
USB Printer Class on an Embedded Host

<p>Author: Kim Otten Microchip Technology Inc.</p>
--

INTRODUCTION

Typical embedded applications are rather restricted in how they can present data to a user. Limited amounts of data can be displayed on an LCD or a graphical display, but large amounts of data are more problematical. Often, data is transferred via an electrical interface or via electronic storage media, such as a thumb drive, for further examination, but it can be handy to obtain an immediate snapshot of the data. The USB Printer Class, used on one of Microchip's microcontrollers with the USB OTG peripheral, allows an embedded application to utilize a USB printer to provide hardcopy output for quick review and possible archival.

Point-of-Sale (POS) printers, sometimes referred to as receipt printers or label printers, can also be very useful in the field. Their small size, often coupled with mobile, battery-powered operation, allows them to be used in locations that are not suitable for a full sheet printer, or even installed in the end product. They also offer extra benefits, including easy bar code generation. This USB Printer Class implementation also offers the ability to utilize these printers.

In this document, the term "full sheet printer" will be used for printers that typically print on standard, 8.5"x11" paper, and the term "POS printer" will be used for Point-of-Sale printers.

USB PRINTER CLASS

The "USB Device Class Definition for Printing Devices" specification, available from the USB Implementers Forum, defines the configuration, interface and endpoint descriptors, as well as the communications protocol used to communicate with a USB printer. It does not define the actual commands used to control the printers.

The USB Printer Class, like all other USB classes, utilizes Endpoint 0 for control transfers to and from the host. In addition, the Printer Class uses one Bulk OUT endpoint to send data to the printer, and sometimes one Bulk IN endpoint for status and other data received from the printer.

The class, subclass and protocol designators for a printing device are not contained in the bDeviceClass, bDeviceSubClass and bDeviceProtocol fields of the device descriptor. Instead, these fields are all set to 0x00, and the designators are specified in the bInterfaceClass (0x07), bInterfaceSubClass (0x01) and bInterfaceProtocol fields of the interface descriptor. Printers support three possible interface protocols, with only one enabled at any given time:

- Unidirectional interface (0x01): This interface supports only the sending of data to the printer via a Bulk OUT endpoint.
- Bidirectional interface (0x02): This interface supports sending data to the printer via the Bulk OUT endpoint, and receiving status and other information from the printer via the Bulk IN endpoint.
- IEEE 1284.4 compatible bidirectional interface (0x03): This interface supports sending data to the printer via the Bulk OUT endpoint, and receiving status and other information from the printer via the Bulk IN endpoint. It also specifies that the data will be transmitted to and from the device using the 1284.4 protocol.

Some printing devices indicate custom driver support in their interface descriptors rather than Printer Class support, but utilize the same basic Bulk OUT communication method. The USB Printer Client Driver implementation allows an application to utilize the same interface, with minor limitations, described below.

CLASS-SPECIFIC REQUESTS

The USB Printer Class specifies three class-specific requests. These requests are described in Table 1.

TABLE 1: PRINTER CLASS-SPECIFIC REQUEST

Name	bmRequestType	bRequest	wValue	wIndex	wLength	Data
GET DEVICE ID	0xA1	0	Config Index	Interface and Alternate Setting	Maximum Length	1284 Device ID String
GET PORT STATUS	0xA1	1	0	Interface	1	BYTE
SOFT RESET	0x21	2	0	Interface	0	None

Get Device ID

This request returns a device ID string that is compatible with IEEE 1284. The first two bytes are the length of the string, including the two length bytes, in big endian format. The string is comprised of a series of keys and values in the form:

key: value[, value];

The keys are case-sensitive. At a minimum, the following keys must be provided by the printing device:

- MANUFACTURER
- COMMAND SET
- MODEL

For example, a portion of the device ID string for the Lexmark E250dn is:

```
MANUFACTURER:Lexmark International;  
COMMAND SET:PCL 6 Emulation, PostScript  
Level 3 Emulation, NPAP, PJI;  
MODEL:Lexmark E250dn;
```

Of these keys, only COMMAND SET is utilized by the Microchip USB Printer Client Driver.

This request cannot be used with printers that indicate customer driver support in their interface descriptor.

Get Port Status

This request returns the printer's current status in a single byte that is compatible with the status register of a standard PC parallel port, as shown in Table 2.

