Important Information

DECLARATION OF EMC CONFORMITY

The FreeWay has been independently tested to be in compliance with the following standards: EN55022, EN61000.

WARNING

Under NO circumstances should AC mains voltage be applied to any connector, terminal or enclosure part of the FreeWay.

The FreeWay should not be used in locations where it will be subjected to high levels of radiated R.F., or where its connecting cables will be subjected to high levels of R.F. interference from other sources.

GUARANTEE

All products are covered by a 1-year return-to-base guarantee for parts and labour due to any manufacturing defect.

POWER SUPPLY

The power supply provided with this product is specifically designed for use with this product for the intended country of application. It converts the local mains voltage AC to a nominal regulated 9V DC voltage supplying up to 500mA of current. The unit operates as soon as it is plugged into the wall socket. This power supply is for indoor use only! Do not expose the unit to water, rain or dust. The power supply must not be covered over. Do not attempt to remove the casing – this should only be done by a qualified engineer.

WARNING: Dangerous Voltage!

Remove the unit from the wall socket when not in use. The unit is protected against short-circuit and overload by a thermal fuse. Never use a fuse with a higher rating than specified. The power supply is class II approved. The socket should always be easily accessible.

Contents

Introduction

Thank you for your investment in the DKT FreeWay AV Gateway Controller. As the name suggests you now have in your hands a powerful and flexible tool for integrating and controlling equipment using the most common control interfaces.

Before delving into the details here is a quick résumé:

- Embedded Web Server for HTML based control
- Embedded Real-Time Clock for calendar alarm and astronomic events
- Powerful FreeScript macro language
- FreeScript compiler using FreeWay Interface PC software
- FreeScript functions accessible from any Web Browser
- FreeScript debugging
- Built-in functions for real-time event handling

FreeWay's I/O:

- 6 x RS232 serial ports – on 9-way male D-type connectors, full-duplex, supporting all the common baud rates and settings, with & without handshaking
- 3 x RS485 ports – 2 on 3-way terminals, 1 on an RJ45 with 9V DC power, half duplex supporting all the common baud rates and settings. These can be used in 2 or 4-wire configurations and are compatible with RS-422
- 1 x 10Base-T Ethernet port – for communications using TCP, UDP and file transfers via FTP
- 4 x LEDs – general purpose programmable status indicators on the front panel
- 4 x Infra-Red outputs – on mono mini-jacks fully selectable and matrixable
- 1 x Infra-Red Receiver – for learning Remote Control commands and decoding RC5 commands
- 6 x opto-isolated Digital Inputs
- 6 x opto-isolated Digital Outputs
- 1 x EIB/KNX bus interface connector allowing 100 EIB/KNX groups

After a quick tour of the front and rear panels this manual will take you through the steps of basic configuration and script writing, before delving into detailed descriptions of all the FreeWay's ports and functions. We recommend that you also read the **FreeScript Programming Reference**.

This manual assumes a certain understanding of the various control interfaces, although you don't need to be an expert in any of them.

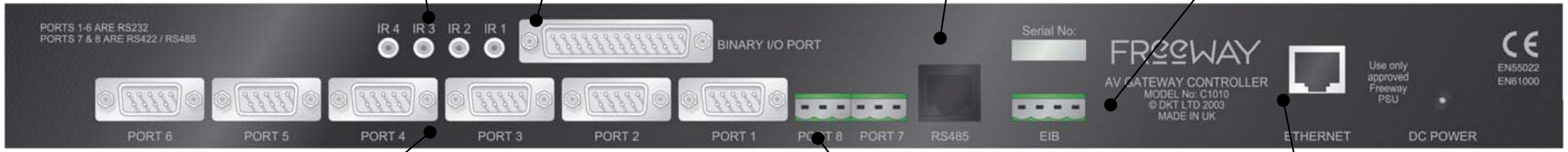
Rear Panel

4 x Infra-Red transmitters on mini-jacks for controlling equipment using Infra-Red emitters

Digital Inputs & Outputs 25-way female D-Type. Provides 6 digital inputs and 6 digital outputs with 5V DC power.

RS485 serial port on an RJ45 with 9V DC power

EIB on a 4-way terminal connector for interfacing to EIB/KNX networks

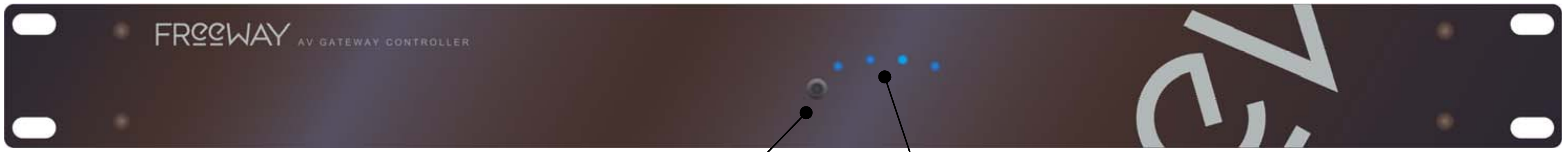


6 x RS-232 serial ports on 9-way male D-Type connectors. Used for controlling RS232 compatible equipment

2 RS-485 serial ports on 3-way terminal connectors. Used for connecting to RS485 and RS422 networks & equipment

Ethernet on RJ45 10Base-T for control & communication using TCP/IP

Front Panel



Front panel Infra-Red receiver used for learning and decoding commands from infra-red remote control handsets

4 Front Panel Status LEDs **– general purpose and system status indicators**

Getting Started

The applications and configurations that FreeWay lends itself to are widespread and numerous. However you use the FreeWay though, the same basic steps are taken to set it up and use it. These can be summarised as follows:

- Connecting to FreeWay
- Basic FreeWay configuration (optional)
- Writing a script
- Compiling and downloading a script
- Testing the Script
- Implementing a user interface (optional)
- Using the FreeWay

We'll explore these steps in a bit more detail.

Connecting to FreeWay

FreeWay is configured using the FreeWay Interface PC software. This is used for:

- Viewing, compiling and downloading scripts
- Changing FreeWay settings
- Learning Infra-Red codes
- Viewing the error log
- Testing functions and debugging scripts

FreeWay Interface software connects to FreeWay over a standard TCP/IP Ethernet connection. Setting up a connection between your computer and FreeWay is straightforward. You must obviously have a network card and TCP/IP drivers installed and working on your PC.

Connect the FreeWay to your computer using a CAT5 Ethernet cable either

- Directly using a standard Ethernet crossover cable
- Or via a standard Ethernet network hub

The FreeWay's default IP address is 192.168.1.111. If the FreeWay's default subnet address (the '1' in the IP address) is different from that of your network you may find it easier to connect directly to FreeWay using a crossover cable. You'll may also need to temporarily give your computer a fixed IP address. This is usually done in your computer's TCP/IP driver's network settings.

You can test that you are properly connected to the FreeWay by 'pinging' it. Open up a Command Prompt window and type the following:

```
ping 192.168.1.111
```

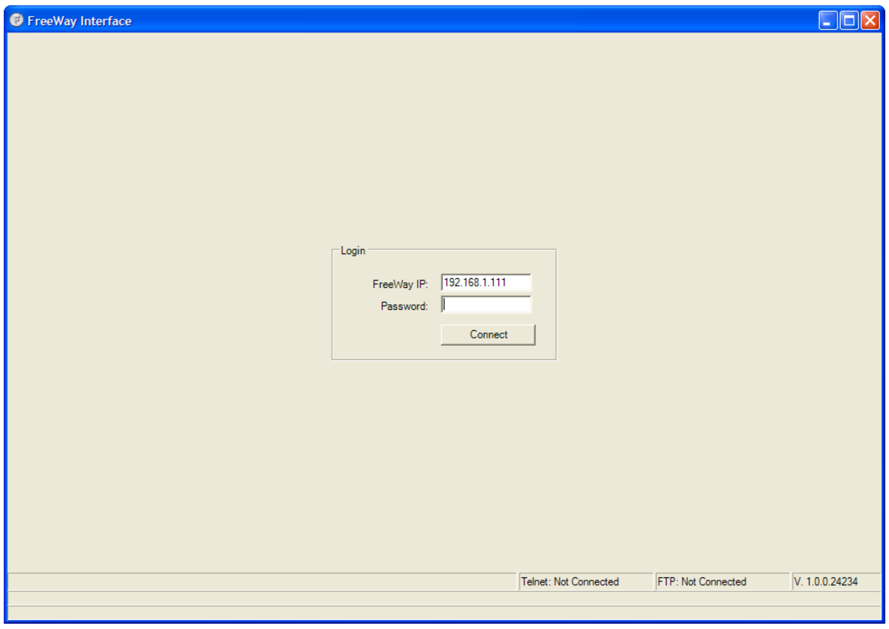
If you get replies from that address then you're properly connected.

FreeWay Interface Software

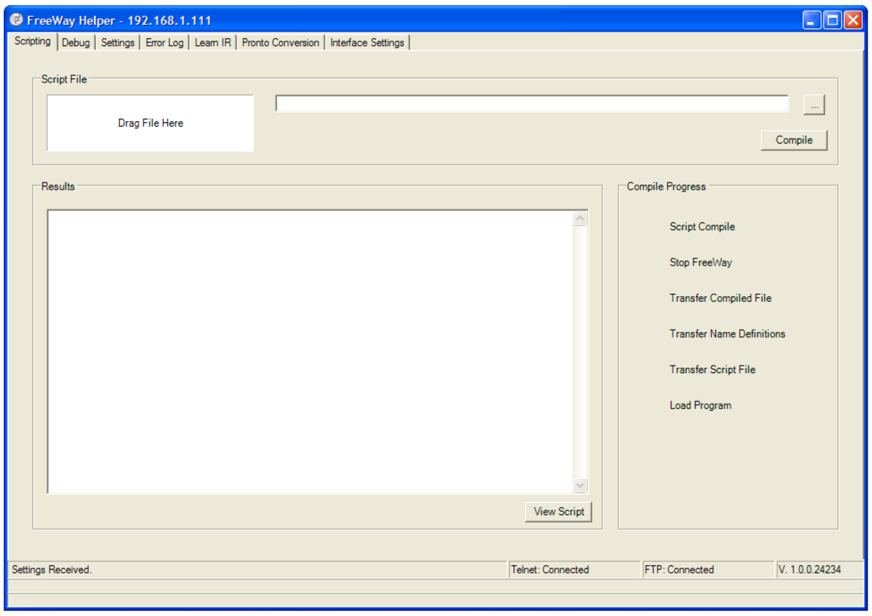
FreeWay Interface is shipped on a CD-ROM with the FreeWay and is also available for download from the DKT website at:

http://www.dktgroup.com/freeway

FreeWay Interface is compatible with Windows XP and Vista and requires Microsoft .Net Framework. When you run FreeWay Interface you'll be presented with the following:

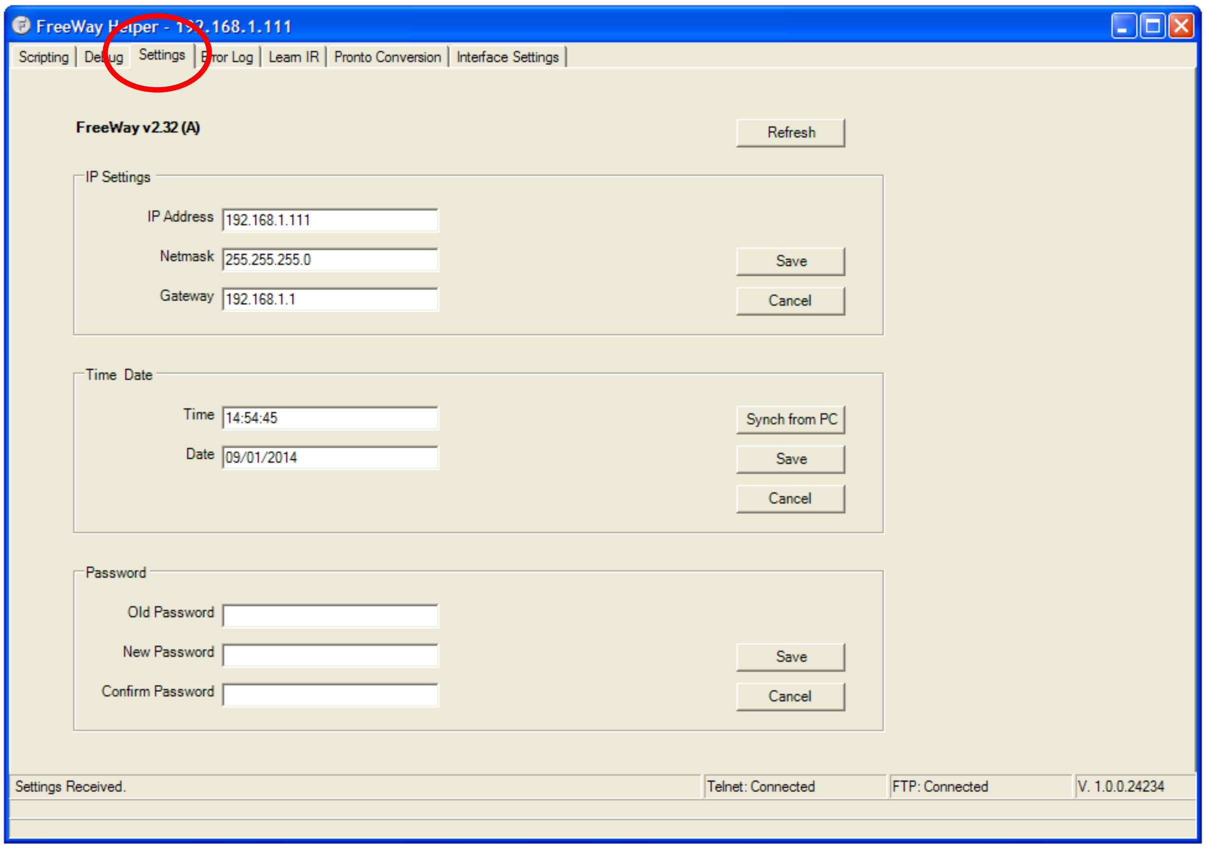


To login to the FreeWay you must enter its IP address and password. The default IP address is **192.168.1.111**. The default password is **dkt**. Press the Connect button. On a successful connection you'll see the Scripting page. This page is used to load and compile your scripts.



