current message to 'testdb'

    String name="testdb";
    bool ret;
    byte ok;

    // Load the message to the printer
    ret = currentServer.TCP_LoadMessage (name,L"C:\\",out ok);
    currentServer.TCP_PrintMessage (name);

    return 0;
}
```



**setFieldValue** sends the passed string to a field called 'MyField'. The string needs to be padded or truncated to ensure the length matches the field size otherwise it will be rejected.

```
int setFieldValue(String value){
    // The incoming field data needs to match the size permitted by the field!
    // The field being populated is called 'MyField'
    byte ok;
    bool ret = currentServer.TCP_SetFieldDataInMessage('MyField', value, out ok);
    return 0;
}
```

## Appendix : functions

### *DLL Configuration Related Functions*

Function Name	<b>CreateUDPManager</b>
Inputs	None
Outputs	Handle to a UDP Manager
Notes	Called to establish a UDP Manager that begins to search for remote FXPro or Touch Controllers. This is done typically at application launch

Function Name	<b>DestroyUDPManager</b>
Inputs	Handle to UDP Manager
Outputs	None
Notes	Called to destroy a previously setup manager when tidying up resources prior to application shutdown

Function Name	<b>UDPRefresh</b>
Inputs	Handle to UDP Manager
Outputs	None
Notes	Called to restart the UDP polling process during initial FXPro or Touch Controller searching

Function Name	<b>DestroyUDPManager</b>
Inputs	Handle to UDP Manager
Outputs	None
Notes	Called to destroy a previously setup manager

Function Name	<b>GetServerListSize</b>
Inputs	Handle to UDP Manager
Outputs	Number of servers found
Notes	This returns the number of PC FXPro or Touch Controllers located over Ethernet (or local) that are responding to the DLL

Function Name	<b>UDP_GetName</b>
Inputs	Handle to UDP Manager, index
Outputs	Returns a string with the name of the server being addressed
Notes	Each server that has responded to the DLL call is assigned an index. The user can cycle through each index and extract the name of the server. This allows individual servers to be identified and displayed to the user

Function Name	<b>UDP_IsPC</b>
Inputs	Handle to UDP Manager, index
Outputs	Returns true or false
Notes	For a particular index, the server can identify itself as being a FXPro PC application (true) or Touch Controller (false). Some DLL commands are specific just for the Touch Controller so this function can be used to distinguish between the different types

Function Name	<b>ConnectToServer</b>
Inputs	Handle to UDP Manager, index
Outputs	None
Notes	Connects the UDP Manager to a specified server (FXPro or Touch Controller). This needs to be done before the host system can issue commands to the server

Function Name	<b>UDP_FoundInitialServers</b>
Inputs	Handle to UDP Manager
Outputs	Returns true or false
Notes	Starts the polling process to search for FXPro or Touch Controller servers. The function will return true if any servers are found or false otherwise within a time period. The user can subsequently call UDP_Refresh to continue the search

### Printer Related Functions

A single or group of connected printers to a FXPro PC application or Touch Controller can be monitored remotely.

Function Name	<b>TCP_GetNetworkListPrinterCount</b>
Inputs	Handle to UDP Manager
Outputs	Number of printers
Notes	Returns the number of printers in the network group connected to the currently selected server.

Function Name	<b>TCP_GetNetworkListPrinterList</b>
Inputs	Handle to UDP Manager
Outputs	Number of printers Structure for each printer index containing: comport – the port used to connect the printer to the group. errorCode – the current error status of the printer. name – the name of the printer as a string.
Notes	Returns the number of printers in the network group connected to the currently selected server.

Function Name	<b>TCP_SwitchToPrinterByname</b>
Inputs	Handle to UDP Manager, printer name as a string.
Outputs	ok – if switch performed successfully.
Notes	A delay of approximately 1 second may be required to allow the server to disconnect from one printer and establish connection with the new printer.