TABLE 2: PRINTER PORT STATUS

Bit(s)	Field	Description
7..6	Reserved	Reserved for future use; device shall return these bits reset to '0'.
5	Paper Empty	1 = Paper Empty, 0 = Paper Not Empty
4	Select	1 = Selected, 0 = Not Selected
3	Not Error	1 = No Error, 0 = Error
2..0	Reserved	Reserved for future use; device shall return these bits reset to '0'.

The Printer Client Driver provides a function to request the printer status:

```
BYTE USBHostPrinterGetStatus  
( BYTE deviceAddress, BYTE *status );
```

This request cannot be used with printers that indicate customer driver support in their interface descriptor.

Soft Reset

This request flushes all buffers, resets the Bulk OUT and Bulk IN pipes to their default states and clears all stall conditions. It does not change the USB addressing or configuration.

The Printer Client Driver provides a function to issue the soft Reset command:

```
BYTE USBHostPrinterReset
( BYTE deviceAddress );
```

This request cannot be used with printers that indicate customer driver support in their interface descriptor.

PRINTER LANGUAGES

The USB specification describes how to send data to the printer, but it does not specify the data itself. The data requirements for a printer are described by the Page Description Language(s) specified in the COMMAND SET portion of the device ID string described above. If a printer indicates custom driver support in its interface, then the printer language cannot be determined via the device ID string, and must be known and specified explicitly.

There are a wide variety of Page Description Languages (PDLs) used to support the vast number of available printers. Many low-end printers receive only binary data, relying on the USB host (usually a PC) to perform the memory and computational intensive processing required to determine exactly how to print a page. These printers are not conducive to an embedded application, where memory and processing resources are fixed. Many manufactures also provide printers with higher level Page Description Language support. These printers are a much better fit for an embedded application, as they allow an embedded host to shift some of the resource burden to the printer.

Currently, Microchip provides support for the following languages:

- **PostScript** – Created by Adobe Systems, PostScript is an interpreted, stack-based language with similarities to Forth and Lisp. Elegant, complex graphics can be described easily (full sheet printers only).

- **PCL 5** – Created by HP, there have been several versions of PCL, not all of which are compatible. In general, PCL 5 utilizes ASCII rather than binary data (as utilized by PCL 6), and introduced support for vector (HP-GL/2) graphics. PCL 3 is a limited subset of PCL 5, without vector graphics and limited landscape support (full sheet printers only).
- **ESC/POS** – Created by Seiko Epson, ESC/POS is tailored for Point-of-Sale applications, and is used by a wide variety of receipt and label printers available from Seiko Epson and other manufacturers. While the basic commands are consistent across printers, not all printers support all commands, and command parameters can differ across printers. If using a POS printer, consider limiting support to explicit models via the device VID/PID, and test the application with those printers to ensure accurate output. Some minor command modifications may be required (POS printers only).

<p>Note: Please refer to the Help file documentation installed with the USB Embedded Host Stack for implementation limitations.</p>
--

In general, there are three types of printed output:

- Text
- Bit-mapped (raster) images
- Vector graphics

Not all printer languages support all types of printed output. Many printers support text and bit-mapped images, but not vector graphics. Be sure to determine the type of printed output you want to produce, and ensure that the target printers support that output.

Creating Custom Printer Language Support

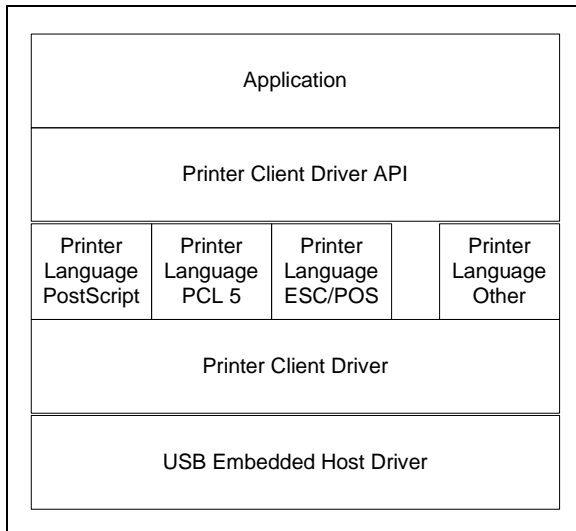
Additional printer language support can be added relatively easily by creating a language driver that has the same interface as the existing language drivers. Refer to the Help file documentation installed with the USB Embedded Host Stack for details about the interface. Note that the application will have to be manually configured to utilize the custom language support.