FreeWay Configuration

This step is optional because you may not need to change the basic settings that the FreeWay was shipped with. To access the FreeWay settings click on the **Settings** tab at the top of the page.


FreeWay Helper - 192.168.1.111
Scripting | Debug | Settings | Error Log | Learn IR | Pronto Conversion | Interface Settings
Refresh
Save
Cancel
Synch from PC
Save
Cancel
Save
Cancel

FreeWay v2.32 (A)

IP Settings

IP Address: 192.168.1.111
Netmask: 255.255.255.0
Gateway: 192.168.1.1

Time Date

Time: 14:54:45
Date: 09/01/2014

Password

Old Password
New Password
Confirm Password

Settings Received. | Telnet: Connected | FTP: Connected | V. 1.0.0.24234

FreeWay Version

This is shown at the top of the page in bold. (A) after the version number indicates that an IR board is <u>not</u> fitted. (B) indicates that it is fitted.

IP Address

By default this is 192.168.1.111. You may want to change this if, for example:

- Another device on your network uses this address
- The subnet address doesn't match that of your network
- The network settings are different where you're setting FreeWay up to the settings where you plan to install the FreeWay
-

Netmask

By default this is 255.255.255.0. You'd typically change this if FreeWay needs to communicate with devices over TCP on a different netmask.

Gateway IP Address

By default this is 192.168.1.1. This usually the IP address of the router on your network.

After making changes to any of the above settings click on the Save button. FreeWay will then automatically reboot with the new settings.

Date & Time

The date should be today's date and the time will be Greenwich Mean Time according to our clock. You may need to change the time if you are in a different time zone from GMT. To change the time or date just edit the values in the text boxes and clock on the Save button.

You can click on the Synch from PC button to set the FreeWay clock to the same as that on your PC.

Password

You may change the default (dkt) password if you like. It should have a maximum of 16 characters with no special characters or spaces. Type in the old password and the new password and click on **Save**. Note that you must have a password – do not try to enter nothing as a new password.

Writing a Script

Now you'll need to write a script file using the FreeScript language to programme the FreeWay for its intended application. At this point you may want to read the **FreeScript Programming Reference** document to familiarise yourself with FreeScript.

If you don't want to do that at this stage – no problem – the script example shown below can be used to demonstrate the whole process of compiling, downloading and testing your script.

If you are at all familiar with programming languages such as C or Java or even Basic then the script should make some sense. If not, don't worry, you can just use it as is. The comments in the script (i.e. lines that start with a //) explain the basic principles.

We'll use the script below to illustrate some fundamentals of scripting and the FreeWay operation. You should type the following script into a text file using any text editor. You can also download it from the DKT website (www.dktgroup.com), or load it from the CD-ROM accompanying FreeWay. It is called **demo with error.txt**.

A few notes on your choice of editor are worthy at this stage.

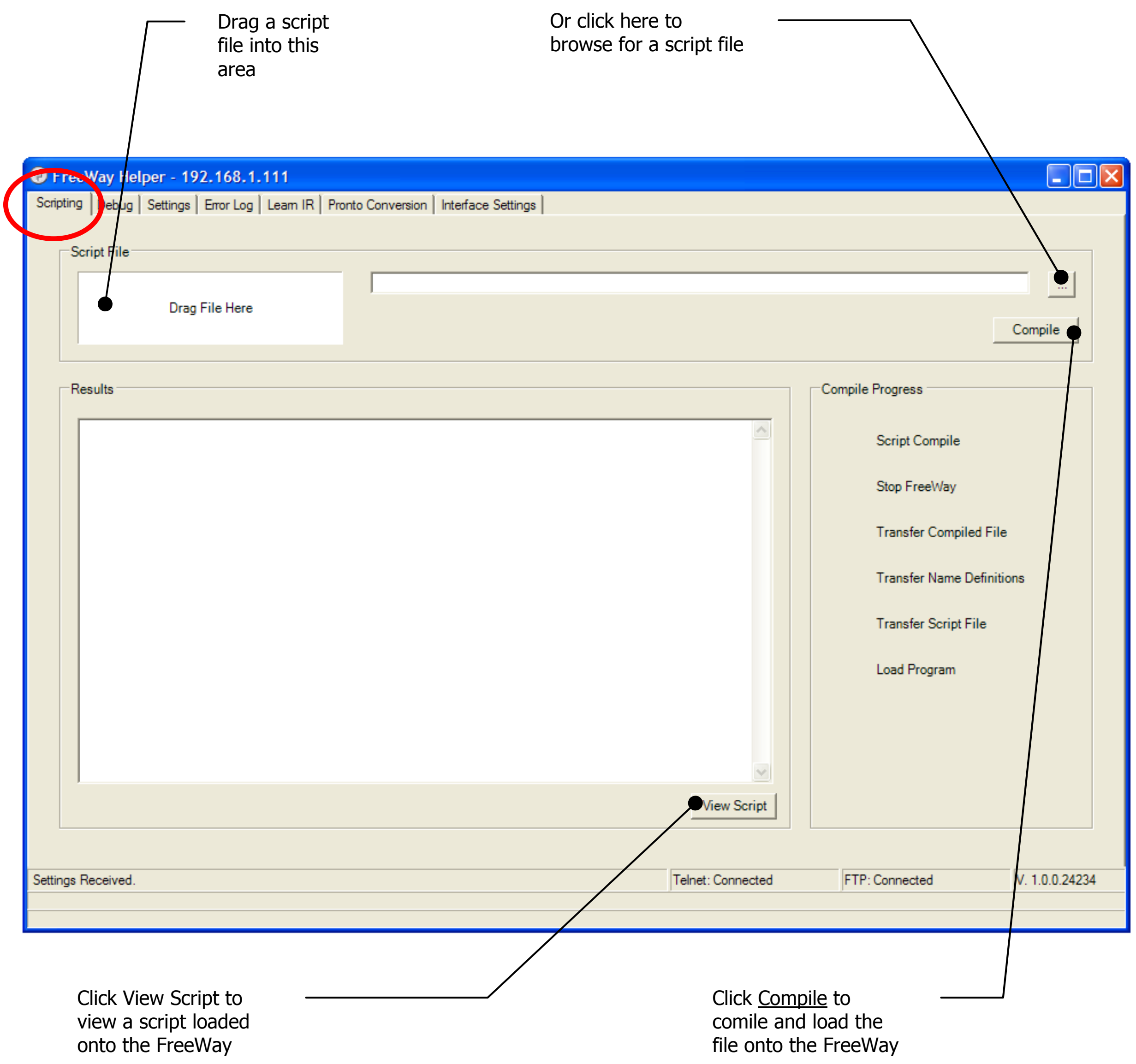
- Notepad is just fine but…
- You may find an editor that display's line numbers useful because if the compiler finds any script errors it'll refer to them by line number
- Be careful if you use Rich Text Editors as they can add hidden control codes into the file that will confuse the compiler. Never use Word.

Here is the script..

```
// First we declare any functions and variables that we're going to use
// They can be declared anywhere in the script
// as long as they're declared before we use them
// Note we don't need to declare the 'built-in' system functions like HubInit()
// ---------------------------------------------------------------------------
Func1();                    // declare a function Func1()
Func2(float fVal);          // declare a function Func2() that’s passed a parameter

string sMessage;            // declare a string to hold a message

// The first function is usually HubInit()
// This is always called when FreeWay powers up
// ---------------------------------------------------------------------------
HubInit()
{
  // print a welcome message to the debug interface
  DebugPrint("Welcome to the FreeWay demo script! \n\r");
}

// Now we'll define a function that we'll call from the FreeWay's
// built-in Web Page
// ---------------------------------------------------------------------------
Func1()
{
  // Print out a debug message
  DebugPrint("Func1 has been called \n\r");
}

// This function demonstrates how to pass a numeric value to the function
// from an external browser - the value is converted into text and printed out
// ---------------------------------------------------------------------------
Func2(float fVal)
{
  // build the message
  sMessage = "You passed value " + format(fVal,1,0) + " to Func2 \n\r";
  // print out the message
  DebugPrint(sMessage)
}
```

Downloading & Compiling a Script

Assuming that you have the above script (or something similar) in a text file somewhere we now need to compile it and download it to the FreeWay. Click on the **Scripting** button.



Click on the Compile button to start compiling the script. You should get a compile error because of the deliberate mistake in the script file.

```
        34 Lines Compiled
        Name memory used: 15% (35/220)
        String memory used: 2% (1/50)
        Number of errors: 1

        Line 34: } expected
```

The last line in `Func2()` is missing a semi-colon. Add the semi-colon and download & compile the script again. You should now get a successful compile. Congratulations.

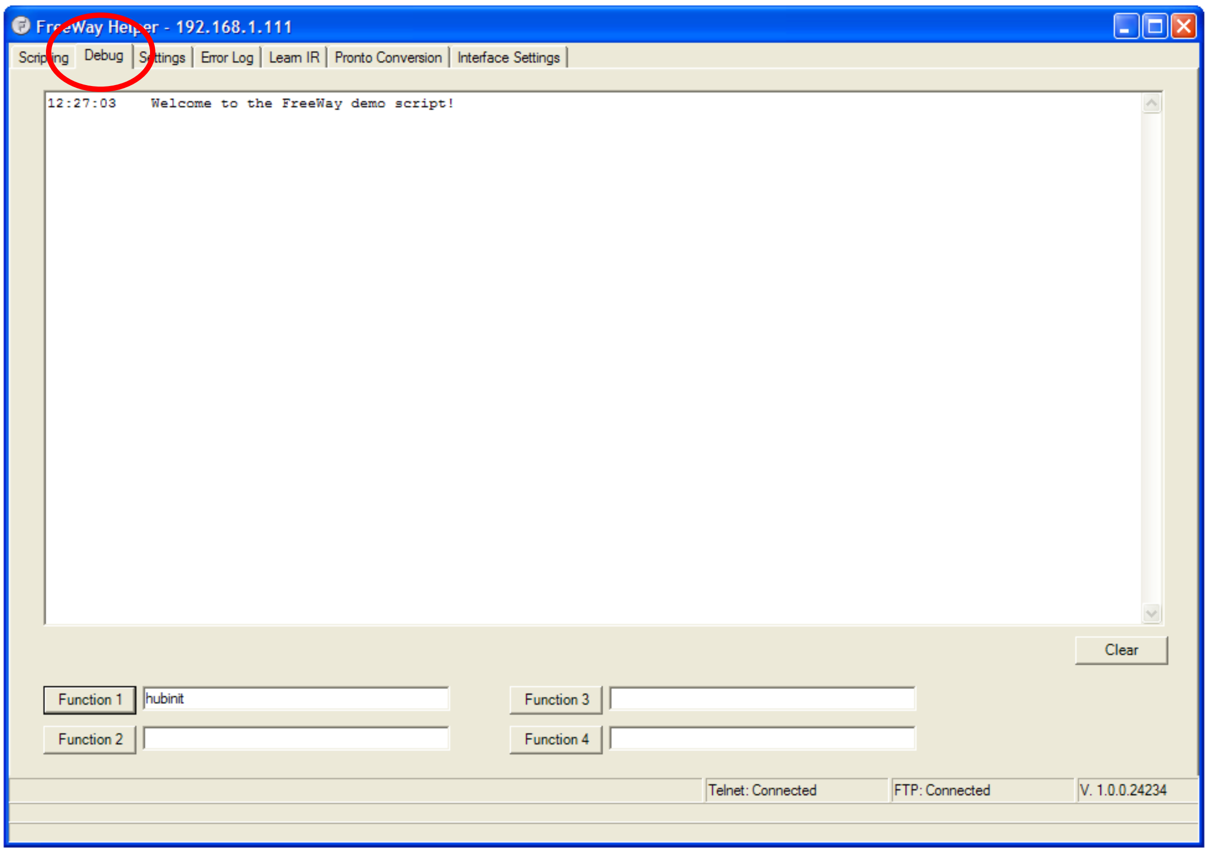
```
        34 Lines Compiled
        Name memory used: 15% (35/220)
        String memory used: 2% (1/50)
        COMPILE SUCCESSFUL!
```

After a successful compile, the script will downloaded automatically to the FreeWay.

