4D LABS SEMICONDUCTORS

Workshop4

Integrated Development Environment

ESP32 4DGL Functions Manual

Document Revision: 1.1

Document Date: 4th March 2025

ESP32 4DGL Functions Manual

Table of Contents

1. Introduction

4D Systems' 4Discovery product line is powered by a Diablo16 microcontroller. Some modules included in this product line is equipped with Espressif System's ESP32 that provides Wi-Fi and Bluetooth functionality to the display module. The 4Discovery variants relevant to this document is highlighted in the table below.

	Wi-Fi	Bluetooth	Flash	microSD
4Discovery-50	✗	✗	✗	✓
4Discovery-50W	✓	✓	✗	✓

4D Systems developed a custom firmware for the ESP32 featuring the most used Wi-Fi and Bluetooth operations. The custom firmware comes preloaded to all 4Discovery variant equipped with ESP32 chip. If users require to reprogram the ESP32 chip of the 4Discovery, the custom firmware is available for download in the display's product page. For instructions on how to update or reprogram the ESP32 chip, please refer to the 4Discovery datasheet.

This custom firmware can be used together conveniently with 4DGL library ***ESP32_4DGL_SD.inc*** available with Workshop4 installation. This document will cover detail discussion for each available function in the library.

2. General Functions

This section contains functions that are generally used to setup communication with ESP32, query version information and perform basic update functionalities.

Below is a list of categories discussed in this section:

- Library Initialization
- Support Functions
- Update Functions

2.1. Library Initialization

This section discusses about the functions used to initialize the UART communication with ESP32.

Below is a list of functions discussed in this section:

- ESP_Initialize
- ESP_RX_pin
- ESP_TX_pin
- ESP_RST_pin
- ESP_Reset

2.1.1. ESP_Initialize(mode)

<table>
<tr><td>Syntax</td><td colspan="2">ESP_Initialize(mode)</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td rowspan="2">Arguments</td><td colspan="2">mode</td></tr>
<tr><td>mode</td><td>Specifies the display configuration to use</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td>Returns</td><td colspan="2">None</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td>Description</td><td colspan="2">Use this function to initialize the UART communication between the graphics processor and WiFi/Bluetooth chip (ESP32)

<table>
<tr><td rowspan="2">Mode</td><td rowspan="2">Variant</td><td colspan="3">Pins</td></tr>
<tr><td>TX</td><td>RX</td><td>RST</td></tr>
<tr><td>ESP_4DISCOVERY50</td><td>4Discovery-5.0</td><td>PA6</td><td>PA5</td><td>PA15</td></tr>
</table>

This function will also reset the ESP chip.</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td>Example</td><td colspan="2"><code>ESP_Initialize(ESP_4DISCOVERY50); // Initialize Comms for 5" 4Discovery</code></td></tr>
</table>

2.1.2. ESP_Reset()

Syntax	ESP_Reset()
Arguments	none
Returns	none
Description	Reset the ESP32 chip
Example	`ESP_Reset();`

2.2. Support Functions

This section discusses about the functions supporting functions that provides users to easily check the firmware and library versions. This also includes functions for checking general errors.

Below is a list of functions discussed in this section:

- ESP_GetVersion
- ESP_PrintVersion
- ESP_CheckVersion
- ESP_GetError

2.2.1. ESP_GetVersion()

Syntax	ESP_GetVersion()	
Arguments	none	
Returns	version	
	version	Returns the byte-aligned pointer to the string containing the firmware version
Description	Checks the number of bytes currently stored in the UART buffer initialized for used with the ESP32 module	

Example	`var version;`
	`version := ESP_GetVersion();`
	`str_Printf(&version, "ESP Version: %s\n");`	

2.2.2. ESP_PrintVersion(outstream)

<table>
<tr><td>Syntax</td><td colspan="2">ESP_PrintVersion(outstream)</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td rowspan="2">Arguments</td><td colspan="2">outstream</td></tr>
<tr><td>outstream</td><td>Specifies the destination for printing the ESP32 firmware version</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td>Returns</td><td colspan="2">none</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td>Description</td><td colspan="2">Prints the firmware version to the specified outstream

<table>
<tr><th>Keyword</th><th>Value</th><th>Description</th></tr>
<tr><td>DSK</td><td>0xF802</td><td>Output is directed to the most recently open file that has been opened in write mode</td></tr>
<tr><td>COM0</td><td>0x01</td><td>Output is redirected to the COM0 (default serial) port</td></tr>
<tr><td>COM1</td><td>0x00</td><td>Output is redirected to the COM1 port</td></tr>
<tr><td>COM2</td><td>0x01</td><td>Output is redirected to the COM2 port</td></tr>
<tr><td>COM3</td><td>0x00</td><td>Output is redirected to the COM3 port</td></tr>
<tr><td>I2C1</td><td>0x01</td><td>Output is redirected to the I2C1 port</td></tr>
<tr><td>I2C2</td><td>0x00</td><td>Output is redirected to the I2C2 port</td></tr>
<tr><td>I2C3</td><td>0x01</td><td>Output is redirected to the I2C3 port</td></tr>
<tr><td>MDA</td><td>0x00</td><td>Output is directed to the SD/SDHC or FLASH media.
Warning – be careful writing to a FAT16 formatted card without checking legal partitioned are else the disk formatting will be destroyed</td></tr>
</table>
</td></tr>
<tr><td colspan="3"></td></tr>
<tr><td>Example</td><td colspan="2">ESP_PrintVersion(COM0); // Print version to COM0</td></tr>
</table>

2.2.3. ESP_CheckVersion()

Syntax	ESP_CheckVersion()	
Arguments	none	
Returns	result	
	result	Returns true if the ESP firmware version matches the version of the library. Otherwise, returns false
Description	Compares the ESP32 firmware version with the library version. If both version match, this function returns true (1), otherwise, it returns false (0).	

Example	`if (ESP_CheckVersion())`
	`  print("ESP firmware version matches the library");`
	`endif`	

2.2.4. ESP_GetError()

Syntax	ESP_GetError()	
Arguments	none	
Returns	error	
	error	Last error that occurred in the ESP processes
Description	Query the ESP chip for the last error that occurred. Possible results and their meaning are discussed below:	

Error	Description
COMMAND_SUCCESS	Last command was processed successfully
COMMAND_NOT_FOUND	Last command is not supported or invalid
WIFI_MODE_INVALID	Mode used for **WiFi_SetMode** is invalid
WIFI_BEGIN_FAILED	Last use of **WiFi_Begin** did not complete successfully
HTTP_CERT_TOO_LONG	The web certificate set by **HTTP_SetSecure** exceed the current limit of **4096** characters
HTTP_WIFI_NOT_CONNECTED	Last use of **HTTP_StartRequest** failed since Wi-Fi is not connected
HTTP_REQUEST_ERROR	An error occurred while sending GET/POST request using **HTTP_StartRequest.** Check the HTTP error code using **HTTP_GetError** for more details
HTTP_INVALID_SIZE	The response of the server to the GET/POST request sent using **HTTP_StartRequest** is zero (0) byte
HTTP_INCORRECT_CHECKSUM	The graphics processor sent an invalid checksum while receiving the response to the GET/POST request sent using **HTTP_StartRequest**
HTTP_SIZE_MISMATCH	The number of bytes received did not match the expected response size to the GET/POST request sent using **HTTP_StartRequest**

Example	`var err;` `err := ESP_GetError();`

2.3. Update Functions

This section discusses about existing update functions for ESP32 and graphics processor.