USING THE PRINTER CLIENT DRIVER

Application Architecture

An application that utilizes the Microchip USB Embedded Host Printer Client Driver has an architecture described by Figure 1.

FIGURE 1: APPLICATION ARCHITECTURE



Selecting Printer Languages

An application can support one or more printer languages. If the application targets a single printer, then it can specify only the printer language used by the target printer. If multiple printers may be used, then the application may want to include multiple printer languages. At least one printer language must be included in the application.

The Printer Client Driver will first see if a printer language has been explicitly selected for an attaching printer. If the printer does not indicate Printer Class support in its interface descriptor, then the printer language must be specified explicitly. Otherwise, the Printer Client Driver can automatically select a printer language for an attaching printer based on the printer languages included in the application and the printer language support published by the printer in its device ID string. In this case, the actual language selection is transparent to the application. If a printer supports multiple languages, the application may choose to specify which language to use to assure consistent behavior.

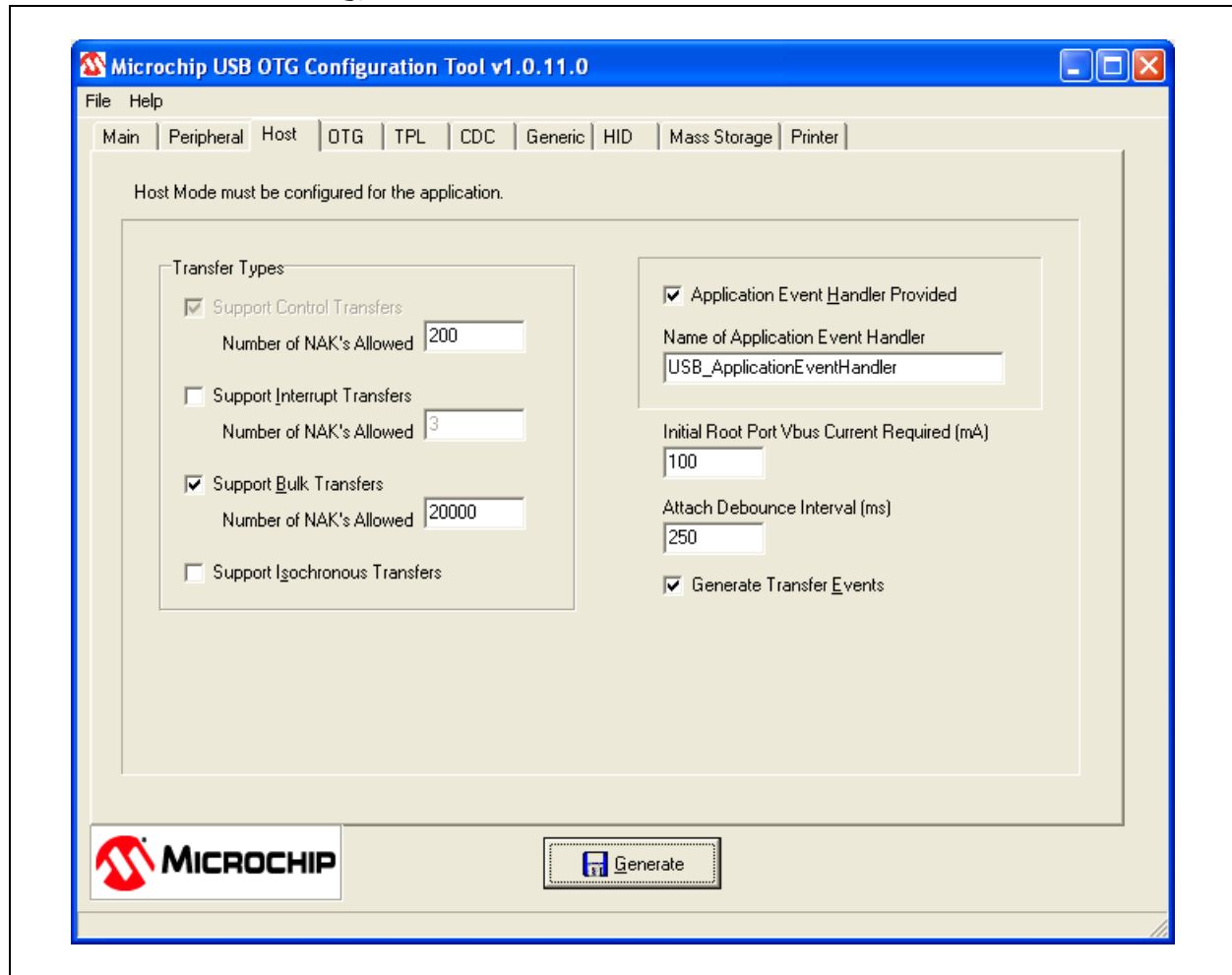
Note: Many printers, especially POS printers, do not advertise support for the Printer Class in their descriptors, and therefore, do not support the device request to obtain the device ID string. Also, many POS printers that do advertise Printer Class support do not have a standard format for specifying the printer language (ESC/POS) within their device ID string. Finally, the exact implementation of the printer language (ESC/POS) varies slightly between different manufacturers and models, and requires compile-time configuration. Therefore, for POS printers, it is recommended to target a specific printer make and model, specify that model explicitly via its VID and PID in the TPL, and explicitly select the printer language for that printer.