Testing the Script

We'll now use the **Debug** page to test the script. Click on the Debug button. Then type in `hubinit` into the text box next to the Function 1 button. Now click on the Function 1 button. In the Debugger window you should see the message:

```
        Welcome to the FreeWay demo script!
```



```
12:27:03    Welcome to the FreeWay demo script!
```

Function 1: hubinit

Telnet: Connected | FTP: Connected | V. 1.0.0.24234

In the Function 2 text box type in `Func1` and click on the Function 2 button. You should see:

```
        Func1 has been called
```

In the Function 3 text box type in `Func2&1234` and click on the Function 3 button. You should see:

```
        You passed value 1234 to Func2
```

In this way you can test all of the functions in your script before installation and commissioning.

Implementing a HTML User Interface (optional)

A user interface is optional because your application may not require one. However, one of the most useful features of the FreeWay is that all of functions that you implement in your script can be accessed using HTML references. In fact all of the functions in your script are treated as separate URLs within the FreeWay. This means that your application can be controlled from a Web Browser.

Application Example

Lets assume that the FreeWay is controlling a Home Theatre setup, integrating control over the AV devices, and being controlled from a web browser interface on some display device. Here's how we would control the DVD Player from the user interface.

You would have an Infra-Red emitter attached to the front of the DVD Player connected to Infra-Red output port IR1, say. Your script would contain a function that looks something like this:

```
//-------------------------------------------------------------
// Function: DVDPlay()
// Job:      Play a DVD
//-------------------------------------------------------------
DVDPlay()
{
  string sIRCommand;

  // the Play IR command for the DVD player
  // this would have been learned earlier using the FreeWay IR Learn
  sIRCommand = "[PF68L836741E0083418C3X42F7BDEFF7FFFFB2F7BDEFF7FFFFB0P2CDFR04]";

  // transmit the command
  SendIR(1,0,0,0,sIRCommand);
}
```

The user interface would doubtless have a button on it called 'DVD Play' and clicking on this button would need to call the `DVDPlay()` function. If our user interface were written in HTML our link would simply be

```
<a href="http://192.168.1.111/run.cgi?DVDPlay">PlayDVD <\a>
```

or more generally, use:

- **`http://`**
- followed by the FreeWay's address
- followed by **`/run.cgi?`**
- followed by the FreeScript function name (without the brackets)

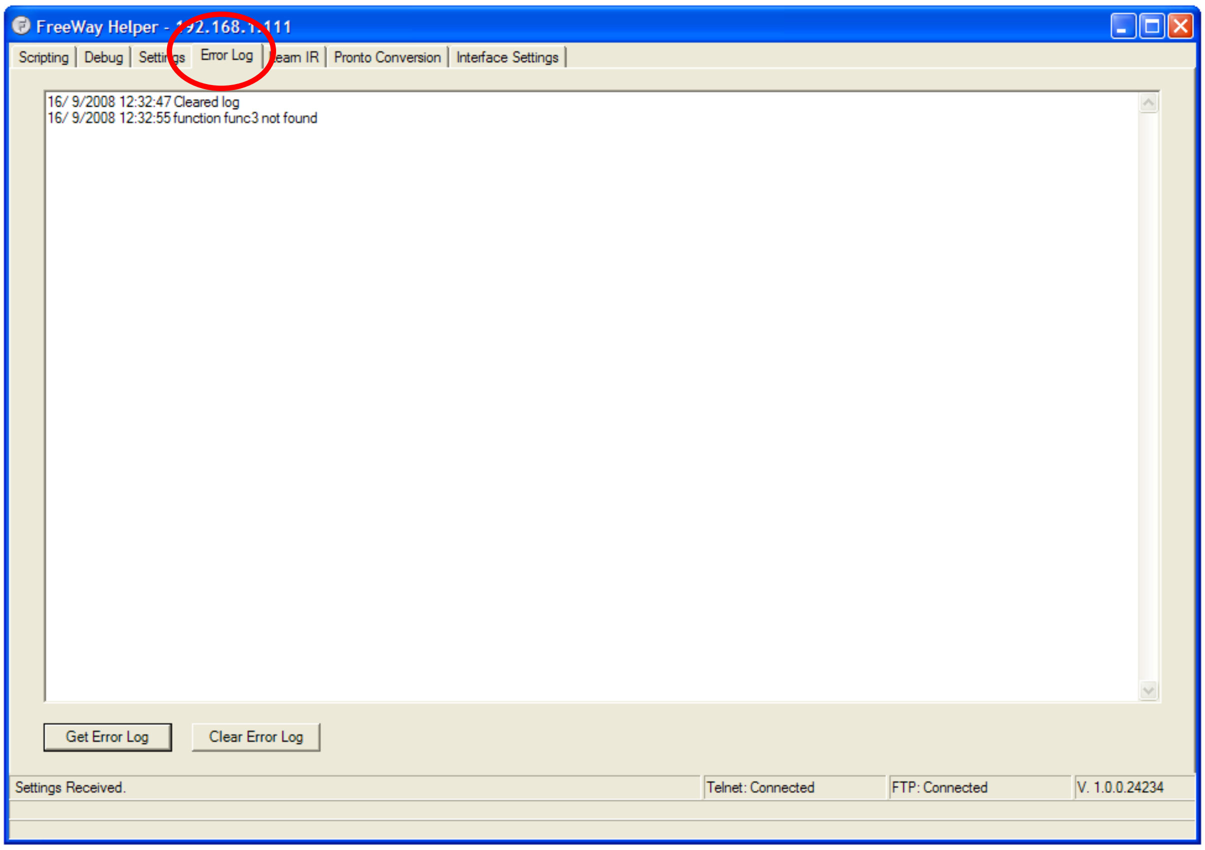
If you want to pass a numeric argument to the function then precede the argument with an ampersand (&) e.g. `PlayDVD&1`. You can pass up to three numeric arguments in this way.

Using the Error Log

The FreeWay will write various status messages and reports into the error log from time to time. If you are experiencing problems with your script it's worth inspecting the log for any unusual messages.

Note that not all messages in the log are problems. For example, there will be entries written when a script is compiled or when the FreeWay boots up.

To view the error log clock on the **Error Log** tab and then click on the Get Error Log buton.



```
16/ 9/2008 12:32:47 Cleared log
16/ 9/2008 12:32:55 function func3 not found
```

Settings Received. | Telnet: Connected | FTP: Connected | V. 1.0.0.24234

For example, if you try and run a function called func3 in the Debug page you will get the above entry in the log because func3 does not exist in the script.

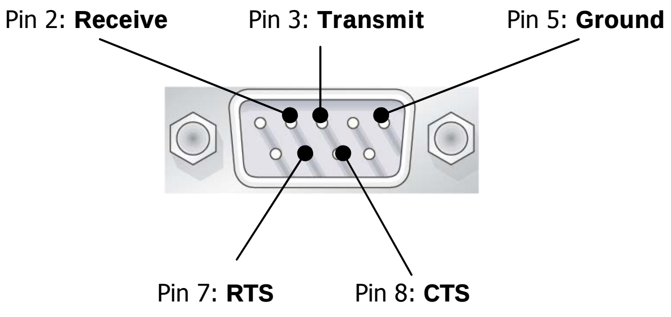
The error log can store aboute 2k bytes of text so it will eventually fill up. Click on the Clear Error Log button to empty the log.

RS-232 Serial Ports

Six serial ports are provided as standard on the FreeWay using 9-way male D-type connectors. These can be used for controlling any 3rd party RS-232 compatible equipment. The FreeScript language provides a set of functions for supporting the RS-232 serial ports. All the popular baud rates, start bits, stop bits and parity settings can be configured using the FreeScript system functions. Functions are also provided for defining message structures to support various communication protocols.

Connector pin assignment

(Male - viewed looking at the rear panel)



All other pins are not connected.

RS232 Port Configuration

Baud Rate

Call the following FreeScript function to configure the baud rate of each the RS-232 ports

```
SetBaud(float fPort, string sBaud)
```

where

`fPort` - the RS232 port from 1 to 6

`sBaud` - the baud rate for the port can be 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bits per second.

Note that this is usually done in the HubInit() function that is the first function to be called when the FreeWay powers up.

Handshaking & Port Settings

Call the following function to configure the handshaking and other port settings for each RS-232 port

```
ConfigPort(float fPort, string sConfig, float fHandshake);
```

where

`fPort` - the RS232 port from 1 to 6

`sConfig`	- a string to configure the number of data bits, parity and the number of stop bits. For example: “`8,n,1`” – 8 data bits, no parity, 1 stop bit

Options are:

Data bits:	`7 or 8`	
Parity:	`e, o, n`	(even, odd, or none)
Stop Bits:	`1 or 2`	

`fHandshake` - a value to enable or disable handshaking. Options are:

0 – no handshaking

1 – hardware handshaking using RTS/CTS

If you want to use the default setting of 8-bits, no parity and one stop bit (which applies most of the time) then you don’t need to call the `ConfigPort()` function.

Example:

```
// ------------------------------------
HubInit()
{
  // configure rs232 port 1 for 9600 baud
  SetBaud(1, “9600”);      // note that the baud rate is in quotes !

  // configure rs232 port 1 for 7 bits, even parity, 1 stop bit, no handshaking
  ConfigPort(1, “7,e,1”, 0);
}
```

RS232 Transmit

Call the following function to transmit data on one of the RS-232 ports

`SerialSend(float fPort, string sOutput)`

where

`fPort`	- the RS232 port from 1 to 6
`sOutput`	- the data to transmit from the port

Example:

```
// transmit a message on serial port 1
// ------------------------------------
SendRS232Command()
{
  string sCommand;                    // declare a string to hold the command
  sCommand = “Hello, World!\n\r”;     // now define the command
  SerialSend(1, sCommand);            // transmit the command on port 1
}
```

Note: the “`\n\r`” are special codes for line-feed and carriage return.

RS232 Receive Functions

The following system event function is automatically called when a message is received from an RS-232 serial port.

```
        SerialReceive(float fPort)
```

where

`fPort` - the RS232 port (from 1 to 6) on which the message was received

To get the message data use the following function:

```
        string SerialGet(float fPort)
```

where

`fPort` - the RS232 port (from 1 to 6) on which the message was received

Example:

```
// receive a message on serial port 1
// -----------------------------------
SerialReceive(float fPort)
{
  // to get here we've received a message on port 'fPort'
  string sMessage;               // declare a string to hold the message

  // now read the message
  sMessage = SerialGet(fPort);
}
```

Note

To be more specific, the FreeWay will call the above `SerialReceive()` function under the following conditions:

- If the defined 'end of message' character is received (see below)
- If the number of characters exceeds a defined message length (see below)
- If the time between characters exceeds the message timeout value (see below) – the default is 100ms.
- If the number of characters exceeds 255.

Defining Messages

Defining an End-of Message Character

Many messages will always end in a certain character, for example, a 'carriage return' for text based communications. Use the following function to specify it. You'd probably want to include this function in the HubInit() function.

```
        SetEndOfMsg(float fPort, float fEom)
```

where

`fPort` - the RS232 port (from 1 to 6) to configure

`fEom` - the 'end of message' value. For example, a carriage return value would be '13'

Example:

```
// RS232 port 2 message ends in a carriage return (which is decimal 13)
SetEndOfMsg(2,13);
```

Defining a Message Length

Some messages will always be a fixed length. Use the following function to specify it. You'd probably want to include this function in the HubInit() function.

```
        SetMsgLength(float fPort, float fLength)
```

where

`fPort`	- the RS232 port (from 1 to 6) to configure
`fLength`	- the length of the message, up to 255 characters. 0 disables this feature & the default value of 255 characters is used.

Example:

```
// RS232 port 3 message length is 24 characters
SetMsgLength(3,24);
```

Defining a Message Timeout

You can delimit message boundaries by specifying timeout value. If no characters are received within this time period the SerialReceive() function will be called. You'd probably want to include this function in the HubInit() function.

```
        SetMsgTimeout(float fPort, float fTime)
```

Where

`fPort`	- the RS232 port (from 1 to 6) to configure
`fTime`	- timeout period of a message in milliseconds (0-10,000). Default is 100ms.

Example:

```
// RS232 port 4 message timeout is 1 second
SetMsgTimeout(4,1000);
```

RS-485 Ports

There are three RS-485 ports fitted to FreeWay

- Two on 3-way screw terminals – labelled Port 7 & Port 8 on the rear panel
- One on an RJ45 connector – labelled RS485 on the rear panel

The RJ45 connector provides an additional 9V DC line for remotely powering equipment. It can source up to 100mA.