Below is a list of functions discussed in this section:

- ESP_OTA_Update
- ESP_AttachWebUpdateHandler
- ESP_ToggleWebUpdater
- ESP_CheckWebUpdater
- ESP_HandleWebUpdater

2.3.1. ESP_FirmwareUpdate(handler, timeout)

Syntax	ESP_FirmwareUpdate(handler, timeout)	
Arguments	handler, timeout	
	handler	User function to handle event and progress reports. Setting this to 0 disables update handler.
	timeout	Wait time in milliseconds for ESP32 firmware update. If set to -1, this function will wait indefinitely
Returns	result	
	result	Returns true if the update was successful, otherwise, returns false
Description	This function enables ESP32 Over-the-Air update. If timeout is set to -1, the display will wait indefinitely until an update is received by the ESP32. Otherwise, this function will only wait until the specified timeout is reached. User needs to specify a custom function to handle the status messages from ESP32. The handler function should follow the format: ***func Handler (var event, var progress)*** Each Over-the-Air update *event* is described below:	

Event	Description
OTA_UPDATE_STARTED	ESP32 received an update request
OTA_UPDATE_PROGRESS	ESP32 sent a progress report in percentage
OTA_UPDATE_ENDED	Firmware update ended successfully
OTA_UPDATE_ERROR	An error occurred while attempting an update

The update *progress* is provided as a percentage value.

The function will disable OTA update as soon as a result is available which is triggered by OTA_UPDATE_ENDED or OTA_UPDATE_ERROR events.

Example

```
#platform "uLCD-50DCT"

#inherit "4DGL_16bitColours.fnc"

#inherit "ESP32_4DGL_SD.inc"

#DATA
    byte SSID       "ssid", 0
    byte PASSWORD   "password", 0
#END
```

```
func main()

    gfx_ScreenMode(PORTRAIT) ; // change manually if orientation change
    putstr("Prepare ESP to receive OTA update (from Arduino)\n");

    ESP_Initialize(ESP_4DISCOVERY50);
    WiFi_SetMode(WIFI_STATION);

    print("Connecting to ", [STR]SSID);
    WiFi_Begin(SSID, PASSWORD);

    while(WiFi_Status() != WIFI_CONNECTED)
        pause(500);
        putch('.');
    wend
    putch('\n');
    WiFi_PrintLocalIP(0); // Print local IP to current cursor position

    putstr("\nWaiting for update...\n");

    var res; // Wait up to ~30 seconds for the update
    res := ESP_FirmwareUpdate(handleEspUpdate, 30000);

    if (res)
        print("Resetting 4Discovery in 1000ms\n");
        pause(1000);
        SystemReset();
    else
        print("Failed to update ESP firmware\n");
    endif

    repeat                      // maybe replace
    forever                     // this as well

endfunc

// Define a Handler Function
func handleEspUpdate(var cmd, var progress)

    var y;
    switch (cmd)

        case OTA_UPDATE_TIMEOUT:
            print("ESP update didn't start on time\n");
            break;

        case OTA_UPDATE_STARTED:
            print("ESP Update Started\n");
            break;
```

```
        case OTA_UPDATE_PROGRESS:
            y := peekW(TEXT_YPOS);
            gfx_MoveTo(0, y);
            print("Update at ", progress, " percent  ");
            break;

        case OTA_UPDATE_ENDED:
            print("\nESP Updated Successfully\n");
            break;

        case OTA_UPDATE_ERROR:
        default:
            print("\nError Occurred while updating ESP\n");
            break;

    endswitch

endfunc
```

2.3.2. ESP_AttachWebUpdateHandler(handler)

Syntax	ESP_AttachWebUpdateHandler(handler)	
Arguments	handler	
	handler	User function to handle progress reports for the web updater. Setting this to 0 disables the update handler.
Returns	none	
Description	Attach a progress report handler for file uploads. The handler function should use the format: ***func updateHandler (var state, var index, var * progress)*** • *state* – the current status of the update • *index* – the index of the file being transferred (starts with 0) • *progress* – the number of bytes received for the current file The ***state*** can have the following values:	
Example	See **`ESP_HandleWebUpdater`**	

State	Description
WEB_UPDATE_FILE	Indicates a new file being uploaded
WEB_CONTINUE_FILE	Indicates the current file continuing upload
WEB_END_FILE	Indicates the current file finished uploading
WEB_END_UPDATE	Indicates all files have been uploaded
WEB_UPDATE_ERROR	Indicates an error during an ongoing file upload

The ***progress*** is a 16-bit pointer to an unsigned 32-bit value.

2.3.3. ESP_ToggleWebUpdater()

Syntax	ESP_ToggleWebUpdater()	
Arguments	none	
Returns	result	
	result	Returns true if the web updater was successfully enabled/disabled. Otherwise, returns false
Description	Toggles the state of the web server which provides a simple web page that can be used to upload files to the display's primary media storage device. **Note:** (1) The webpage is accessible using any browser by acquiring the IP address using **WiFi_LocalIP or WiFi_PrintLocalIP** (2) Ensure that all files follow 8.3 FAT Filename format	
Example	See **`ESP_HandleWebUpdater`**	

2.3.4. ESP_CheckWebUpdater()

Syntax	ESP_CheckWebUpdater()	
Arguments	none	
Returns	result	
	result	Returns true if a file upload has been started. Otherwise, returns false
Description	This function checks if the ESP32 is receiving file uploads through the webserver started by **ESP_ToggleWebUpdater** **Note:** (1) The webpage is accessible using any browser by acquiring the IP address using **WiFi_LocalIP** or **WiFi_PrintLocalIP** (2) Ensure that all files follow 8.3 FAT Filename format	
Example	See **`ESP_HandleWebUpdater`**	