Function Name	<b>TCP_GetPrinterType</b>
Inputs	Handle to UDP Manager
Outputs	Name of printer[17 bytes], Type of printer 1 byte - 0:FXPRO-S, 1:FXPRO-R, 2:FXPRO-P Ok – 0:Printer not found, 1:Printer found
Notes	Returns whether there is a printer connected to the server and what type it is (FXPRO-S, FXPRO-R or FXPRO-P). The name assigned to that printer is also returned

Function Name	<b>TCP_GetPrinterFirmware</b>
Inputs	Handle to UDP Manager
Outputs	Type of printer 1 byte - 0:FXPRO-S, 1:FXPRO-R, 2:FXPRO-P Major firmware version 1 byte Minor firmware version 1 byte
Notes	Returns the type of printer is connected (FXPRO-S, FXPRO-R or FXPRO-P) and also the firmware version installed in that printer

Function Name	<b>TCP_GetPrinterStatus</b>
Inputs	Handle to UDP Manager
Outputs	Ok – 0:Printer not found, 1:Printer found Error code 1 byte - see enumerated list below Status code 1 byte – see enumerated list below Number of prints 4 bytes – number of prints done by this printer
Notes	Returns the type of printer is connected (FXPRO-S, FXPRO-R or FXPRO-P) and also the firmware version installed in that printer

Please consult the printer user manual for further information about these codes.

Enumeration	Error Code
0	ERROR_NO_ERROR
1	ERROR_LOW_INK
2	ERROR_SHUTTER_FAULT
3	ERROR_NO_FILE_ON_USB_STICK
4	ERROR_TOO_MUCH_DATA_USB_STICK
5	ERROR_DATA_INVALID_CHECKSUM_USB_STICK
6	ERROR_DATA_NO_STX_USB_STICK
7	ERROR_DATA_NO_ETX_USB_STICK
8	ERROR_DATA_SIZE_MISMATCH_USB_STICK
9	ERROR_DATA_INVALID_TYPE_USB_STICK
10	ERROR_TOO_MUCH_DATA_DIRECT_CONNECTION
11	ERROR_DATA_INVALID_CHECKSUM_DIRECT_CONNECTION
12	ERROR_DATA_NO_STX_DIRECT_CONNECTION
13	ERROR_DATA_NO_ETX_DIRECT_CONNECTION
14	ERROR_DATA_SIZE_MISMATCH_DIRECT_CONNECTION
15	ERROR_DATA_INVALID_TYPE_DIRECT_CONNECTION
16	ERROR_DATA_INVALID_ELEMENT_TYPE
17	ERROR_DATA_INVALID_TIME
18	ERROR_DATA_INVALID_DATE
19	ERROR_DATA_INVALID_SHIFT
20	ERROR_DATA_INVALID_COUNTER
21	ERROR_DATA_INVALID_BARCODE
22	ERROR_DATA_INVALID_SPIT
23	ERROR_DATA_INVALID_TICKLE
24	ERROR_DATA_INVALID_SHUTTER_OPEN_CALIBRATION
25	ERROR_DATA_INVALID_SHUTTER_CLOSE_CALIBRATION
26	ERROR_DATA_INVALID_SHUTTER_OPEN_TIME
27	ERROR_DATA_INVALID_PRINT_PARAMETERS
28	ERROR_DATA_INVALID_BACKGROUND
29	ERROR_DATA_TOO_MANY_DYNAMIC_ELEMENTS
30	ERROR_DATA_TOO_MANY_FONT_COMBINATIONS
31	ERROR_DATA_FONT_TABLE
32	ERROR_INVALID_SYMBOL

33	ERROR_PRINTER_BUSY
34	ERROR_POWER_FAULT_14V
35	ERROR_POWER_FAULT_24V
36	ERROR_POWER_FAULT_36V
37	ERROR_CORRUPTED_PACKET

Enumeration	Status Code
0	PRINTER_ERROR
1	PRINTER_NO_MESSAGE
2	PRINTER_MESSAGE_PRINT_OFF
3	PRINTER_MESSAGE_PRINT_ON
4	PRINTER_PRINTING
5	PRINTER_CONFIGURATION

Function Name	<b>TCP_SetPrinterStatus</b>
Inputs	Handle to UDP Manager, Print status
Outputs	Ok – 0:Printer not found, 1:Printer found
Notes	Turns printing on (1) or off (0)

Function Name	<b>TCP_GetPrinterSettings</b>
Inputs	Handle to UDP Manager
Outputs	Ok – 0:Printer not found, 1:Printer found Printer settings structure (see below)
Notes	Turns printing on (1) or off (0)

Property	Notes
Orientation	0: Normal, 1: Upside down printing
Direction	0: Right to left, 1: Left to right printing
Ink Type	0: Oil, 1:Solvent
Shutter Type	0: Not fitted, 1: Manual, 2: Motorised
Autoprint	0: Single shot, 1: Repeat print
Encoder	0: Not fitted, 1: Fitted
Ink Saver	0: Disabled, 1: Enabled
Spit Time	Time in seconds (up to 255) between spits
Tickle Time	Time in seconds (up to 255) between tickles
Shutter Time	Time in seconds (up to 255) before shutter closes
Shutter Open Calibration	Calibration value (0 to 999)
Shutter Close Calibration	Calibration value (0 to 999)
Printer Name[17 bytes]	ASCII printer name

Function Name	<b>TCP_SetPrinterSettings</b>