These ports can be used for communicating with any 3rd party RS-485 and RS-422 compatible equipment and networks in 2-wire (half-duplex RS-485) and 4-wire (full-duplex RS-485 and RS-422) modes. Depending on the baud rate, cable capacitance and network load, the RS-485 interfaces can usually drive cable lengths of up to 4000 feet. They provide a standard 12kΩ input impedance and can drive a bus of up to 32 standard load devices.

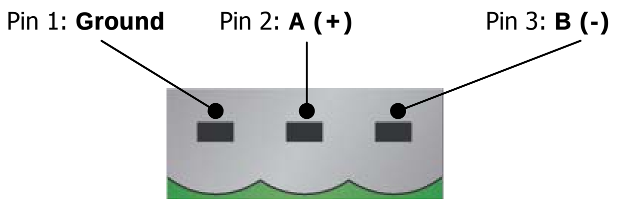
The FreeScript language provides a set of functions for supporting the RS-485 serial ports. All the popular baud rates, start bits, stop bits and parity settings can be configured using the FreeScript system functions. Functions are also provided for defining message structures to support various communication protocols.

The FreeScript functions for RS-485 ports are the same as those for RS-232 but differ in the following ways:

- The port parameter is 7 & 8 for the two 3-terminal ports (labelled Port 7 & Port 8)
- The port parameter is 9 for the RJ45 connector port
- Supported baud rates are 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 and 230400
- Handshaking is not supported

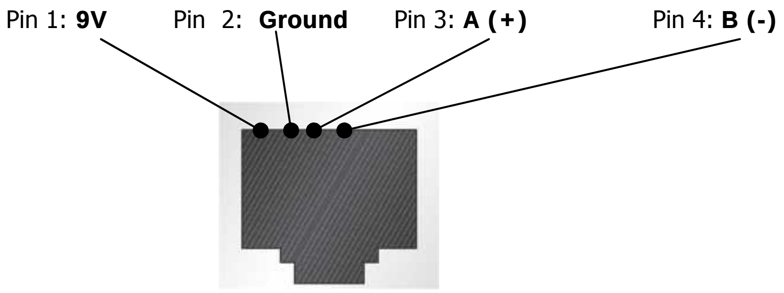
Port 7 & 8 pin assignment

Viewed looking at the rear panel



RJ45 Connector pin assignment

Viewed looking at the rear panel



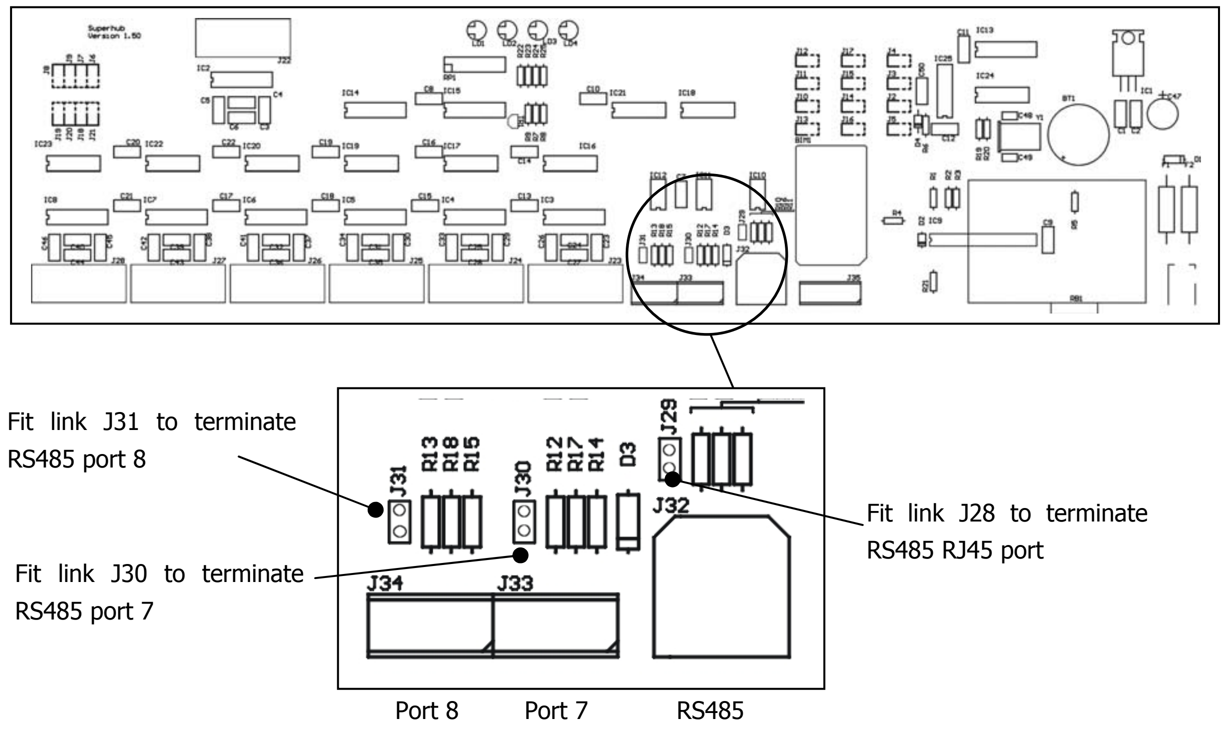
All other pins are not connected.

Termination resistors

For RS-485 connections with long cable lengths and high baud rates (>100m at 115kbaud say), and if the FreeWay is at the end of the cable, the RS-485 data lines may need to be terminated. This is done by removing the FreeWay's lid and fitting jumpers links to add termination resistors into the circuit. The default setting is for termination with the links connected, i.e. terminated.

Termination Resistor Jumper Links

The disable the line termination resistors remove the lid of the FreeWay and remove the links as shown below:


Fit link J31 to terminate RS485 port 8
Fit link J30 to terminate RS485 port 7
Fit link J28 to terminate RS485 RJ45 port
J31
R13
R18
R15
J30
R12
R17
R14
D3
J29
J32
J34
J33
Port 8
Port 7
RS485

RS-485 and RS-422 modes

RS-485 2-wire mode

In this mode you have 2 half-duplex RS485 ports available on ports 7 & 8. The FreeWay can act as a Master or Slave device.

RS-485 4-wire mode

This mode would typically use ports 7 & 8 together to form a 4 wire interface. It's up to you to decide which terminals transmit and which receive. The FreeWay can act as a Master or Slave device.

RS-422 mode

This mode would typically use ports 7 & 8 together to form an RS-422 interface. It's up to you to decide which terminals transmit and which receive. The FreeWay can act as a Master or Slave device. The RS-485 drivers used in FreeWay meet all the required specifications for RS-422 communications.

Infra-Red Receiver

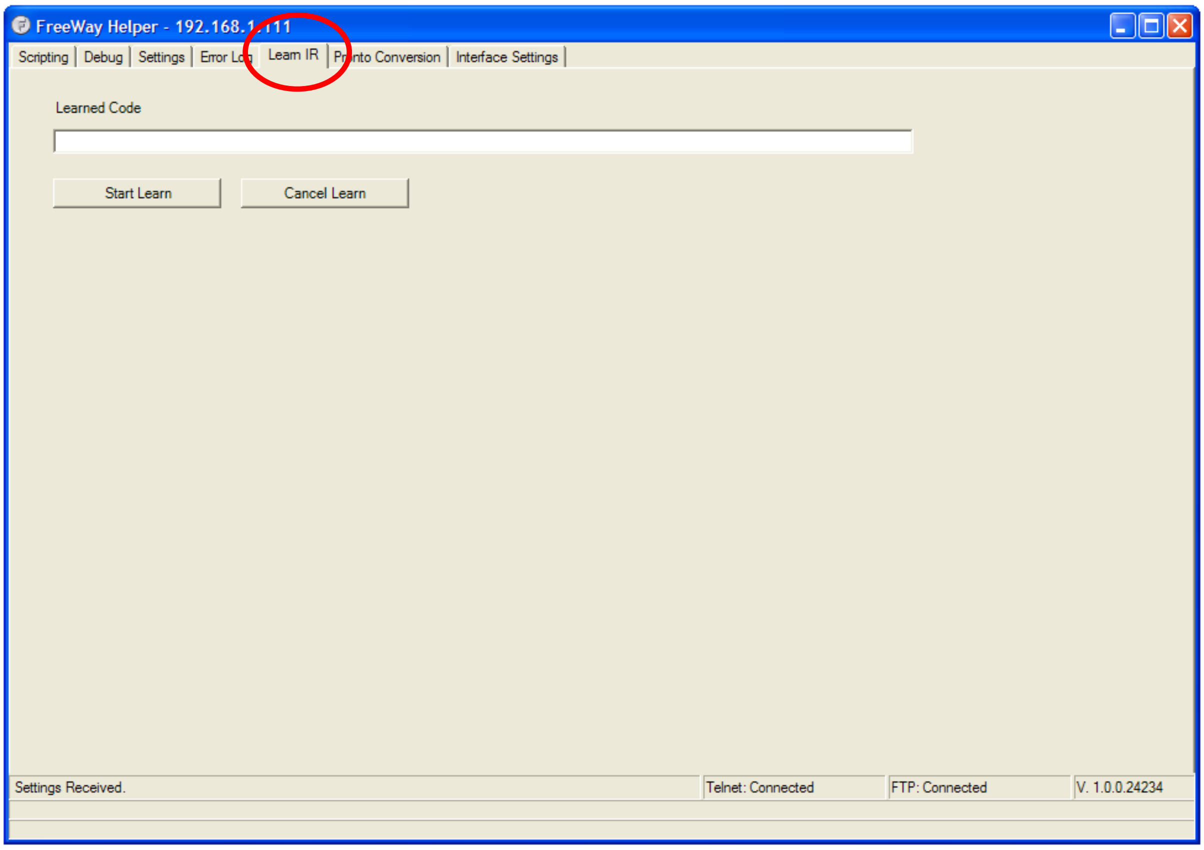
An Infra-Red receiver is provided on the Front Panel so that Infra Red commands from the Infra-Red Remote Controls of various AV devices can be sampled and learned. These commands can then be used in a FreeWay script to control AV devices via the Infra-Red Output Ports.

FreeWay can also decode RC5 Infra-Red commands and use them to perform other functions in your script. Received RC5 commands are handled in a similar way to messages on RS232, RS485 and Telnet ports. This effectively gives you a very powerful tool for controlling A/V equipment from programmable remote controls such as the Philips Pronto.

The FreeWay Interface program also provides a utility to convert Philips Pronto IR codes into FreeWay compatible codes. This is described at the end of this chapter.

Learning Infra-Red Commands

Connect to FreeWay via FreeWay Interface and click on the **IR Learn** tab.



Settings Received. | Telnet: Connected | FTP: Connected | V. 1.0.0.24234

Click on the **Start Learn** button. The LED 2 on the FreeWay front panel will then turn on indicating that the FreeWay is waiting for an IR command (unless this LED is being used in your script).

Press the button once on the Remote Control whose Infra-Red command you want to learn. LED 2 will then turn off to indicate a successful command capture. The IR command will be displayed in the window. Copy and paste all of this command (including the square brackets) into your Script or a text file (Select the command using the mouse, right click the mouse button and select Copy).

To learn another Remote Control command click on the **IR Learn** button again. To cancel a Learn operation click on the **Cancel Learn** button.

Hints: Remember to point the Remote Control at the Infra-Red Receiver The distance between the receiver and the Remote Control should not be greater than 10cm. Also beware of learning IR commands in direct sunlight, or close to Plasma displays and fluorescent lights. Always use new batteries in your remote handset.

Here's an example of a Script function that will send a learned command out of an Infra-Red port:

```
// transmit an IR command on IR1 port
// ---------------------------------
TransmitInfraRed()
{
  string sIRCommand;       // declare a string to store the command

  // IR Play command captured from a JVC VCR remote control
  sIRCommand = "[PF68L839841F7081E18E4X42F7FFBF7BF7FFB2F7FFBF7BF7FFB0P2D1ER03]”;

  // transmit the command
  SendIR(1,0,0,0,sIRCommand);
}
```

Editing Infra-Red Commands

If you look at a learned IR command you will see that the last few characters look like this

```
…R03]
```

The number after the 'R' denotes the number of times that the IR command is repeated, in this case 3 times. The number of repeats is in hexadecimal format. You can change the repeat value to a different value if you need to. Note that the repeat value is the only part of the IR command that you should edit.

Decoding RC5 Infra-Red Commands

The FreeWay has the ability to decode Infra-Red commands which conform to the RC5 protocol. This Infra-Red format is used by all Philips products and many other manufacturers such as Linn, Meridian and Arcam. RC5 is a common interface and is quite simple to decode and interpret. FreeWay's RC5 decoding is a very powerful and flexible method to control all sorts of A/V, heating and lighting control from a programmable Infra-Red handset.