2.3.5. ESP_HandleWebUpdater()

Syntax	ESP_HandleWebUpdater()
Arguments	none
Returns	none
Description	A blocking function that will handle the receiving of file uploads. The file upload progress can be monitored by attaching a simple handler function using **ESP_AttachWebUpdateHandler** or through the progress bar displayed in the basic file upload webpage.
Example	

```
#platform "uLCD-50DCT"

// Program Skeleton 1.4 generated 8/24/2020 1:36:04 AM

#inherit "4DGL_16bitColours.fnc"

#inherit "VisualConst.inc"

// var gradientRAM[29+xxx*2] := [-1,-1,-9999,0,0,xxx];
// uncomment and replace xxx with maximum of all inherent 'media' widgets

#inherit "InitialGfxFirmwareConst.inc"

#inherit "ESP32_4DGL_SD.inc"

#DATA
    byte SSID       "ssid", 0
    byte PASSWORD   "password", 0
#END

func main()

    gfx_Set(SCREEN_MODE, PORTRAIT);

    while(!(file_Mount()))
        putstr("Drive not mounted...");
        pause(200);
        gfx_Cls();
        pause(200);
    wend
    hndl := file_LoadImageControl("INITIA~1.dat", "INITIA~1.gci", 1);

    ESP_Initialize(ESP_4DISCOVERY50);

    WiFi_SetMode(WIFI_STATION);
```

```
print("Connecting to ", [STR]SSID);
WiFi_Begin(SSID, PASSWORD);
while (WiFi_Status() != WIFI_CONNECTED)
    pause(500);
    putch('.');
wend
putch('\n');
gfx_Cls();

var frame := 0;
// Angularmeter1 1.0 generated 8/24/2020 2:17:53 AM
img_SetWord(hndl, iAngularmeter1, IMAGE_INDEX, frame);
img_Show(hndl, iAngularmeter1);

// Winbutton1 1.0 generated 8/24/2020 1:41:02 AM
img_ClearAttributes(hndl, iWinbutton1, I_TOUCH_DISABLE);
img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, 0);
img_Show(hndl, iWinbutton1);

var touchState, touchImg, pressedImg;
touch_Set(TOUCH_ENABLE);

var gaugeTimer;
    gaugeTimer := sys_T();

repeat
    if (sys_T() - gaugeTimer > 500) // update gauge every half second
        frame++;
        if (frame > 10) frame := 0;
        img_SetWord(hndl, iAngularmeter1, IMAGE_INDEX, frame);
        img_Show(hndl, iAngularmeter1);
        gaugeTimer := sys_T();
    endif

    touchState := touch_Get(TOUCH_STATUS);
    touchImg := img_Touched(hndl, -1);

    switch (touchState)
        case TOUCH_PRESSED:
            pressedImg := touchImg;
            if (touchImg == iWinbutton1)
                img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, 1);
                img_Show(hndl, iWinbutton1);
            endif
            break;
        case TOUCH_RELEASED:
            if ((touchImg == pressedImg) && (touchImg == iWinbutton1))
                img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, 0);
                img_Show(hndl, iWinbutton1);
```

```
                    goto PerformUpdate;
                endif
                break;
            default:
                break;
        endswitch
    forever

PerformUpdate:

    gfx_Cls();
    putstr("Waiting for file uploads @");
    WiFi_PrintLocalIP(0); // Print local IP to current cursor position
    putch('\n');

    ESP_AttachWebUpdateHandler(handleGfxUpdate);
    if (!ESP_ToggleWebUpdater()) // Starts the file upload server
        print("Can't start web updater\n");
        repeat forever
    endif
    print("Waiting for update...\n");

    var isUpdating := 0;
    var updateTimer;
        updateTimer := sys_T();

    repeat
        // Required to run as frequent as possible
        isUpdating := ESP_CheckWebUpdater();
        // Ideally, there is no other ESP32 code
        // while running this
    until(isUpdating || (sys_T() - updateTimer >= 30000));
    // Wait up to 30 seconds

    if (!isUpdating) // If no update has is initiated
        ESP_ToggleWebUpdater(); // Stop the file upload server
        print("No update was received\n");
    else // Otherwise, immediately handle the update
        ESP_HandleWebUpdater(); // This will block until update ends
        // ESP_ToggleWebUpdater(); // Stop the file upload server
        // Not required if the display is going to reset resulting
        // to reinitialization of ESP32

        // Reset the display when done
        print("Web update ended... Resetting in 5 seconds..\n");
        pause(5000);
        SystemReset();
    endif

    repeat forever
```

```
endfunc

// Define a Handler Function
func handleGfxUpdate(var state, var index, var * progress)
    var y, ptr;
    switch (state)
        case WEB_UPDATE_FILE:
            print("Receiving File No. ", index, "\n");
            break;
        case WEB_CONTINUE_FILE:
            y := peekW(TEXT_YPOS);
            ptr := str_Ptr(progress);
            gfx_MoveTo(0, y);
            str_Printf(&ptr, "Received %lu bytes     ");
            break;
        case WEB_END_FILE:
            print("\nReceived File No. ", index, " successfully\n");
            break;
        case WEB_END_UPDATE:
            print("\nFile(s) Received Successfully\n");
            break;
        case WEB_UPDATE_ERROR:
        default:
            print("\nError Occurred while receiving file(s)\n");
            break;
    endswitch
endfunc
```

3. Wi-Fi Functions

This section contains functions that are used for any ESP32's Wi-Fi functionality.

Below is a list of features related to Wi-Fi discussed in this section:

- Wi-Fi Setup
- Hypertext Transfer Protocol
- User Datagram Protocol
- Network Time Protocol
- Alexa Integration

3.1. Setup Functions

This section discusses the functions for initializing Wi-Fi functionality of ESP32.

Below is a list of functions discussed in this section:

- WiFi_ScanSSIDs
- WiFi_GetSSID
- WiFi_GetRSSI
- WiFi_ConnectedSSID
- WiFi_ConnectedRSSI
- WiFi_GetEncryptionType
- WiFi_SetMode
- WiFi_Begin
- WiFi_Status
- WiFi_LocalIP
- WiFi_PrintLocalIP
- WiFi_Disconnect

3.1.1. WiFi_SetMode(mode)

Syntax	WiFi_SetMode(mode)	
Arguments	mode	
	mode	Specifies the Wi-Fi mode to use: Access Point or Client Station
Returns	result	
	result	Specifies whether the function was executed successfully (1) or not (0)
Description	Sets the Wi-Fi mode to be used when initializing the Wi-Fi features using: **WiFi_Begin(ssid, pass)** The default mode for Wi-Fi operations is as a station (WIFI_STATION). **Keyword** / **Value** / **Description** WIFI_STATION / 0x00 / Sets Wi-Fi mode as client station WIFI_SOFT_AP / 0x01 / Sets Wi-Fi mode as software-enabled access point **Note:** Although ESP32 supports running at both modes at the same time, this may cause undesirable latency. Furthermore, most applications would not require running at both modes. Therefore, **the custom firmware only allows using one mode at a time**.	