Configuring the Printer Client Driver

Use the USB configuration tool, USBConfig.exe, or the USB library configuration tool provided in the MPLAB® IDE VDI to configure the Printer Client Driver.

Printer Class devices utilize bulk transfers, so ensure that the **Support Bulk Transfers** checkbox is checked. The Printer Client Driver utilizes transfer events from the USB Embedded Host driver, so be sure to check the **Generate Transfer Events** checkbox on the **Host** tab.

FIGURE 2: USBConfig, HOST TAB



Select the **TPL** tab and add support for the required printers. Explicit support for the Epson TM-T88IV is shown in Figure 3. Generic printer support can be specified by the entries shown in Figure 4.

FIGURE 3: USBConfig, TPL TAB – EXPLICIT PRINTER SUPPORT

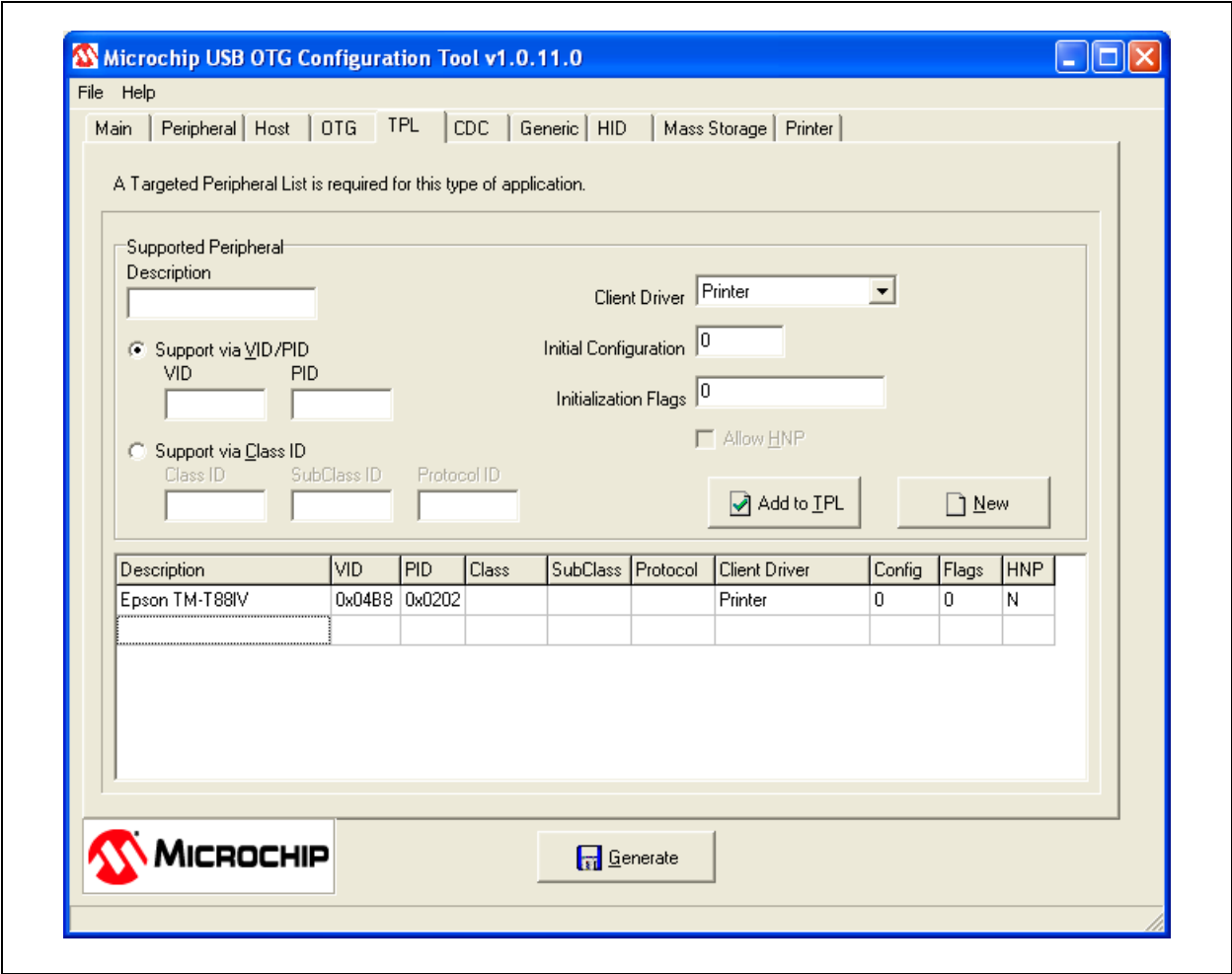


FIGURE 4: USBConfig, TPL TAB – GENERIC PRINTER SUPPORT

The screenshot shows the Microchip USB OTG Configuration Tool v1.0.11.0. The TPL tab is selected, and the 'Printer' client driver is chosen. The configuration fields are as follows:

- Supported Peripheral Description: [Empty text box]
- Client Driver: Printer (dropdown menu)
- Initial Configuration: 0 (text box)
- Initialization Flags: 0 (text box)
- Support via VID/PID: [Empty text boxes for VID and PID]
- Support via Class ID: [Empty text boxes for Class ID, SubClass ID, and Protocol ID]
- Allow HNP: [Unchecked checkbox]
- Buttons: Add to IPL (checked), New

A table at the bottom lists supported peripherals:

Description	VID	PID	Class	SubClass	Protocol	Client Driver	Config	Flags	HNP
Printers - Unidirectional			0x07	0x01	0x01	Printer	0	0	N
Printers - Bidirectional			0x07	0x01	0x02	Printer	0	0	N

The Microchip logo and a 'Generate' button are at the bottom of the window.