The RC5 protocol is split into to 3 fields:

address	with 32 values from 0 to 31
data	with 64 values from 0 to 63
toggle	this is a flag which changes state when a new button is pressed on the handset. This is used for detecting if a key is being held down – for example for a volume control

When the FreeWay detects an RC5 command it will automatically call the following system event function:

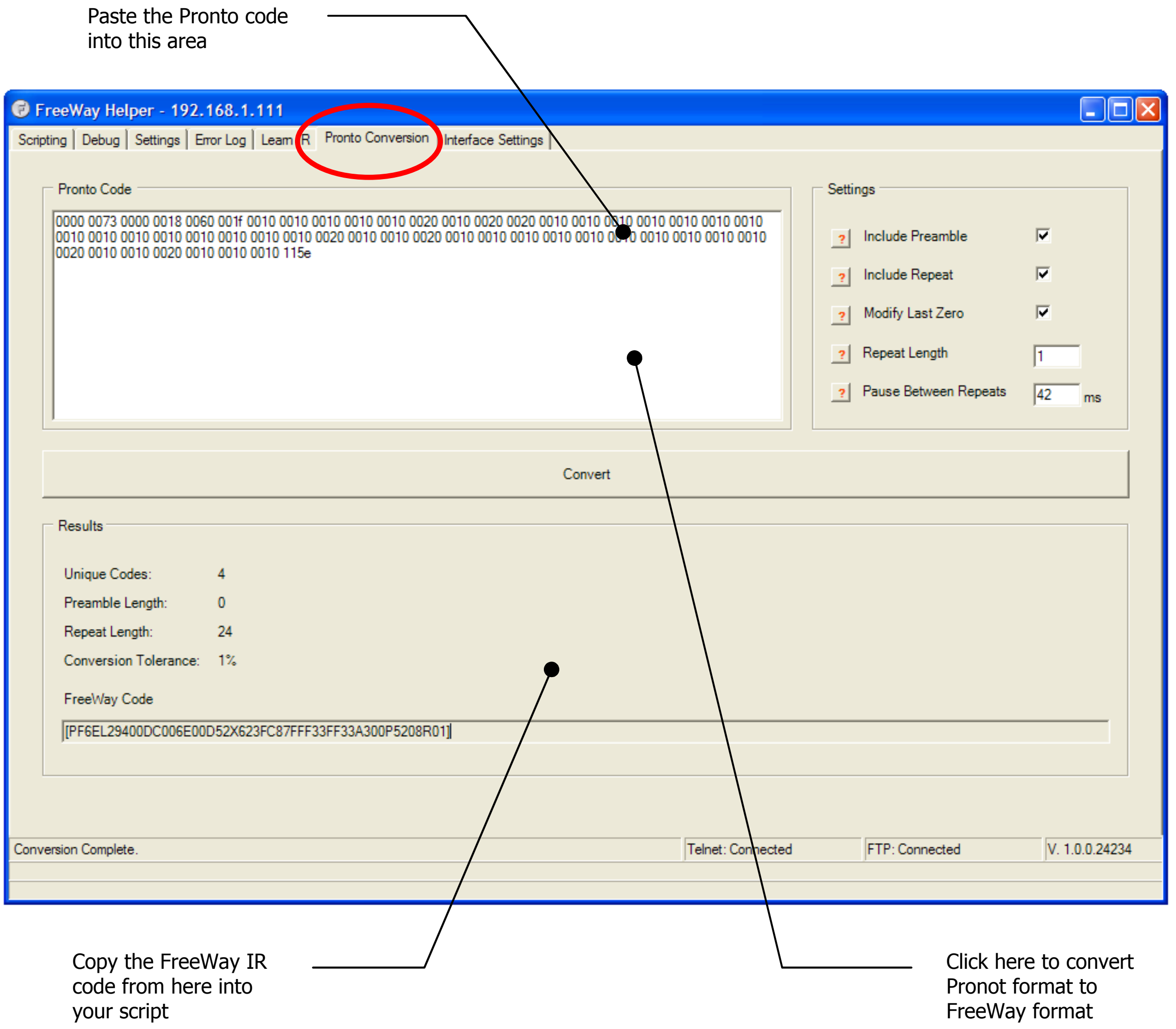
```
IRReceive(float fAddress, float fData, float fToggle)
```

You can then inspect the passed address, data and toggle fields and perform script processing as required. For example the following script extract will send an RS232 message on port 1 when it receives an RC5 command with an address field of 31 and a data field of 20.

```
IRReceive(float fAddress, float fData, float fToggle)
{
  // check the address field first - we will respond to address 31
  if (fAddress == 31)
  {
    // check the data field next - we will respond to data value 20
    if (fData == 20)
    {
      // Send out a message on RS232 port 1
      SerialSend(1,"Got RC5 Address 31 Data 20 \n\r");
    }
  }
}
```

Converting Philips Pronto IR Codes

The Freeway Interface software provides a utility for converting Philips Pronto codes into a FreeWay compatible format. Click on the **Pronto Conversion** tab:



Paste the Pronto code into this area

Copy the FreeWay IR code from here into your script

Click here to convert Pronot format to FreeWay format

FreeWay Helper - 192.168.1.111

Pronto Code

```
0000 0073 0000 0018 0060 001f 0010 0010 0010 0010 0010 0020 0010 0020 0020 0010 0010 0010 0010 0010 0010 0010
0010 0010 0010 0010 0010 0010 0010 0010 0020 0010 0010 0020 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010
0020 0010 0010 0020 0010 0010 0010 115e
```

Settings

Include Preamble	✓
Include Repeat	✓
Modify Last Zero	✓
Repeat Length	1
Pause Between Repeats	42 ms

Results

Unique Codes:	4
Preamble Length:	0
Repeat Length:	24
Conversion Tolerance:	1%

FreeWay Code

```
[PF6EL29400DC006E00D52X623FC87FFF33FF33A300P5208R01]
```

Conversion Complete. | Telnet: Connected | FTP: Connected | V. 1.0.0.24234

Infra-Red Outputs

Four fully matrixed Infra Red ports are provided. These are suitable for connection to any standard Infra-Red emitters (single or dual, blinking or non-blinking) fitted with mono mini-jack plugs (e.g. Xantech 282M IR Mouse Emitters). Infra-Red commands can be transmitted an a single port or simultaneously on multiple IR ports. Independent transmission is very useful for separately controlling identical pieces of equipment, for example 2 or 3 satellite receivers of the same model.

The Infra-Red commands that are transmitted must be captured using the FreeWay's built-in Infra-Red receivers & Infra-Red Learn facility.

Infra-Red Port Transmission

The following FreeScript function is provided for transmitting Infra-Red commands:

```
SendIR(float fIR1, float fIR2, float fIR3, float fIR4, string sIROut)
```

where

`fIR1` - set to 1 to transmit the IR command on port IR1, set to 0 otherwise

`fIR2` - set to 1 to transmit the IR command on port IR2, set to 0 otherwise

`fIR3` - set to 1 to transmit the IR command on port IR3, set to 0 otherwise

`fIR4` - set to 1 to transmit the IR command on port IR4, set to 0 otherwise

`sIROut` – the IR command to send to the IR port(s)

Example:

```
// transmit an IR command on IR1 & IR3 ports
// --------------------------------------------
TransmitInfraRed()
{
  string sIRCommand;       // declare a string to store the command

  // IR Play command captured from a JVC VCR remote control
  sIRCommand = "[PF68L839841F7081E18E4X42F7FFBF7BF7FFB2F7FFBF7BF7FFB0P2D1ER03]";

  // transmit the command
  SendIR(1,0,1,0,sIRCommand);
}
```

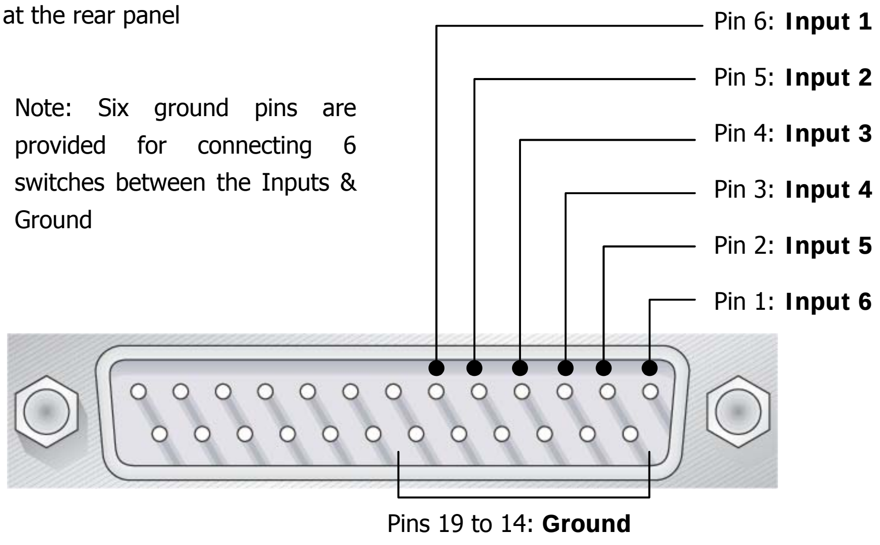
Digital Inputs

Six opto-isolated Digital Inputs are provided on the FreeWay. These are presented on the 25-way female D-Type connector on the Rear Panel. These can be typically used with switch contact closures to trigger script functions which can then control other pieces of equipment.

Connector pin assignment

Viewed looking at the rear panel

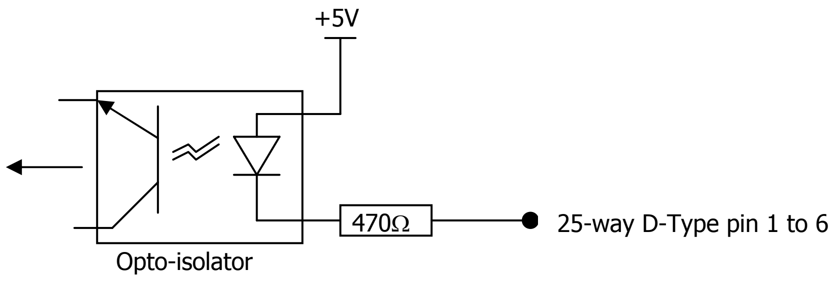
Note: Six ground pins are provided for connecting 6 switches between the Inputs & Ground



See the Digital Outputs section for the connections of the other pins

Electrical Interface

The equivalent electrical circuit for each digital input is as follows:



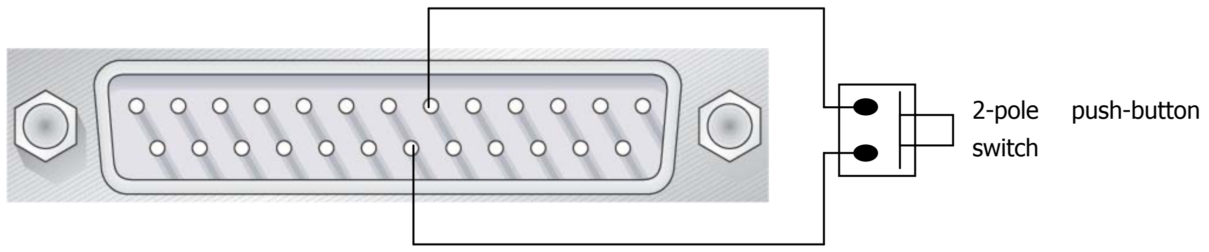
With the Digital input pin unconnected the opto-isolator transistor is switched off and a LOW state is read by the microprocessor.

With the Digital input pin shorted to Ground current (approx. 10mA) flows through the opto-isolator LED which turns the transistor on and a HIGH state is read by the microprocessor.

Whenever the Digital Input state changes the FreeScript `DigitalIn()` function is automatically called.

Basic Interface Example

The following diagram shows a simple 2 pole switch connected between Digital Input 1 and Ground.



Contact DKT for advice on other Digital-Input configurations you may require.

FreeScript Functions

The following FreeScript functions are available for handling digital input events.

```
        DigitalIn(float fPort)
```

where

`fPort` - the Digital Input port pin from 1 to 6

This system event function will automatically be called whenever a change in pin status is detected. Note that the function only reports a change on a single pin. If multiple pins change simultaneously the `DigitalIn()` function will be called for each pin change in numerical order (i.e. from pin 1 to pin 6).

Once the DigitalIn() function has been called the following system function is then provided to get the state of the port pin:

```
        float GetDin(float fPort)
```

where

`fPort` - the Digital Input port pin from 1 to 6

The function will return a value of 1 or 0 depending on the state of the port pin. Note that this function can be called at any time within any function to get the port pin state. You should also call the `GetDin()` function as soon as possible in the `DigitalIn()` function because `GetDin()` returns the current state of the pin not the state of the pin when the `DigitalIn()` function was triggered.