Example	`// Example 1:`
	`// Sets mode as Wi-Fi Client/Station`
	`WiFi_SetMode(WIFI_STATION);`
	
	`// Example 2:`
	`// Sets mode as Wi-Fi Access Point`
	`WiFi_SetMode(WIFI_SOFT_AP);`	

3.1.2. WiFi_ScanSSIDs()

Syntax	WiFi_ScanSSIDs()	
Arguments	none	
Returns	count	
	count	Number of detected nearby access points
Description	Scans for nearby Wi-Fi networks while in **WIFI_STATION** mode and returns the number of access points found	

Example	`var count, i, enc, rssi, ssid[10]; // Max 19 characters SSID name`
	`count := WiFi_ScanSSIDs();`
	
	`for (i := 0; i < count; i++)`
	`  WiFi_GetSSID(i, ssid);`
	`  rssi := WiFi_GetRSSI(i);`
	`  enc := WiFi_GetEncryptionType(i);`
	`  print([STR]ssid, " [", enc, "] ", rssi, " dBm  \n");`
	`next`	

3.1.3. WiFi_GetSSID(index, buffer)

Syntax	WiFi_GetSSID(index, buffer)	
Arguments	index, buffer	
	index	Specifies the index number to query
	buffer	Specifies a word-aligned pointer to a buffer to store the SSID
Returns	ssid	
	ssid	Returns the byte-aligned pointer to the SSID string
Description	**WiFi_ScanSSIDs** must be used successfully before using this function. This function queries the ESP32 for the SSID name of the access point specified by *index* and detected by the last SSID scan. The SSID name is then stored to the *buffer* specified.	
Example	See **`WiFi_ScanSSIDs`**	

3.1.4. WiFi_GetRSSI(index)

Syntax	WiFi_GetRSSI(index)	
Arguments	index	
	index	Specifies the index number to query
Returns	rssi	
	rssi	Returns the RSSI / Received Signal Strength in dBm
Description	**WiFi_ScanSSIDs** must be used successfully before using this function. This function queries the ESP32 and returns the signal strength of the access point specified by *index* and detected by the last SSID scan.	
Example	See **`WiFi_ScanSSIDs`**	

3.1.5. WiFi_GetEncryptionType(index)

Syntax	WiFi_GetEncryptionType(index)	
Arguments	index	
	index	Specifies the index number to query
Returns	encryption	
	encryption	Returns the encryption type
Description	**WiFi_ScanSSIDs** must be used successfully before using this function. This function queries the ESP32 and returns the encryption type of the access point specified by *index* and detected by the last SSID scan. Possible encryption types are as shown below:	
Example	See **`WiFi_ScanSSIDs`**	

Encryption Type	Value
ENC_TYPE_TKIP	2
ENC_TYPE_CCMP	4
ENC_TYPE_WEP	5
ENC_TYPE_NONE	7
ENC_TYPE_AUTO	8

3.1.6. WiFi_Begin(ssid, pass)

Syntax	WiFi_Begin(ssid, pass)	
Arguments	ssid, pass	
	ssid	pointer to string containing the access point name
	pass	pointer to string containing the password
Returns	result	
	result	Specifies whether the function was executed successfully (1) or not (0)
Description	This function either connects to a Wi-Fi access point or create a new access point with the provided SSID and Password depending on the mode set using **WiFi_SetMode(mode)**. In **WIFI_STATION** mode, the actual status of connection can be checked using **WiFi_Status()**	

Example	`// Example 1:`
	`// Directly using strings as parameter`
	`WiFi_Begin("WIFI_NAME", "WIFI_PASS");`
	
	`// Example 2:`
	`// Storing the SSID and Password before starting Wi-Fi`
	`var ssid[10], pass[10];`
	`to(ssid); print("WIFI_NAME");`
	`to(pass); print("WIFI_PASS");`
	
	`WiFi_Begin(ssid, pass);`
	
	`// Example 3:`
	`// Creating an access point with SSID and Password as provided`
	`var ssid[10], pass[10];`
	`to(ssid); print("WIFI_NAME");`
	`to(pass); print("WIFI_PASS");`
	
	`WiFi_SetMode(WIFI_SOFT_AP);`
	`WiFi_Begin(ssid, pass);`	

3.1.7. WiFi_ConnectedSSID(buffer)

Syntax	WiFi_ConnectedSSID(buffer)	
Arguments	buffer	
	buffer	Specifies a word-aligned pointer to a buffer to store the SSID
Returns	ssid	
	ssid	Returns the byte-aligned pointer to the SSID string
Description	This function queries the ESP32 for the SSID name of the current access point it is connected to while in **WIFI_STATION** mode. The SSID name is then stored to the *buffer* specified.	

Example	`var rssi, ssid[10]; // max 19 characters for SSID`
	`WiFi_ConnectedSSID(ssid);`
	`rssi := WiFi_ConnectedRSSI();`
	`print([STR]ssid, " : ", rssi, " dBm\n");`	

3.1.8. WiFi_ConnectedRSSI()

Syntax	WiFi_GetRSSI(index)	
Arguments	none	
Returns	rssi	
	rssi	Returns the RSSI / Received Signal Strength in dBm
Description	This function queries the ESP32 and returns the signal strength of the access point it is currently connected to while in **WIFI_STATION** mode.	
Example	See **`WiFi_ConnectedSSID`**	

3.1.9. WiFi_Status()

Syntax	WiFi_Status()	
Arguments	none	
Returns	status	
	status	Specifies whether the chip is connected to an access point
Description	Checks if ESP32 is successfully connected to an access point set using **WiFi_Begin(ssid, pass)** while mode is **WIFI_STATION**: see **WiFi_SetMode(mode)** **Note:** As mentioned above, this function is only for **WIFI_STATION** mode	

Example	`// By default, Wi-Fi mode is WIFI_STATION`
	`// Therefore, there is no need to set it initially`
	`WiFi_Begin("WIFI_NAME", "WIFI_PASS");`
	
	`// Wait until Wi-Fi is connected`
	`// Note that this example will block continuously until it connects`
	`while (WiFi_Status() != WIFI_CONNECTED)`
	`  pause(500);`
	`  print(".");`
	`wend`	

3.1.10. WiFi_LocalIP()

Syntax	WiFi_LocalIP()	
Arguments	none	
Returns	IPaddress	
	IPaddress	Returns the byte-aligned pointer to the string containing the IP address
Description	Queries IP address and stores the string to **IPaddress** which is a word-aligned pointer to the stored IPaddress	

Example	`// Example 1: Stores the IP Address to IPaddress`
	`WiFi_LocalIP();`
	`print([STR]IPaddress); // Print IP address`
	
	`// Example 2:`
	`var ip;`
	`ip := WiFi_LocalIP();`
	`str_Printf(&ip, "Local IP: %s\n");`	