Inputs	Handle to UDP Manager Orientation Direction Ink Type Shutter Type Autoprint Encoder Ink Saver Spit Time Tickle Time Shutter Time Shutter Open Calibration Shutter Close Calibration Printer Name[17 bytes]
Outputs	Ok – 0:Printer not found, 1:Printer found
Notes	Writes the printer settings back to the printer. It is the users responsibility to make sure the data is valid

*FXPro And Touch Controller Settings Related Functions*

These functions relate to both the FXPro PC application and Touch Controller.

Function Name	<b>TCP_GetLanguage</b>
Inputs	Handle to UDP Manager
Outputs	Single byte representing language based on winnt.h LANG_ENGLISH      0x09 LANG_ARABIC        0x01 LANG_CHINESE       0x04 LANG_FARSI         0x29 LANG_GERMAN        0x07 LANG_GREEK         0x08 LANG_SPANISH       0x0a LANG_FRENCH        0x0c LANG_HEBREW        0x0d LANG_ITALIAN        0x10 LANG_JAPANESE      0x11 LANG_KOREAN        0x12 LANG_DUTCH         0x13 LANG_PORTUGUESE   0x16 LANG_POLISH        0x15 LANG_RUSSIAN        0x19 LANG_TURKISH       0x1f
Notes	Returns the current language setting in the server

Function Name	<b>TCP_SetLanguage</b>
Inputs	Handle to UDP Manager, language (based on winnt.h)
Outputs	Ok- 0: Error, 1: No Error
Notes	Returns the current language setting in the server

Function Name	<b>TCP_GetShiftSettings</b>
Inputs	Handle to UDP Manager
Outputs	The shift structure below is populated
Notes	Fills in the shift structure from the server



## ShiftSettings

ShiftDay[0]
ShiftDay[1]
ShiftDay[2]
ShiftDay[3]
ShiftDay[4]
ShiftDay[5]
ShiftDay[6]
Start Of Day - Hour
Start Of Day - Minute

Each shift day is defined as;

Start Hour[0]	Start Minute[0]	Shift Code[0]
Start Hour[1]	Start Minute[1]	Shift Code[1]
Start Hour[2]	Start Minute[2]	Shift Code[2]
Start Hour[3]	Start Minute[3]	Shift Code[3]

Function Name	<b>TCP_SetShiftSettings</b>
Inputs	Handle to UDP Manager, Shift hours [28] bytes – 4 shifts per day, 7 days per week starting Monday Shift minutes [28] bytes – 4 shifts per day, 7 days per week starting Monday Shift codes [28] ASCII bytes – 4 shifts per day, 7 days per week starting Monday Start of day hours – 1 byte Start of day minutes – 1 byte
Outputs	Ok- 0: Error, 1: No Error
Notes	The shift pattern is sent as a group of three arrays consisting of all the hours, minutes and ASCII codes that define each of the 4 shifts that can be used for a 7 day week. It is the responsibility of the user to ensure the data is valid

Function Name	<b>TCP_GetCalendarType</b>
Inputs	Handle to UDP Manager
Outputs	Single byte representing calendar type; 0: Gregorian 1: Jalali
Notes	The calendar type is used for date forwarding calculations. The default setting is Gregorian

Function Name	<b>TCP_SetCalendarType</b>
Inputs	Handle to UDP Manager, calendar type (0: Gregorian, 1: Jalali)
Outputs	Ok- 0: Error, 1: No Error
Notes	Returns the current language setting in the server

Function Name	<b>TCP_GetCartridgeCost</b>
Inputs	Handle to UDP Manager
Outputs	Major value (4 bytes) Minor value (4 bytes)
Notes	Returns the decimal (major.minor) value that indicates the cost of the cartridge in relevant currency. There is no need to specify the currency as this value is used to scale the volume of ink used per print

Function Name	<b>TCP_SetCartridgeCost</b>
Inputs	Handle to UDP Manager Major value (4 bytes) Minor value (4 bytes)
Outputs	Ok- 0: Error, 1: No Error
Notes	It is the users responsibility to convert the xxx.yyy value into appropriate major and minor parts

Function Name	<b>TCP_GetTime</b>
Inputs	Handle to UDP Manager
Outputs	Structure with; Hours (1 byte) Minutes (1 byte)
Notes	Returns the current time in the server

Function Name	<b>TCP_SetTime</b>
Inputs	Handle to UDP Manager Hours (1 byte) Minutes (1 byte)
Outputs	Ok- 0: Error, 1: No Error