Example:

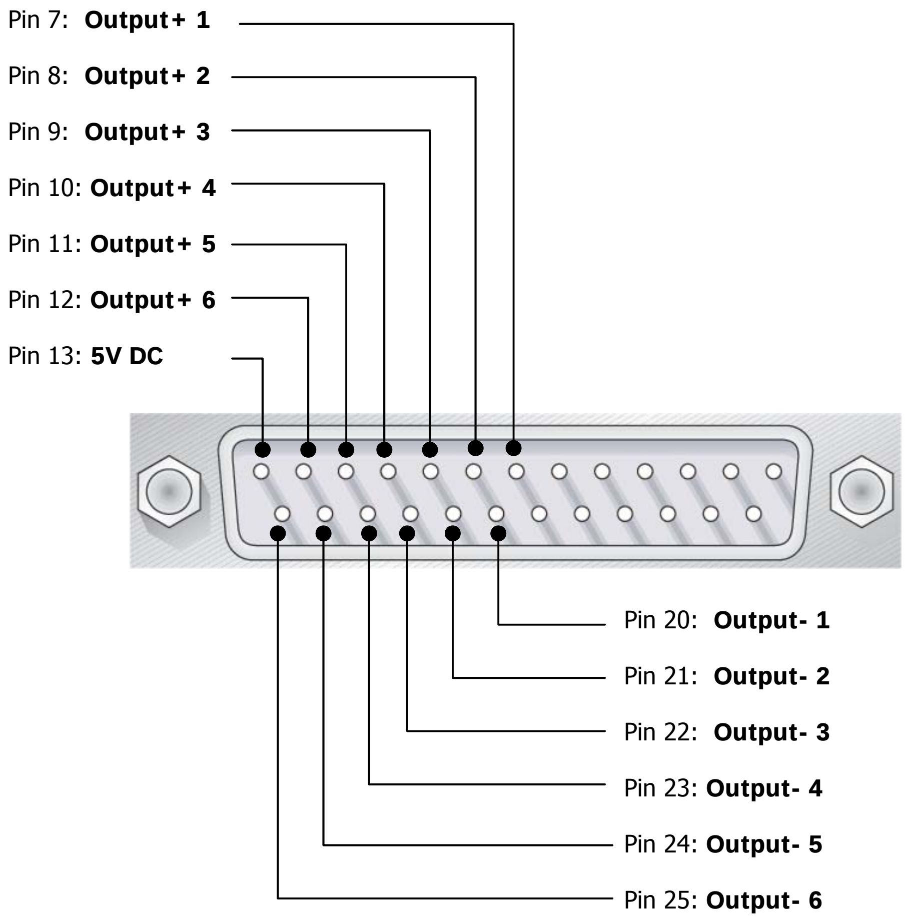
```
// Called when a Digital Input port pin changes state
// -----------------------------------------------
DigitalIn(float fDigitalInPort)
{
  // declare a variable to hold the pin state
  float fDigitalInState;

  // read the state of the digital input
  fDigitalInState = GetDin(fDigitalInPort);

}
```

Digital Outputs

Six opto-isolated Digital Outputs are provided on the FreeWay. These are presented on the 25-way female D-Type connector on the Rear Panel. These can be typically used to control external equipment with a binary or relay type control inputs from events on other ports using the script functions.

Connector pin assignment

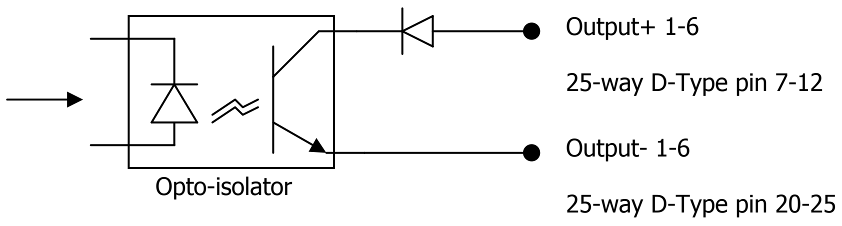
Viewed looking at the front panel.



See the Digital Inputs section for the connections of the other pins.

Electrical Interface

The equivalent electrical circuit for each digital output is as follows:



When the microprocessor sets the Digital input to a 1 (ON) the output transistor is turned on and current will flow through it.

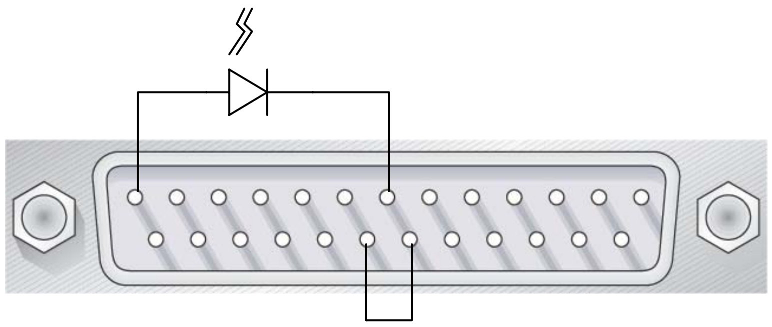
When the microprocessor sets the Digital input to a 0 (OFF) the output transistor is turned off and no current will flow through it.

Current will only flow through the output transistor from Output+ to Output- due to the blocking diode. The transistor will pass current up to a limit of 10mA.

A courtesy voltage of 5V DC is available on pin 13 of the D-Type connector. You should not draw more than 100mA from this pin.

Basic Interface Example

The following diagram shows how an LED can be turned on when Digital Output 1 is turned on.



When the digital output transistor #1 turns on 10mA of current will flow

from the 5V courtesy voltage

through the LED – turning it on

into Output+1 pin through the output transistor

out of the Output-1 pin and into the Ground pin.

Contact DKT for advice on other Digital-Output configurations you may require.

FreeScript Functions

The following FreeScript functions are available for handling Digital Outputs.

```
SetDout(float fPort, float fState)
```

where

`fPort` - the Digital Output port pin from 1 to 6

`fState` - the state of the output pin (0=OFF, 1=ON)

Example:

```
// General purpose function to
// change the state of a Digital Output
// -----------------------------------
SetDigitalOutput(float fDoutPort, float fDoutState)
{
  SetDout(fDoutPort, fDoutState)
}
```

Alarms

The FreeWay has a Real-Time clock which keeps track of the current time and date and provides the facility for triggering time & date based events – or alarms. There are eight alarms available which can be triggered on a daily, weekly or one-off basis. There is also an astronomic clock function which can trigger alarms at dusk & dawn.

Setting Up Alarms

The following FreeScript function is used to configure an alarm. You would typically include this function in the `HubInit()` function.

```
        SetAlarm(float fAlarm, string sConfig)
```

where

`fAlarm`	- the number of the alarm you are configuring, from 1 to 8
`sConfig`	- the alarm configuration string. Options are:

`“O dd/mm/yyyy hh:mm”`	one time alarm at this time and date
`“D hh:mm”`	daily alarm at this time
`“W dd/mm/yyyy hh:mm”`	weekly alarm at this time from this date
`“A +/-hh:mm”`	at dawn + or – an offset
`“P +/-hh:mm”`	at dusk + or – an offset

When using the astronomic clock you also need to call the SetLocation() function. This function is used to specify the FreeWay’s location so that it can calculate the dawn and dusk alarm times. You can obtain these values from any online map service, for example: www.multimap.com.

```
        SetLocation(float fLat, float fLon)
```

where

`fLat`	the Latitude value
`fLon`	the Longitude value

Example

```
// configure some alarms
HubInit()
{
  // setup alarm 1 as a one-off on 27th November 2004 at 9:00am
  SetAlarm(1,”O 27/11/2004 09:00”);

  // setup alarm 2 as a daily alarm at 5:30pm
  SetAlarm(2,”D 17:30”);

  // setup alarm 3 as a weekly from 1st January 2005 at 7:30am
  SetAlarm(3,”W 01/01/2005 07:30”);

  // setup a dawn alarm set for 1 hour after dawn
  // The location is London’s Oxford Street
  SetLocation(51.51, -0.148);
  SetAlarm(4, “A +01:00”);

}
```

Responding to Alarms

When an alarm is activated the following system event function will automatically be called:

```
Alarm(float fAlarm)
```

where

`fAlarm`	- the number of the alarm from 1 to 8

Example

```
// assume the alarms have been setup as in the above example
Alarm(float fAlarm)
{
  string sMessage;

  if (fAlarm == 1) {
    sMessage = “Its my birthday!\n\r”
  }
  if (fAlarm == 2) {
    sMessage = “Time to go home!\n\r”
  }
  if (fAlarm == 3) {
    sMessage = “Its Monday\n\r”
  }
  if (fAlarm == 4) {
    sMessage = “Good Morning\n\r”
  }
  // print out the message to the debug terminal
  DebugPrint(sMessage);
}
```

Telnet Ports

There are four Telnet ports built-in to the FreeWay which you can use for communicating with equipment over a TCP/IP Ethernet connection. In the script these are effectively treated as four serial ports but use the Ethernet port. These ports are typically used for 2-way Ethernet communications between :

- multiple FreeWays
- multiple FreeWays and TCP/IP sockets built into Macromedia Flash Actionscript
- FreeWay and TCP/IP sockets built into Ethernet enabled A/V devices (e.g. CD Servers, Projectors etc)

Telnet ports can operate in standard Telnet mode or in raw TCP mode. There are script functions for opening, closing & configuring Telnet Ports.

Opening Telnet Ports

Before the FreeWay uses a Telnet port (or another devices uses a FreeWay Telnet port) you need to open it up. You would do this with the OpenTelnet() function, probably included in the `HubInit()` function, as follows:

```
OpenTelnet(float fPort, float fCS, float fIPport, string sIPaddress)
```

where

`fPort`	- the port number of the Telnet port we want to open: 10, 11, 12, or 13
`fCS`	- specifies the type of Telnet port to open: 0 or 1 0 – Telnet Client 1 – Telnet Server
`fIPport`	- the TCP/IP port number of the Telnet connection. This is determined by your application, or specified by the device you are trying to connect to. For example a Xiva CD server's port number is 6789.
`fIPaddress`	- the IP address of the port to connect to. If the FreeWay's Telnet port is is a server then just leave this blank i.e. "". If the FreeWay's Telnet port is a client then specify the IP address of the Telnet server you are connecting to, e.g. "192.168.7.112".

Closing Telnet Ports

FreeWay also provides a close Telent function, as follows:

```
CloseTelnet(float fPort)
```

where

`fPort`	- the port number of the Telnet port we want to close: 10, 11, 12, or 13

There are a number of situations where you may want to use `closeTelnet()`. Firstly, some network devices require that you open a Telnet connection, communicate and then close a Telnet communication.

Also, FreeWay typically stays switched on 24 hours a day, 7 days a week. If you have a Telnet connection to a device and the network goes down, or the devices reboots, then the Telnet connection may not be valid anymore. In this case you may want to open & close telnet connections each time you want to communicate so that you know that a valid connection has been made.

An additional advantage of closing Telnet ports is that it lets you communicate with an unlimited number of network devices. You can open a connection to one device, close it & then open a connection to another device. You can only have four *simultaneous* connections though.

TCP Mode

Most devices implement a full Telnet interface for network communications. Some devices require a raw TCP connection. This is similar to Telnet but does not support various additional control characters & handshaking. If you want to use a Telnet port in raw TCP mode then use the `ConfigTelnet()` function as follows:

```
ConfigTelnet(float fPort, float fMode)
```

where

`fPort`	- the port number of the Telnet port we want to configure: 10, 11, 12, or 13
`fMode`	- 0 is Telent mode – which is default
	- 1 is Raw TCP mode

Note that if you want raw TCP mode you must call this function before caling OpenTelnet(). If you want standard Telent mode you don't need to call this function (unless you are changing a connection on a FreeWay port between TCP & Telnet modes).

Example:

```
HubInit()
{
  // Open a raw TCP client port
  // on 192.168.7.27 port 6789
  // ------------------------------------------
  ConfigTelnet(10, 1);
  OpenTelnet(10, 0, 6789, “192.168.7.27”);


  // Open a Telnet server port
  // on 192.168.7.112 port 10000
  // ------------------------------------------
  OpenTelnet(11, 1, 10000, “”);
}
```

Using Telnet Ports

FreeWay treats Telnet ports just like normal serial ports so the `SerialSend()` and `SerialReceive()` functions can be used to receive and transmit messages over the Telnet connections that you make. The `SetEndOfMsg()`, `SetMsgLength()` and `SetMsgTimeout()` functions can also be used if required.