3.1.11. WiFi_PrintLocalIP(outstream)

Syntax	WiFi_PrintLocalIP(outstream)	
Arguments	outstream	
	outstream	Specifies the destination for printing the IP Address
Returns	none	
Description	Queries IP address and stores the string to **IPaddress** which is a word-aligned pointer to the stored IP address and immediately print it on the specified destination	
Example	`// Stores the IP Address to IPaddress and immediately print it to COM0` `WiFi_PrintLocalIP(COM0);`	

Keyword	Value	Description
DSK	0xF802	Output is directed to the most recently open file that has been opened in write mode
COM0	0x01	Output is redirected to the COM0 (default serial) port
COM1	0x00	Output is redirected to the COM1 port
COM2	0x01	Output is redirected to the COM2 port
COM3	0x00	Output is redirected to the COM3 port
I2C1	0x01	Output is redirected to the I2C1 port
I2C2	0x00	Output is redirected to the I2C2 port
I2C3	0x01	Output is redirected to the I2C3 port
MDA	0x00	Output is directed to the SD/SDHC or FLASH media. **Warning** – be careful writing to a FAT16 formatted card without checking legal partitioned are else the disk formatting will be destroyed

3.1.12. WiFi_Disconnect()

Syntax	WiFi_Disconnect(index)	
Arguments	none	
Returns	result	
	result	Returns result of the operation, usually ignored
Description	This function attempts to disconnect ESP32 from the current network it is connected to while in **WIFI_STATION** mode.	
Example	`WiFi_Disconnect(); // Disconnect from current network`	

3.2. Hypertext Transfer Protocol (HTTP) Functions

This section discusses about the functions used to utilize HTTP/HTTPS. This is commonly used when using ESP32 in **WIFI_STATION** mode to access the internet. However, it is also possible to use in while the ESP32 is in **WIFI_SOFT_AP** mode with a known local web server connected to it. See **WiFi_SetMode(mode)** for details.

Below is a list of functions discussed in this section:s

- HTTP_SetMode
- HTTP_SetHost
- HTTP_SetPath
- HTTP_AddData
- HTTP_SetOutputFile
- HTTP_StartRequest

3.2.1. HTTP_AttachProgressHandler(handler)

Syntax	HTTP_AttachProgressHandler(handler)	
Arguments	handler	
	handler	User function to handle progress reports for the HTTP/HTTPS requests. Setting this to 0 disables the update handler.
Returns	none	
Description	Attach a progress report handler for HTTP requests. The handler function should use the format: ***func updateHandler(var percent, var * progress, var * total)*** • *percent* – percentage progress of the request • *progress* – the number of bytes received for the current file • *total* – the total number of bytes expected Both ***progress*** and ***total*** are 16-bit pointers to unsigned 32-bit values.	
Example	See `HTTP_StartRequest`	

3.2.2. HTTP_SetOutputFile(filename)

Syntax	HTTP_SetOutputFile(filename)	
Arguments	filename	
	filename	word aligned pointer to string containing the filename
Returns	none	
Description	This function should be used when using an **SD card** as the main media storage device. This sets the filename for the output file that will be used to save the output when **HTTP_StartRequest** is executed.	

Example	`// Example 1:`
	`// Directly using string as parameter`
	`HTTP_SetOutputFile("output.txt");`
	
	`// Example 2:`
	`// Storing the filename before setting it`
	`var filename[10], value[10];`
	`to(filename); print("output.txt");`
	`HTTP_SetOutputFile(filename);`	

3.2.3. HTTP_SetMode(mode)

Syntax	HTTP_SetMode(mode)	
Arguments	mode	
	mode	Specifies the HTTP mode to use: GET or POST request
Returns	result	
	result	Specifies whether the function was executed successfully (1) or not (0)
Description	Sets the next HTTP/HTTPS request to the specified mode.	

Keyword	Value	Description
HTTP_GET	0x00	Sets TCP request to GET request
HTTP_POST	0x01	Sets TCP request to POST request

Example

```
// Example 1:
// Sets TCP mode as GET request
HTTP_SetMode(HTTP_GET);

// Example 2:
// Sets TCP mode as POST request
HTTP_SetMode(HTTP_POST);
```

3.2.4. HTTP_SetPort(port)

Syntax	HTTP_SetPort(port)	
Arguments	port	
	port	Specifies the network port to use when sending HTTP/HTTPS request
Returns	result	
	result	Specifies whether the function was executed successfully (1) or not (0)
Description	Sets the port to use for the next HTTP/HTTPS request to the specified mode.	

Example	`// Example 1:`
	`// Sets port to 80 (commonly used for HTTP)`
	`HTTP_SetPort(80);`
	
	`// Example 2:`
	`// Sets port to 443 (commonly used for HTTPS)`
	`HTTP_SetPort(443);`	

3.2.5. HTTP_SetHost(host)

Syntax	HTTP_SetHost(host)	
Arguments	host	
	host	word aligned pointer to string containing the hostname
Returns	result	
	result	Specifies whether the function was executed successfully (1) or not (0)
Description	This function sets the hostname to connect to when **HTTP_StartRequest** is executed	

Example	`// Example 1:`
	`// Directly using strings as parameter`
	`HTTP_SetHost("www.4dsystems.com.au");`
	
	`// Example 2:`
	`// Storing the hostname before setting it`
	`var host[10];`
	`to(host); print("www.4dsystems.com.au");`
	`HTTP_SetHost(host);`	

3.2.6. HTTP_SetPath(path)

Syntax	HTTP_SetPath(path)	
Arguments	path	
	path	word aligned pointer to string containing the path
Returns	result	
	result	Specifies whether the function was executed successfully (1) or not (0)
Description	This function sets the path to access from the provided hostname when **HTTP_StartRequest** is executed	

Example	`// Example 1:`
	`// Directly using strings as parameter`
	`HTTP_SetPath("sample-path");`
	
	`// Example 2:`
	`// Storing the hostname before setting it`
	`var host[10];`
	`to(host); print("sample-path");`
	`HTTP_SetPath(path);`	

3.2.7. HTTP_SetSecure(key)

Syntax	HTTP_SetSecure(key)	
Arguments	key	
	key	Pointer to a **byte** #DATA array location containing the certificate
Returns	result	
	result	Specifies whether the function was executed successfully (1) or not (0)
Description	This function sets the certificate to use when sending an HTTPS request using **HTTP_StartRequest(1)** If ***key*** set to zero (0), HTTPS calls will ignore security checks.	
Example	See `HTTP_StartRequest`	

3.2.8. HTTP_AddData(name, value)

Syntax	HTTP_AddData(name, value)	
Arguments	name, value	
	name	word aligned pointer to string containing the name of key
	value	word aligned pointer to string containing the value of the key
Returns	result	
	result	Specifies whether the function was executed successfully (1) or not (0)
Description	This function adds a key-value pair data to be sent to the server when **HTTP_StartRequest** is executed	