Notes	It is the users responsibility to confirm the hours and minutes are within the correct range

Function Name	<b>TCP_GetDate</b>
Inputs	Handle to UDP Manager
Outputs	Structure with; Day (1 byte) Month (1 byte) Year (1 byte)
Notes	Returns the current date in the server. The returned year starts from 2000

Function Name	<b>TCP_SetDate</b>
Inputs	Handle to UDP Manager Day (1 byte) Month (1 byte) Year (1 byte)
Outputs	Ok- 0: Error, 1: No Error
Notes	It is the users responsibility to confirm the days, months and years are within the correct range (year starts from 2000)

Function Name	<b>TCP_GetCOMPortSetting</b>
Inputs	Handle to UDP Manager
Outputs	Structure containing COM settings: Baudrate (4 bytes) Databits – 7 or 8 (1 byte) Stopbits – 1 or 2 (1 byte) Parity – 0: None, 1: Odd, 2: Even (1 byte)
Notes	Returns the COM port setting for the server

Function Name	<b>TCP_SetCOMPortSetting</b>
Inputs	Handle to UDP Manager Baudrate (4 bytes) Databits – 7 or 8 (1 byte) Stopbits – 1 or 2 (1 byte) Parity – 0: None, 1: Odd, 2: Even (1 byte)
Outputs	Ok- 0: Error, 1: No Error
Notes	It is the responsibility of the user to ensure the COM settings are valid for that port type e.g. baudrate is achievable and the additional parameters are within range

### *Touch Controller Settings Related Functions*

These functions relate to a Touch Controller only.

Function Name	<b>TCP_GetControllerFirmware</b>
Inputs	Handle to UDP Manager
Outputs	Major firmware version 1 byte Minor firmware version 1 byte
Notes	Returns firmware version installed in that controller

Function Name	<b>TCP_GetScreenBrightness</b>
Inputs	Handle to UDP Manager
Outputs	Brightness (0 to 99) 1 byte
Notes	Returns the brightness value set for the screen backlight

Function Name	<b>TCP_SetScreenBrightness</b>
Inputs	Handle to UDP Manager, brightness (0 to 99)
Outputs	Ok- 0: Error, 1: No Error
Notes	Sets the brightness value for the screen backlight

Function Name	<b>TCP_GetBeeper</b>
Inputs	Handle to UDP Manager
Outputs	Beeper state 0: Off, 1: On
Notes	Returns the beeper state for the controller

Function Name	<b>TCP_SetBeeper</b>
Inputs	Handle to UDP Manager, beeper state 0: Off, 1: On
Outputs	Ok- 0: Error, 1: No Error
Notes	Sets the beeper state for the controller

Function Name	<b>TCP_GetIPAddress</b>
Inputs	Handle to UDP Manager
Outputs	Little endian IP address in hex
Notes	Returns the IP Address of the controller

Function Name	<b>TCP_SetIPAddress</b>
Inputs	Handle to UDP Manager Little endian IP address in hex
Outputs	Ok- 0: Error, 1: No Error
Notes	Sets the IP Address of the controller. Note it may be necessary to rebind the interface to use any settings changes

Function Name	<b>TCP_GetSubnetMask</b>
Inputs	Handle to UDP Manager
Outputs	Little endian subnet mask in hex
Notes	Returns the Subnet mask of the controller

Function Name	<b>TCP_SetSubnetMask</b>
Inputs	Handle to UDP Manager Little endian subnet mask in hex
Outputs	Ok- 0: Error, 1: No Error
Notes	Sets the subnet mask of the controller. Note it may be necessary to rebind the interface to use any settings changes

Function Name	<b>TCP_GetDHCPState</b>
Inputs	Handle to UDP Manager
Outputs	DHCP in use? - 0: Disabled, 1: Enabled
Notes	Returns the DHCP status of the controller

Function Name	<b>TCP_SetDHCPState</b>
Inputs	Handle to UDP Manager DHCP usage - 0: Disabled, 1: Enabled
Outputs	Ok- 0: Error, 1: No Error
Notes	Sets the use of DHCP in the controller

Function Name	<b>TCP_RebindInterface</b>
Inputs	Handle to UDP Manager Interface (1 byte) – default 0
Outputs	IP Address
Notes	Forces the controller to rebind the Ethernet interface to use any settings changes e.g. IP Address, Subnet mask and DHCP usage. The function returns the IP Address

Function Name	<b>TCP_GetInstalledFontList</b>