Example:

```
HubInit()
{
  // Open a Telnet client port
  // on 192.168.7.27 port 6789
  // ---------------------------------------------
  OpenTelnet(10, 0, 6789, “192.168.7.27”);

  // Wait one second to allow the device port to initialise
  // This is optional
  // ----------------------------------------------------
  Delayms(1000);

  // Send a hello message to the Kivor
  // --------------------------------
  SerialSend(10, ”Hello\n\r”);
}

SerialReceive(float fPort)
{
  // test if the message is from Kivor
  // --------------------------------
  if (fPort == 10)
  {
    DebugPrint(“Got a message from Kivor\n\r”);
  }
  else
  {
    DebugPrint(“Got a message from someone else\n\r”);
  }
}
```

UDP Ports

There are four UDP ports built-in to the FreeWay which you can use for communicating with equipment over an Ethernet connection. In the script these are effectively treated as four serial ports but use the Ethernet port. These ports are typically used for 2-way Ethernet communications between :

- multiple FreeWays
- FreeWay and UDP sockets built into Ethernet enabled A/V devices

UDP connections are similar to TCP & Telnet but, whereas TCP guarantees that a message will get through, UDP doesn't. Under situations of high network traffic it may be possible for UDP messages to get lost. Therefore, if it is important to your application that messages don't get lost, then you need to implement a retry mechanism into your application.

In addition, unlike TCP, UDP is a connectionless protocol - which means that we don't need a closeUDP() function.

An advantage of UDP messages is their speed. Because they don't have the retry overhead of TCP messages they typically get through more quickly. UDP also has a broadcast mechanism which allows a message to be sent to all interested parties on the network. This is useful for communications between a large number of FreeWays on a network.

FreeWay provides script functions for configuring UDP connections as follows.

Configuring & Opening UDP Ports

This function tells the FreeWay to listen for UDP messages on a particular UDP port.

```
OpenUDP(float fPort, float fListenPort)
```

where

fPort	- specifies one of four the Freeway port numbers. This can be 14, 15, 16 or 17.
fListenPort	- specifies the UDP port that FreeWay will listen to. E.g. 3456.

This opens a UDP port which is set to listen for UDP messages on a particular port number. Received messages can be handled in the `SerialReceive()` function as normal.

If you want to send UDP messages then you need to call the following function:

```
ConfigSendUDP(float fPort, string sRemoteIP, float fDestPort)
```

where

fPort	- specifies one of four the Freeway port numbers. This can be 14, 15, 16 or 17.
sRemoteIP	- the IP address of the remote UDP port we want to send to.
	An IP address of "255.255.255.255" will broadcast to all UDP listeners.
fDestPort	is the UDP port number at the destination

This function tells the FreeWAy where to send the UDP messages (which IP address, or all IP addresses for a broadcast), and to which UDP port to address the message. You can then send UDP messages using the `SerialSend()` function as normal.

Using UDP Ports

FreeWay treats UDP ports just like normal serial ports so the `SerialSend()` and `SerialReceive()` functions can be used to receive and transmit messages over the Telnet connections that you make. The `SetEndOfMsg()`, `SetMsgLength()` and `SetMsgTimeout()` functions can also be used if required.

Example:

```
HubInit()
{
  OpenUDP(14, 5600);
  ConfigSendUDP(14, “192.168.7.110”, 5600);
  SerialSend(14, “Hello”);
}


SerialReceive(float fPort)
{
  // test if the message is from UDP
  // --------------------------------
  if (fPort == 14)
  {
    DebugPrint(“Got a message from UDP\n\r”);
  }
  else
  {
    DebugPrint(“Got a message from someone else\n\r”);
  }
}
```

EIB/KNX Interface

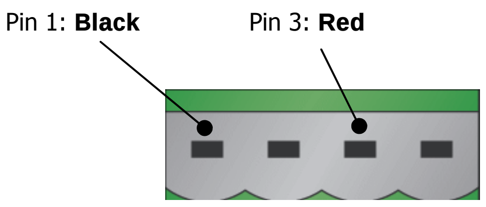
The EIB port allows the FreeWay to send and receive messages to groups on a EIB/KNX bus. Support is provided for all of the EIB data types. Applications include:

- Controlling A/V equipment from EIB switches and sensors
- Control of EIB devices such as lighting dimmers and heating actuators from web browser-based user interfaces
- Control of EIB devices from an Infra Red handset
- Schedule astronomic time and date events using FreeWay's alarm functions
- EIB time synchronisation from FreeWay's internal clock
- Perform logic operations

EIB Port Connector pin assignment

FreeWay connection directly to the bus is via a 4-way terminal connector.

(Viewed looking at the rear panel)



All other pins are not connected.

Configuring ETS

Under most circumstances you do not need to make any configuration changes to your ETS project to communicate with FreeWay, particularly if FreeWay exists on the same line as the devices you wish FreeWay to interact with. Simply allocate FreeWay an unused physical address.

However, for systems with multiple lines, where FreeWay needs to communicate with devices across a line coupler you'll need to add a dummy device to your ETS project, and add objects from the dummy devices to the Groups that FreeWay needs to communicate with. For very large systems, because FreeWay 'listens' to all bus messages, you may also want to assign a FreeWay to its own line.

Gira provide two dummy devices with their product database, for example. The dummy device's physical address is effectively the FreeWay's physical address. You don't need download a physical address to the FreeWay via ETS as you normally do with other EIB devices. You also don't need to download an application to FreeWay using ETS as you'll program the application using FreeWay's script.

FreeScript EIB Functions

The following FreeScript functions are provided for EIB communications:

`EibPhysical()`	Call this in `hubinit()` to set the FreeWay's Physical address
`EibRegister()`	Call this in `hubinit()` to specify the groups you want to communicate with
`EibReceive()`	System Event function called when a registered group message is received
`EibSendFloat()`	Use this to send a float value to the bus
`EibSendString()`	Use this to send a string to the bus

EibPhysical()

You must call this function, usually in HubInit, to set the FreeWay's physical address. This should be the same as the physical address of the dummy device that you added to your ETS project.

```
       EIBPhysical(string sEibPhysical)
```

where

`sEibPysical`	- a string to set the FreeWay's physical address e.g. "0.1.2"

Example

```
       HubInit()
       {
         // FreeWay's physical address
         EIBPhysical("0.1.2");
       }
```

EibRegister()

You must call this function, usually in HubInit, for every group that you want FreeWay to communicate with This should be the same as the group addresses in your ETS project that are associated with the dummy device's objects. The function requires a unique handle which is used by send and receive functions to refer to the group; a group address as a string; a Data Type. You can register up to 64 groups.

```
EIBRegister(float fEibRegHandle, string sEibRegGroup, float fEibRegType)
```

where

`fEibRegHandle`	a unique handle to identify the EIB group - between 1 and 64
`sEibRegGroup`	the EIB group address that you want to register e.g. "0/0/1"
`fEibType`	the type of Datapoint that the Group uses. Options are

fEibType	Data Type	Format	Example
1	Boolean	1-bit	Switches & status
2	2-bit	2-bit	
3	3-bit Controlled	4-bit	Dimmers
4	Character Set	8-bit	Displays
5	8-Bit Unsigned Value	8-bit	e.g. dimmer brightness value
6	8-bit Signed Value	8-bit	Values
7	2-octet Unsigned Value	2-octet	e.g. thermostat temperature
8	2-octet Signed Value	2-octet	Values
9	2-octet Floating Point Value	2-octet	Values
10	Time	3-octet	Set bus time
11	Date	3-octet	Set bus date
12	4-octet Unsigned Value	4-octet	Values
13	4-octet Signed Value	4-octet	Values
14	4-octet Float Value	4-octet	Values
15	Access	4-octet	Access control
16	Character String	14 bytes	Text displays

Note: Registering a group address as Type 0 signifies an un-registered group.

Example

```
HubInit()
{
  // FreeWay’s physical address
  EIBPhysical(“0.1.2”);

  // Register a switch group address
  // -------------------------------
  // Group Handle = 1
  // Group Address = 0/1/2
  // Data Type = 1 = Boolean
  EIBRegister(1,”0/1/2”,1);

  // Register a dimmer control group address
  // -------------------------------
  // Group Handle = 2
  // Group Address = 0/1/3
  // Data Type = 3 = 3-bit control
  EIBRegister(2,”0/1/3,3);

}
```

EibReceive()

This is a System Event function that is automatically called when FreeWay receives a message from a group that you've registered. The function is passed 3 parameters: the group handle, the message value as a float, and the message value also as a string. You can use either the float or the string values for most data types with a few exceptions.

```
EIBReceive(float fEibRxHandle, float fEibRxValue, string sEibRxValue)
```

Where

`fEibHandle`	the handle of the registered group
`fEibRxValue`	the floating point value of the data received from the group
`sEibRxValue`	a string version of the data received from the group

Example

```
EIBReceive(float fEibRxHandle, float fEibRxValue, string sEibRxValue)
{
  // check the group's handle
  if (fEibRxHandle == 1)
  {
    // Read the value as a string....
    if (sEibRxValue[0] == 0) {DebugPrint("Value = On");}
    if (sEibRxValue[0] == 1) {DebugPrint("Value = Off");}

    // ... or Read the value as a float
    if (fEibRxValue == 0) {DebugPrint("Value = On");}
    if (fEibRxValue == 1) {DebugPrint("Value = Off");}
  }
}
```

EibSendFloat()

Use this function to send a value to group that you've registered. The function is passed 2 parameters: the group handle, the message value as a float. You'd use this function for sending 1, 4 and 8-bit and floating point values.

```
EIBSendFloat(float fEibTxfHandle, float fEibTxfValue)
```

where

`fEibTxfHandle`	the group handle previously registered with EIBRegister()
`fEibTxfValue`	the floating point value of the data to send to the group

Example

```
HubInit()
{
  EIBPhysical("0.1.2");

  // Group Handle = 1
  // Group Address = 0/1/2
  // Data Type = 1 = Boolean
  EIBRegister(1,"0/1/2",1);

  // send a value of 1 to the group
  EIBSendFloat(1, 1)
}
```

EibSendString()

Use this function to send a value to group that you've registered. The function is passed 2 parameters: the group handle, the message value as a string. You can use this function to send values for any of the data types, particularly those with multiple bytes.

```
EIBSendString(float fEibTxsHandle, string sEibTxsValue)
```

where

`fEibTxsHandle`	the group handle previously registered with EIBRegister()
`sEibTxsValue`	the string value of the data to send to the group

Example

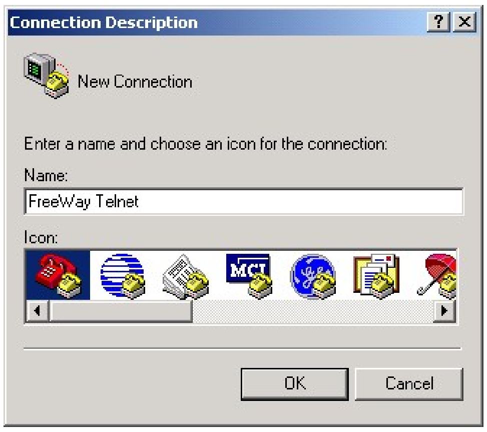
```
string sEibString;

HubInit()
{
  EIBPhysical("0.1.2");

  // Group Handle = 1
  // Group Address = 0/1/2
  // Data Type = 1 = Boolean
  EIBRegister(1,"0/1/2",1);

  // send a value of 1 to the group
  sEibString[0] = 1;
  EIBSendString(1, sEibString)
}
```

Appendix

A1 - Setting up a Telnet HyperTerminal Session

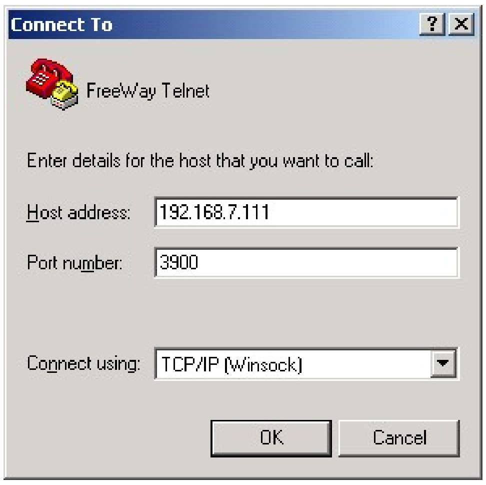
This section gives instructions on setting up a Telnet session file for HyperTerminal supplied with most versions of Windows.