Example	`// Example 1:`
	`// Directly using strings as parameters`
	`HTTP_AddData("sample-name", "sample-value");`
	
	`// Example 2:`
	`// Storing the key/name and value before setting it`
	`var name[10], value[10];`
	`to(name); print("sample-name");`
	`to(value); print("sample-value");`
	`HTTP_AddData(name, value);`	

3.2.9. HTTP_StartRequest(secure)

Syntax	HTTP_StartRequest(secure)	
Arguments	secure	
	secure	Specifies whether the function should send HTTP (0) or HTTPS (1) request
Returns	result	
	result	Specifies whether the function was executed successfully (1) or not (0)
Description	Starts the TCP request that was set by: **HTTP_SetMode**, **HTTP_SetHost**, **HTTP_SetPath** and **HTTP_AddData** The response is saved to a file in the uSD Card with filename set by: **HTTP_SetOutputFile**	
Example	`// See example projects`	

3.2.10. HTTP_GetError()

Syntax	HTTP_GetError()	
Arguments	none	
Returns	error	
	error	Error the occurred during the last HTTP request
Description	Query the ESP chip for the last HTTP request error that occurred. Possible results and their values are shown below:	

Error	Value
HTTP_ERROR_CONNECTION_SUCCESS	0
HTTP_ERROR_CONNECTION_FAILED	1
HTTP_ERROR_SEND_HEADER_FAILED	2
HTTP_ERROR_SEND_PAYLOAD_FAILED	3
HTTP_ERROR_NOT_CONNECTED	4
HTTP_ERROR_CONNECTION_LOST	5
HTTP_ERROR_NO_STREAM	6
HTTP_ERROR_NO_HTTP_SERVER	7
HTTP_ERROR_TOO_LESS_RAM	8
HTTP_ERROR_ENCODING	9
HTTP_ERROR_STREAM_WRITE	10
HTTP_ERROR_READ_TIMEOUT	11

Example

```
var espError, httpError;
espError := ESP_GetError();

if (espError == HTTP_REQUEST_ERROR)
    httpError := HTTP_GetError();
endif
```

3.3. User Datagram (UDP) Functions

This section discusses about the functions used to utilize UDP when using ESP32 in **WIFI_STATION** mode. See **WiFi_SetMode(mode)** for details.

Below is a list of functions discussed in this section:

- UDP_Begin
- UDP_GetPort
- UDP_BeginPacket
- UDP_Write
- UDP_EndPacket
- UDP_SendPacket
- UDP_GetPacket
- UDP_ReturnPacket
- UDP_GetRemoteIP
- UDP_GetRemotePort

3.3.1. UDP_Begin(port)

Syntax	UDP_Begin(port)	
Arguments	port	
	port	Specifies the local port to listen on
Returns	result	
	result	1 if successful, 0 if there are no sockets available to use
Description	Initializes the UDP service and network settings	
Example	`var localPort := 9940;` `UDP_Begin(localPort);`	

3.3.2. UDP_GetPort()

Syntax	UDP_GetPort()	
Arguments	none	
Returns	port	
	port	Specifies the port currently listening from
Description	Returns the local port specified previously using **UDP_Begin(port)**	

Example	`var localPort := 9940;`
	`UDP_Begin(localPort);`
	
	`if (UDP_GetPort() == localPort)`
	`  print("UDP Started Successfully");`
	`endif`	

3.3.3. UDP_BeginPacket(address, port)

Syntax	UDP_BeginPacket(address, port)	
Arguments	address, port	
	address	Specifies the IP address of the remote connection
	port	Specifies the port of the remote connection
Returns	result	
	result	1 if successful, 0 if there was a problem resolving the hostname or port
Description	Starts a connection to write UDP data to the remote connection	

Example	`UDP_BeginPacket("192.168.1.123", 9940);`
	`UDP_Write('A');`
	`UDP_EndPacket();`	

3.3.4. UDP_Write(byte)

Syntax	UDP_Write(byte)	
Arguments	byte	
	byte	Specifies a character or byte to send
Returns	none	
Description	Writes a UDP character to the remote connection. Must be wrapped between **UDP_BeginPacket(address, port)** and **UDP_EndPacket()**. UDP_BeginPacket initializes the packet of data. The data is not sent until UDP_EndPacket is called.	

Example	`UDP_BeginPacket("192.168.1.123", 9940);`
	`UDP_Write('A');`
	`UDP_EndPacket();`	

3.3.5. UDP_EndPacket()

Syntax	UDP_EndPacket()
Arguments	none
Returns	none
Description	Called to finalize writing of data to the remote connection. The data sent using **UDP_Write(byte)** is not sent until this function is called.

Example	`UDP_BeginPacket("192.168.1.123", 9940);`
	`UDP_Write('A');`
	`UDP_EndPacket();`

3.3.6. UDP_SendPacket(address, port, packet)

Syntax	UDP_SendPacket(address, port, packet)	
Arguments	address, port, packet	
	address	Specifies the IP address of the remote connection
	port	Specifies the port of the remote connection
	packet	Specifies the string to be written to the remote connection
Returns	none	
Description	Starts a connection to write UDP data to the remote connection, write the string and ends the write. Carries out the functions of the **UDP_BeginPacket(address, port), UDP_Write(byte)** and **UDP_EndPacket()** in a single command.	
Example	`// Example: Sending a string` `UDP_SendPacket("192.168.1.123", 9940, "Hello World");`	

3.3.7. UDP_GetPacket()

Syntax	UDP_GetPacket()	
Arguments	none	
Returns	length	
	length	length of incoming packet, 0 indicates no packet received
Description	Starts a connection to write UDP data to the remote connection	

Example	`var length;`
	`length := UDP_GetPacket();`
	
	`if (length > 0)`
	`  putstr(ESPbuffer);`
	`endif`	

3.3.8. UDP_ReturnPacket(packet)

Syntax	UDP_ReturnPacket(packet)	
Arguments	packet	
	packet	Specifies the string to be written to the remote connection
Returns	none	
Description	Returns a packet to the last received remote IP address and port	

Example	`var length;`
	`length := UDP_GetPacket();`
	
	`if (length > 0)`
	`  putstr(ESPbuffer);`
	`  UDP_ReturnPacket("Received Packet");`
	`endif`	

3.3.9. UDP_GetRemoteIP()

Syntax	UDP_GetRemoteIP()	
Arguments	none	
Returns	address	
	address	Specifies the IP address of the last incoming UDP connection
Description	Returns the IP address of the last incoming UDP connection	

Example	`var length;`
	`length := UDP_GetPacket();`
	
	`if (length > 0)`
	`  putstr(ESPbuffer);`
	`  print("\r\n");`
	`  print([STR]UDP_GetRemoteIP(), "\r\n");`
	`  print(UDP_GetRemotePort(), "\r\n");`
	`endif`	

3.3.10. UDP_GetRemotePort()

Syntax	UDP_GetRemotePort()	
Arguments	none	
Returns	port	
	port	Specifies the port of the last incoming UDP connection
Description	Returns the port of the last incoming UDP connection	

Example	`var length;`
	`length := UDP_GetPacket();`
	
	`if (length > 0)`
	`  putstr(ESPbuffer);`
	`  print("\r\n");`
	`  print(UDP_GetRemoteIP(), "\r\n");`
	`  print(UDP_GetRemotePort(), "\r\n");`
	`endif`	

3.4. Network Time (NTP) Functions

This section discusses about the functions used to utilize NTP when using ESP32 in **WIFI_STATION** mode. See **WiFi_SetMode(mode)** for details.