Inputs	Handle to UDP Manager
Outputs	A string of font names
Notes	A single list of installed fonts in the controller. This list can then be used to reference a font for deletion or confirm that a particular font is installed

Function Name	<b>TCP_AddFont</b>
Inputs	Handle to UDP Manager Font name Path on host PC where font is stored
Outputs	Ok- 0: Error, 1: No Error
Notes	The name of the font includes the extension 'ttf'. The path can reference any drive that is available within Windows Explorer. If coding in C (or similar) it may be necessary to include '\\ within the path string as a single \' will be interpreted as a formatting character

Function Name	<b>TCP_DeleteFont</b>
Inputs	Handle to UDP Manager Font name
Outputs	Ok- 0: Error, 1: No Error
Notes	The name of the font includes the extension 'ttf'. Any message that uses this deleted font will need to have a new font chosen

Function Name	<b>TCP_GetInstalledLogoList</b>
Inputs	Handle to UDP Manager
Outputs	A string of logo names
Notes	A single list of installed logos in the controller. This list can then be used to reference a logo for deletion or confirm that a particular logo is installed

Function Name	<b>TCP_AddLogo</b>
Inputs	Handle to UDP Manager Logo name Path on host PC where logo is stored
Outputs	Ok- 0: Error, 1: No Error
Notes	The name of the logo includes the extension e.g. 'png'. The path can reference any drive that is available within Windows Explorer. If coding in C (or similar) it may be necessary to include '\\\' within the path string as a single \' will be interpreted as a formatting character

Function Name	<b>TCP_DeleteLogo</b>
Inputs	Handle to UDP Manager Logo name
Outputs	Ok- 0: Error, 1: No Error
Notes	The name of the logo includes the extension e.g. 'png'. Any message that uses this deleted logo will need to have a new font chosen

*Message Transfer And Management Related Functions*

Function Name	<b>TCP_GetNumberOfInstalledMessages</b>
Inputs	Handle to UDP Manager Optional path
Outputs	The number of messages available
Notes	This function returns the number of messages available from the server. A PC FXPro can specify an optional folder path to search within. The Touch Controller has a fixed location so is NULL

Function Name	<b>TCP_GetInstalledMessageList</b>
Inputs	Handle to UDP Manager Optional path
Outputs	A string of message names
Notes	This function returns a string containing the names of all the messages available on the server. A PC FXPro can specify an optional folder path to search within. The Touch Controller has a fixed location so is NULL

Function Name	<b>TCP_AddMessage</b>
Inputs	Handle to UDP Manager Message name Path on host PC where message is stored
Outputs	Ok- 0: Error, 1: No Error
Notes	The name of the message doesn't need to have the extension SVN. The path can reference any drive that is available within Windows Explorer. If coding in C (or similar) it may be necessary to include '\\ within the path string as a single \' will be interpreted as a formatting character

Function Name	<b>TCP_DeleteMessage</b>
Inputs	Handle to UDP Manager Message name Optional path
Outputs	Ok- 0: Error, 1: No Error
Notes	The name of the message doesn't need to have the extension SVN. There is no undo. A PC FXPro can specify an optional folder path to search within. The Touch Controller has a fixed location so is NULL

Function Name	<b>TCP_RenameMessage</b>
Inputs	Handle to UDP Manager Original message name New message name Optional path
Outputs	Ok- 0: Error, 1: No Error
Notes	The name of the message doesn't need to have the extension SVN. There is no undo. A PC FXPro can specify an optional folder path to search within. The Touch Controller has a fixed location so is NULL