- From the **Start** menu select **Programs->Accessories->Communications->HyperTerminal**.
- In the **New Connection** dialog box type in: **FreeWay Telnet** and click OK



Enter a name and choose an icon for the connection:
Name: FreeWay Telnet

- In the **Connect To** window, in the **Connect Using** drop-down box select **TCP/IP (Winsock)**
- For **Host address** type in **192.168.1.111** (or the FreeWay's new IP address if you have changed it)
- The **Port number** should be **3900.** Click on **OK**



Enter details for the host that you want to call:
Host address: 192.168.7.111
Port number: 3900
Connect using: TCP/IP (Winsock)

- Select **File->Save** from the HyperTerminal menu to save the session file. You can access the session file again from **Programs->Accessories->Communications->HyperTerminal->FreeWay Telnet.ht**

A2 - Downloading Scripts using FTP

FreeWay has an FTP interface as an alternative to Telnet for downloading scripts to FreeWay. There are numerous FTP Client programmes freely available on the Internet and Windows has one built-in. To use it:

- First run a Command Prompt session: **Select Start->Programs->Accessories->Command Prompt**
- You should then navigate to the directory that contains your script file. For example at the **C:\>** prompt type in:

  ```
  cd My Documents\FreeWay Scripts\script.txt
  ```
- Note that for Telnet transfers you can call the script anything you like. For FTP transfers the script file name must be **script.txt**.
- Then run the FTP Client by typing in `ftp` followed by the FreeWay's IP address, for example:

  ```
  ftp 192.168.1.111
  ```
- You will be prompted for a username, type in `dkt`
- You will be prompted for a password, type in `dkt`
- You are now logged in as an FTP client to FreeWay. To send your script file to FreeWay type in:

  ```
  send script.txt
  ```
- Type `quit` to exit the FTP client programme.
- You will now need to compile the script file using the **Compile** button on the FreeWay's built-in web page

A3 – EIB/KNX Datapoint Types

Type 1: Boolean

Format	1-bit
Range	0,1

This data type is used for switch states (e.g. 0/1, on/off, open/closed, up/down etc). On transmit it is used to switch devices on/off. On receive it provides device states.

On receive the bit value is provided in fEibRxValue and in sEibRxValue[0] as values 0 or 1.

On transmit the bit value can be sent using EibSendFloat() or EibSendString().

Example

This example receives a switch state from group address 0/1/1, prints out a debug message and sends the received value to group address 0/1/2.

```
HubInit()
{
  EIBPhysical(“0.0.1”);          // FreeWay physical address

  EIBRegister(1,”0/1/1”,1);      // Group Handle = 1
                                 // Group Address = 0/1/1
                                 // Data Type = 1 = Boolean
  EIBRegister(2,”0/1/2”,1);      // Group Handle = 2
                                 // Group Address = 0/1/2
                                 // Data Type = 1 = Boolean
}

EIBReceive(float fEibRxHandle, float fEibRxValue, string sEibRxValue)
{
float fValue;
string sValue;

  // check the group’s handle
  if (fEibRxHandle == 1)
  {
    // get the float value or ...
    fValue = fEibRxValue;
    // get the string value
    sValue = sEibRxValue[0];

    // send float to group 2 or ...
    EIBSendFloat(2, fValue);
    // send string to group 2
    EIBSendString(2, sValue);

  }
}
```

Type 2: 2-bit

Format	2-bit [C V]
Range	C = 0,1 V = 0,1
On receive	sString[0] = V
	sString[1] = C
	fFloat = CV

On transmit the bit value can be sent using EibSendFloat() or EibSendString().

Type 3 3-Bit Controlled

Format 4-bit [C VVV]

Values C = 0,1

0 = dim down

1 = dim up

V = 000…111

001-111 = dimming step

000 = stop

This data type is commonly used to increase or decrease the set value in steps, or to stop/start movement (e.g. to control a dimmer). Data is formatted as follows:

fFloat = [_ _ _ _ C V V V]

or

sString[0] = [_ _ _ _ _ _ _ C]

sString[1] = [_ _ _ _ _ V V V]

Example 1

This example prints out the status of received dimming messages.

```
HubInit()
{
  EIBPhysical(“0.0.1”);           // FreeWay physical address

  EIBRegister(1,”0/0/1”,3);       // Group Handle = 1
                                  // Group Address = 0/1/2
                                  // Data Type = 3 = 4-bit controlled
}


EIBReceive(float fEibRxHandle, float fEibRxValue, string sEibRxValue)
{
string sControl;
string sStep;
string sDebug;

  // check the group’s handle
  if (fEibRxHandle == 1)
  {
     // get the control value
     sControl = sEibRxValue[0];
     // get the step value
     sStep = sEibRxValue[1];

     if (sStep == 0)
     {
       DebugPrint(“Dimming stopped \n\r”);
     }
     else
     {
       if (sControl == 0)
       {
         sDebug = “Dimming Up with step = “ + s + “\n\r”;
         DebugPrint(sDebug);
       }
       else
       {
         sDebug = “Dimming Down with step = “ + s + “\n\r”;
         DebugPrint(sDebug);
       }
     }
  }
}
```

Example 2

This example shows a function which can be called by web browser buttons to control a dimmer.

```
ControlDimmer(float fCommand, float fStep)
{
string sEibString;

  // 1 = stop dimming
  if (fCommand == 1)
  {
    sEibString[0] = 0;
    sEibString[1] = 0;
    EibSendString(1,sEibString);
  }

  // 1 = dim up
  if (fCommand == 2)
  {
    sEibString[0] = 0;
    sEibString[1] = fStep;
    EibSendString(1,sEibString);
  }

  // 1 = dim down
  if (fCommand == 3)
  {
    sEibString[0] = 1;
    sEibString[1] = fStep;
    EibSendString(1,sEibString);
  }
}
```

Type 4 Character Set

Format 8-bit [AAAAAAAA]

Range 00h to FFh

This data type is used to transfer an ASCII character over the bus. Data is formatted as follows:

fFloat = [A A A A A A A A]

or

sString[0] = [A A A A A A A A]

Example 1

```
HubInit()
{
  EIBPhysical("0.0.1");           // FreeWay physical address

  EIBRegister(1,"0/0/1",4);       // Group Handle = 1
                                  // Group Address = 0/0/1
                                  // Data Type = 4 = character set
}

EIBReceive(float fEibRxHandle, float fEibRxValue, string sEibRxValue)
{
string sCharacter;
string sMessage;

  // check the group's handle
  if (fEibRxHandle == 1)
  {
    sMessage = "Character is " + sEibRxValue[0] + "\n\r";
    DebugPrint(sMessage);
  }
}
```

Type 5 8-Bit Unsigned Value

Format 8-bit [UUUUUUUU]

Range U = 0…255

fFloat = [UUUUUUUU]

sString[0] = [UUUUUUUU]

This data type is commonly used to transfer 8-bit unsigned integer values (e.g. counter values). Normally the range is 0 to 255 but the interpretation can differ depending on the sub-type. If sub-type scaling is selected the value is interpreted as a percent value from 0 to 100%. This can be used to set the brightness value of dimming actuators. If the sub-type Wind Direction is selected the value is interpreted as an angle in the range from 0^{O} to 360^{O}.

Example

```
HubInit()
{
  EIBPhysical("0.0.1");          // FreeWay physical address

  EIBRegister(1,"0/0/1",5);      // Group Handle = 1
                                 // Group Address = 0/1/2
                                 // Data Type = 5 = 8-bit Unsigned Value
}

EIBReceive(float fEibRxHandle, float fEibRxValue, string sEibRxValue)
{
string sMessage;
float fValue;

  // check the group's handle
  if (fEibRxHandle == 1)
  {
    // interpret as a raw value from 0-255
    sMessage = "Raw value is " + fEibRxValue + "\n\r";
    DebugPrint(sMessage);

    // interpret as a scaling value from 0-100%
    fValue = 100*fEibRxValue/255;
    sMessage = "Scaling value is " + fValue + "%\n\r";
    DebugPrint(sMessage);

    // interpret as an angle value from 0-360⁰
    fValue = 360*fEibRxValue/255;
    sMessage = "Angle value is " + fValue + "degrees\n\r";
    DebugPrint(sMessage);
  }
}
```

Type 6 8-Bit Signed Value

Format	8-bit	[VVVVVVVV]
Range	V = -128…+127	

fFloat = [VVVVVVVV]

sString[0] = [VVVVVVVV]

This data type is commonly used to transfer 8-bit signed integer values in the range is -127 to +128.

Type 7 2-octet Unsigned Value

Format	2-octet:	MSB [UUUUUUUU]	LSB [UUUUUUUU]
Range	U = 0…65535		

fFloat = value

sString[0] = [UUUUUUUU]	LSB
sString[1] = [UUUUUUUU]	MSB

This data type is commonly used to transfer 2-byte unsigned integer values (e.g. counter values).

Type 8 2-octet Signed Value

Format	2-octet:	MSB [VVVVVVVV]	LSB [VVVVVVVV]
Range	V = -32768…+32767		

fFloat = value

sString[0] = [VVVVVVVV] LSB

sString[1] = [VVVVVVVV] MSB

This data type is commonly used to transfer 2-byte signed integer values (e.g. counter values).

Type 9 2-octet Floating Point Value

Format	2-octet:	MSB [M EEEE MMM]	LSB [MMMMMMMM]
Range	-671088.64…+670760.96		

fFloat = value

sString[0] = [VVVVVVVV] LSB

sString[1] = [VVVVVVVV] MSB

This data type is commonly used to transfer 2-byte analogue values (e.g. values from a temperature sensor).

Type 10 Time

Format	3-octet	Byte 3 [ddd hhhhh]
		Byte 2 [00 mmmmmm]
		Byte 1 [00 ssssss]

fFloat = not used

sString[0] = day, 0 to 6, 0 = monday

sString[1] = hour

sString[1] = minutes

sString[1] = seconds

This data type is used to transfer 3-byte time values. The time value is read from the bus and stored in 4 characters of the sEibRxValue string as follows:

```
Day in        sEibRxValue[0] =    "0..6" where 0 is Monday, etc
Hours in      sEibRxValue[1] =    "0…23"
Minutes in    sEibRxValue[2] =    "0…59"
Seconds in    sEibRxValue[3] =    "0…59"
```

Example

This example reads the time from the bus and prints out of the FreeWays' debug output.

```
HubInit()
{
  EIBPhysical("0.0.1");          // FreeWay physical address

  EIBRegister(1,"0/0/1",12);     // Group Handle = 1
                                 // Group Address = 0/1/2
                                 // Data Type = 12 = 3-octet Time
}

EIBReceive(float fEibRxHandle, float fEibRxValue, string sEibRxValue)
{
string sMessage;

  // check the group's handle
  if (fEibRxHandle == 1)
  {
    sMessage = "EIB Time is " + sEibRxValue[0] + ", " +
                                 sEibRxValue[1] + ":" +
                                 sEibRxValue[2] + ":" +
                                 sEibRxValue[3] + "\n\r";
    DebugPrint(sMessage);
  }
}
```

Type 11 Date

Format	3-octet	Byte 3 [000 DDDDD]
		Byte 2 [0000 MMMM]
		Byte 1 [0 YYYYYYY]

This data type is used to transfer 3-byte date values. The date value is read from the bus and stored in 3 characters of the sEibRxValue string as follows:

```
      Day in       sEibRxValue[0] =    "1…31"
      Month in     sEibRxValue[1] =    "1…12"
      Year in      sEibRxValue[2] =    "0…99"
```

Example

This example reads the date from the bus and prints out of the FreeWay's debug output.

```
HubInit()
{
  EIBPhysical("0.0.1");          // FreeWay physical address

  EIBRegister(1,"0/0/1",13);     // Group Handle = 1
                                 // Group Address = 0/1/2
                                 // Data Type = 13 = 3-octet Date
}

EIBReceive(float fEibRxHandle, float fEibRxValue, string sEibRxValue)
{
string sMessage;

  // check the group's handle
  if (fEibRxHandle == 1)
  {
    sMessage = "EIB Date is " + sEibRxValue[0] + "/" +
                                 sEibRxValue[1] + "/20" +
                                 sEibRxValue[2] + "\n\r";
    DebugPrint(sMessage);
  }
}
```

Type 12 4-octet Unsigned Value

Format	4-octet:	MSB [UUUUUUUU] [UUUUUUUU] [UUUUUUUU] [UUUUUUUU] LSB
Range	U = 0…4,294,967,295	

fFloat = value

sString[0] = [UUUUUUUU] LSB

sString[1] = [UUUUUUUU]

sString[2] = [UUUUUUUU]

sString[3] = [UUUUUUUU] MSB

This data type is commonly used to transfer 4-byte unsigned integer values (e.g. counter values).

Type 13 4-octet Signed Value

Format	4-octet:	MSB [UUUUUUUU] [UUUUUUUU] [UUUUUUUU] [UUUUUUUU] LSB
Range	V = -2,147,483,648…+2,147,483,647	

fFloat = value

sString[0] = [UUUUUUUU] LSB

sString[1] = [UUUUUUUU]

sString[2] = [UUUUUUUU]

sString[3] = [UUUUUUUU] MSB

This data type is commonly used to transfer 4-byte signed integer values.

Type 14 4-octet Floating Point Value

Format	4-octet:	MSB [M EEEE MMM]	LSB [MMMMMMMM]
Range			

fFloat = value

sString[0] = [UUUUUUUU] LSB

sString[1] = [UUUUUUUU]

sString[2] = [UUUUUUUU]

sString[3] = [UUUUUUUU] MSB

This data type is commonly used to transfer 4-byte analogue values (e.g. values from a temperature sensor).

Type 15 Access

Type 16 Character String

Format	14-octet:
Range	n/a

fFloat = not used

sString[0] = first character

….

sString[13] = last character

This type is used for sending text messages for textual displays.