Below is a list of functions discussed in this section:

- NTP_Start
- NTP_GetDateFAT
- NTP_GetTimeFAT
- NTP_GetYear
- NTP_GetMonth
- NTP_GetDayOfMonth
- NTP_GetHours
- NTP_GetMinutes
- NTP_GetSeconds

3.4.1. NTP_Start(server, timezone, port)

Syntax	NTP_Start(server, timezone, port)	
Arguments	server, timezone, port	
	server	String that specifies the NTP server to synchronize with
	timezone	Specifies the UTC offset in hours
	port	Specifies the port used by the NTP server
Returns	result	
	result	Specifies whether the function was executed successfully (1) or not (0)
Description	Initializes the network time service by syncing with the specified *server* through the specified port. The *time zone* can be set by setting the number of hours offset from UTC.	

Example	`// Example 1:`
	`// Starts NTP service in UTC+8 using "pool.ntp.org" as server`
	`// and using default port`
	`NTP_Start("pool.ntp.org", 8, 1337);`
	
	`// Example 2:`
	`// Starts NTP service in UTC-7 using "pool.ntp.org" as server`
	`// and using default port`
	`NTP_Start("pool.ntp.org", -7, 1337);`	

3.4.2. NTP_GetDateFAT()

Syntax	NTP_GetDateFAT()	
Arguments	none	
Returns	FAT Date	
	FAT Date	Specifies network date in FAT format
Description	Returns the current network date in FAT format **Note:** The NTP service should be previously started using **NTP_Start**	
Example	`var FileFatDate;` `FileFatDate := NTP_GetDateFAT();`	

3.4.3. NTP_GetTimeFAT()

Syntax	NTP_GetTimeFAT()	
Arguments	none	
Returns	FAT Time	
	FAT Time	Specifies network time in FAT format
Description	Returns the current network time in FAT format **Note:** The NTP service should be previously started using **NTP_Start**	
Example	`var FileFatTime;` `FileFatTime := NTP_GetTimeFAT();`	

3.4.4. NTP_GetYear()

Syntax	NTP_GetYear()	
Arguments	none	
Returns	year	
	year	Specifies network year
Description	Returns the current network year **Note:** The NTP service should be previously started using **NTP_Start**	
Example	`var year;` `year := NTP_GetYear();`	

3.4.5. NTP_GetMonth()

Syntax	NTP_GetMonth()	
Arguments	none	
Returns	month	
	month	Specifies network month
Description	Returns the current network month **Note:** The NTP service should be previously started using **NTP_Start**	
Example	`var month;` `month := NTP_GetMonth();`	

3.4.6. NTP_GetDayOfMonth()

Syntax	NTP_GetDayOfMonth()	
Arguments	none	
Returns	dayOfMonth	
	dayOfMonth	Specifies network day of month
Description	Returns the current network month **Note:** The NTP service should be previously started using **NTP_Start**	
Example	`var day;` `day := NTP_GetDayOfMonth();`	

3.4.7. NTP_GetHours()

Syntax	NTP_GetHours()	
Arguments	none	
Returns	hours	
	hours	Specifies the hour of day in 24-hour format
Description	Returns the current hour of day in 24-hour format **Note:** The NTP service should be previously started using **NTP_Start**	
Example	`var hour;` `hour := NTP_GetHours();`	

3.4.8. NTP_GetMinutes()

Syntax	NTP_GetMinutes()	
Arguments	none	
Returns	hours	
	hours	Specifies the number of minutes in the current hour
Description	Returns the number of minutes in the current hour **Note:** The NTP service should be previously started using **NTP_Start**	
Example	`var minute;` `minute := NTP_GetMinutes();`	

3.4.9. NTP_GetSeconds()

Syntax	NTP_GetSeconds()	
Arguments	none	
Returns	seconds	
	seconds	Specifies the number of seconds counting to the next minute
Description	Returns the number of seconds counting to the next minute **Note:** The NTP service should be previously started using **NTP_Start**	
Example	`var second;` `second:= NTP_GetSeconds();`	

3.5. Alexa Functions

This section discusses about the functions used to integrate Alexa devices when using ESP32 in **WIFI_STATION** mode. See **WiFi_SetMode(mode)** for details.

Below is a list of functions discussed in this section:

- Alexa_Enable
- Alexa_AddDevice
- Alexa_ReadState
- Alexa_ReadValue

3.5.1. Alexa_Enable(enable)

Syntax	Alexa_Enable(enable)	
Arguments	enable	
	enable	Specifies if Alexa service is to be enabled (1) or disabled (0)
Returns	none	
Description	Enables or disables the Alexa compatible device service. The service must be enabled before adding an Alexa device.	

Example	`Alexa_Enable(1);`
	`Alexa_AddDevice(0, "Fan Speed");`
	`Alexa_AddDevice(1, "Button 1");`	

3.5.2. Alexa_AddDevice(num, command)

Syntax	Alexa_AddDevice(num, command)	
Arguments	num, command	
	num	Specifies the Alexa device number
	command	Specifies the Alexa device command
Returns	result	
Description	Adds a device to the Alexa compatible device service. The device is identified by a numerical value (**num**) which must in be numerical sequence and a text name identifiable to Alexa which will be the **command** you say to Alexa. On a correctly configured Alexa, the device can be controlled by saying the **command** followed the state (ON or OFF) or value (0 to 100) after a device discovery on the Alexa has been performed **Note: Alexa_Enable(enable)** must be called before adding a device.	

Example	`Alexa_Enable(1);`
	`Alexa_AddDevice(0, "Fan Speed");`
	`Alexa_AddDevice(1, "Button 1");`	

3.5.3. Alexa_ReadState(num)

Syntax	Alexa_ReadState(num)	
Arguments	num	
	num	Specifies the Alexa device number
Returns	result	
	result	Specifies the last state of the Alexa device
Description	This function will check the state of the last Alexa command of the number identified device. A device can have both a state and a value, but this command is typically used for a 2-state (ON/OFF) device.	