Function Name	<b>TCP_GetUSBStickConnectedState</b>
Inputs	Handle to UDP Manager
Outputs	Ok- 0: Not connected, 1: Connected
Notes	This is the state of the USB stick connected to the controller or PC not the printer

Function Name	<b>TCP_GetCurrentMessageName</b>
Inputs	Handle to UDP Manager
Outputs	Message name string (17 bytes)
Notes	This function returns the name of the current message name set to print

Function Name	<b>TCP_GetAnyFieldDataInCurrentMessage</b>
Inputs	Handle to UDP Manager
Outputs	Ok- 0: No fields, 1: Fields
Notes	This returns whether there are any fields within the message that can be updated

Function Name	<b>TCP_GetFieldNamesInCurrentMessage</b>
Inputs	Handle to UDP Manager
Outputs	A string containing all the field labels
Notes	This function is called if TCP_GetAnyFieldDataInCurrentMessage indicates there is at least one field in the current message

Function Name	<b>TCP_GetFieldDataInCurrentMessage</b>
Inputs	Handle to UDP Manager Field name
Outputs	A structure containing; The field data (50 bytes) – ASCII string Match (1 byte) – 0: No matching field found, 1: Match found Length (1 byte) – the number of characters needed in the field Type (1 byte) - 0: numbers, 1: letters
Notes	This function is called if TCP_GetAnyFieldDataInCurrentMessage indicates there is at least one field in the current message. The field structure is populated with the current field information. This allows the user to extract the current data and replace with new

Function Name	<b>TCP_SetFieldDataInCurrentMessage</b>
Inputs	Handle to UDP Manager Field name Field data
Outputs	Ok- 0: Error, 1: No Error
Notes	This function writes the 'Field data' string to the field called 'Field name'

Function Name	<b>TCP_GetCounterListInMessage</b>
Inputs	Handle to UDP Manager
Outputs	Ok- 0: No counters, 1: Counters present Len (2 bytes) – the length of the counter list string Array of indices (2 bytes) – used to reference the counter
Notes	This returns a list of counters in the current message



Function Name	<b>GetCounterValueInMessage</b>
Inputs	Handle to UDP Manager Index into counter
Outputs	Value (4 bytes) – current value of counter Batch (4 bytes) – current value within a batch (batch counter) Range (4 bytes) – number of digits for counter Rollover (4 bytes) – the number at which the counter rolls over to the start Type(1 byte) – 0: count up, 1: count down Batch Size (4 bytes) – The number of products per batch Suppress Zeroes (1 byte) – 0: No, 1: Yes Edit Out Of Message (1 byte) – 0: No, 1: Yes Ok- 0: Index invalid, 1: Index valid
Notes	This returns a structure which has all the information needed to determine for the specified index, the current counter value for display purposes.

Function Name	<b>TCP_SetCounterValueInMessage</b>
Inputs	Handle to UDP Manager Index New counter value New batch value (1 if not using a batch)
Outputs	Ok- 0: Error, 1: No Error
Notes	This function writes the 'Field data' string to the field called 'Field name'

Function Name	<b>TCP_LoadMessage</b>
Inputs	Handle to UDP Manager Message name Optional path
Outputs	Ok- 0: Error, 1: No Error
Notes	This function loads the specified message name to the server. At this stage, the message is not in the printer. The user can then interrogate the message for any dynamic data and update the message prior to it being sent to the printer. The name of the message doesn't need to have the extension SVN. A PC FXPro can specify an optional folder path to search within. The Touch Controller has a fixed location so is NULL

Function Name	<b>TCP_PrintMessage</b>
Inputs	Handle to UDP Manager
Outputs	Ok- 0: Error, 1: No Error
Notes	This function transmits the currently loaded message name to the server. Any dynamic content is rolled up prior to despatch.