Example	`Alexa_Enable(1);`
	`Alexa_AddDevice(0, "Button 1");`
	
	`var result;`
	`result := Alexa_ReadState(0);`	

3.5.4. Alexa_ReadValue(ADnum)

Syntax	Alexa_ReadValue(num)	
Arguments	num	
	num	Specifies the Alexa device number
Returns	result	
	result	Specifies the last value of the Alexa device
Description	This function will check the value of the last Alexa command of the number identified device. A device can have both a state and a value, but this command is typically used for a device that has a value range of 0 to 100.	

Example	`Alexa_Enable(1);`
	`Alexa_AddDevice(0, "Room Temperature");`
	
	`var result;`
	`result := Alexa_ReadValue(0);`	

4. Bluetooth Functions

This section contains functions that are used for any ESP32's Bluetooth functionality.

Below is a list of Bluetooth features discussed in this section:

- Serial-over-Bluetooth

4.1. Serial-over-Bluetooth

The ESP32's Bluetooth feature can be setup to send and receive data wirelessly and behave similarly to UART.

Below is a list of functions discussed in this section:

- BT_Begin
- BT_Available
- BT_Read
- BT_Write
- BT_WriteArray
- BT_Print
- BT_Println

4.1.1. BT_Begin(name)

Syntax	BT_Begin(name)	
Arguments	name	
	name	Specifies the Bluetooth device name to use
Returns	result	
	result	Specifies whether the function was executed successfully (1) or not (0)
Description	Starts Serial-over-Bluetooth services as a Bluetooth device with the name as specified	

Example	`// Example 1:`
	`// Directly using string as parameter`
	`var res;`
	`res := BT_Begin("4Discovery-5.0");`
	
	`// Example 2:`
	`// Storing the Bluetooth name before setting it`
	`var name[10], res;`
	`to(name); print("4Discovery-5.0");`
	`res := BT_Begin(name);`	

4.1.2. BT_Available()

Syntax	BT_Available()	
Arguments	none	
Returns	count	
	count	Specifies the number of bytes received through Bluetooth Serial
Description	Checks the number of bytes currently available for reading from the Serial-over-Bluetooth service	
Example	`var count;` `count := BT_Available();`	

4.1.3. BT_Read()

Syntax	BT_Read()	
Arguments	none	
Returns	byte	
	byte	Specifies the byte received through Bluetooth Serial
Description	Reads a single byte received by the Bluetooth service	
Example	`var Byte;` `Byte := BT_Read();`	

4.1.4. BT_Write(byte)

Syntax	BT_Write(byte)	
Arguments	byte	
	byte	Specifies the byte to send through Bluetooth Serial
Returns	none	
Description	Send the specified byte using Serial-over-Bluetooth service	
Example	`BT_Write(0x06);`	

4.1.5. BT_WriteArray(array, length)

Syntax	BT_WriteArray(array, length)	
Arguments	array, length	
	array	Specifies an array located in Flash or RAM
	length	Specifies the number of bytes to send using Bluetooth Serial
Returns	none	
Description	Sends a number of bytes as specified by **length** from the given **array** using the Serial-over-Bluetooth service	

Example	`// Example 1: Using #DATA stored in Flash`
	`// Global Scope`
	`#DATA`
	`  byte arrayFlash  0x30, 0x31, 0x32, 0x33, 0x34,`
	`  0x35, 0x36, 0x37, 0x38, 0x39`
	`#END`
	
	`// Write Data Array`
	`BT_WriteArray(arrayFlash, 10);`
	
	`// Example 2: Using data stored in RAM`
	`var arrayRam[5];`
	`arrayRam[0] := 0x3130;`
	`arrayRam[1] := 0x3332;`
	`arrayRam[2] := 0x3534;`
	`arrayRam[3] := 0x3736;`
	`arrayRam[4] := 0x3938;`
	
	`// Write Data Array`
	`BT_WriteArray(arrayRam, 10);`	

4.1.6. BT_Print(str)

Syntax	BT_Print(str)	
Arguments	str	
	str	Specifies the string to send through Bluetooth Serial
Returns	none	
Description	Sends a string using the Serial-over-Bluetooth service	

Example	`// Example 1:`
	`// Directly using string as parameter`
	`var res;`
	`res := BT_Print("4Discovery-5.0");`
	
	`// Example 2:`
	`// Storing the string before sending it`
	`var str[10], res;`
	`to(str); print("4Discovery-5.0");`
	`res := BT_ Print(str);`	

4.1.7. BT_Println(str)

Syntax	BT_Println(str)	
Arguments	str	
	str	Specifies the string to send through Bluetooth Serial
Returns	none	
Description	Sends a string followed by a newline (Carriage Return and Line Feed) using the Serial-over-Bluetooth service	

Example	`// Example 1:`
	`// Directly using string as parameter`
	`var res;`
	`res := BT_Println("4Discovery-5.0");`
	
	`// Example 2:`
	`// Storing the string before sending it`
	`var str[10], res;`
	`to(str); print("4Discovery-5.0");`
	`res := BT_ Println(str);`	

5. Revision History

Revision No.	Description	Date
1.0	Initial Revision	27/08/2020
1.1	Removed ESP8266 support and unreleased 4Discovery modules	02/03/2025

6. Legal Notice

6.1. Proprietary Information

The information contained in this document is the property of 4D Labs Semiconductors and may be the subject of patents pending or granted and must not be copied or disclosed without prior written permission.

4D Labs Semiconductors endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Labs Semiconductors products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Labs Semiconductors. 4D Labs Semiconductors reserves the right to modify, update or makes changes to Specifications or written material without prior notice at any time.

All trademarks belong to their respective owners and are recognised and acknowledged.

6.2. Disclaimer of Warranties & Limitation of Liability

4D Labs Semiconductors makes no warranty, either expressed or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

Images and graphics used throughout this document are for illustrative purposes only. All images and graphics used are possible to be displayed on the 4D Labs Semiconductors range of products, however the quality may vary.

In no event shall 4D Labs Semiconductors be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Labs Semiconductors, or the use or inability to use the same, even if 4D Labs Semiconductors has been advised of the possibility of such damages.

4D Labs Semiconductors products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail – safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage (‘High Risk Activities’). 4D Labs Semiconductors and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Use of 4D Labs Semiconductors’ products and devices in 'High Risk Activities' and in any other application is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless 4D Labs Semiconductors from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Labs Semiconductors intellectual property rights.

7. Contact Information

For Technical Support: www.4dlabs.com.au/support
For Sales Support: sales@4dlabs.com.au
Website: www.4dlabs.com.au