Select the **Printer** tab and check the **Printer Client is used in Host Mode** checkbox. The Printer Client Driver allows users to create a queue of printer commands, so printer commands can be issued with a minimum time delay between commands. Select the size of this queue at the **Command Queue Size** edit box.

Check the **Provide Explicit Language Selection** checkbox to specify the language for a particular printer. The configuration tool will automatically populate the **Printer** combo box with all printer devices specified by VID and PID in the TPL. Select the desired printer in the **Printer** combo box, select the printer language in the **Printer Language** combo box, adjust the **Support Flags** as required, and click **Add to Printer List**.

The **Support Flags** are:

- **Vector Graphics** – Whether or not this printer supports vector graphics. To configure support for a printer that uses the PCL 3 printer language, select PCL 5 in the **Printer Language** combo box and uncheck the **Vector Graphics** support flag.

To remove an entry from the **Explicit Printer Language Selection** list, select the entry to remove, then right click on the entry and select **Remove From List**. If all of the supported printers have explicit language support, you can uncheck the **Allow Dynamic Language Selection** checkbox to save program memory and heap space.

Verify that the required printer languages are enabled in the **Supported Printer Languages** box. If ESC/POS is supported, specify the printer model configuration file.

If the application uses the printer interface to the Microchip Graphics Library, check the **Use Graphics Library Interface** checkbox. Refer to the “**Demonstration Programs**” section for an example of using the Graphics Library with the USB Printer Client Driver.

Figure 5 illustrates how to populate the tab to support a single POS printer model, corresponding to the TPL shown in Figure 3.

FIGURE 5: USBConfig, PRINTER TAB – EXPLICIT PRINTER SUPPORT

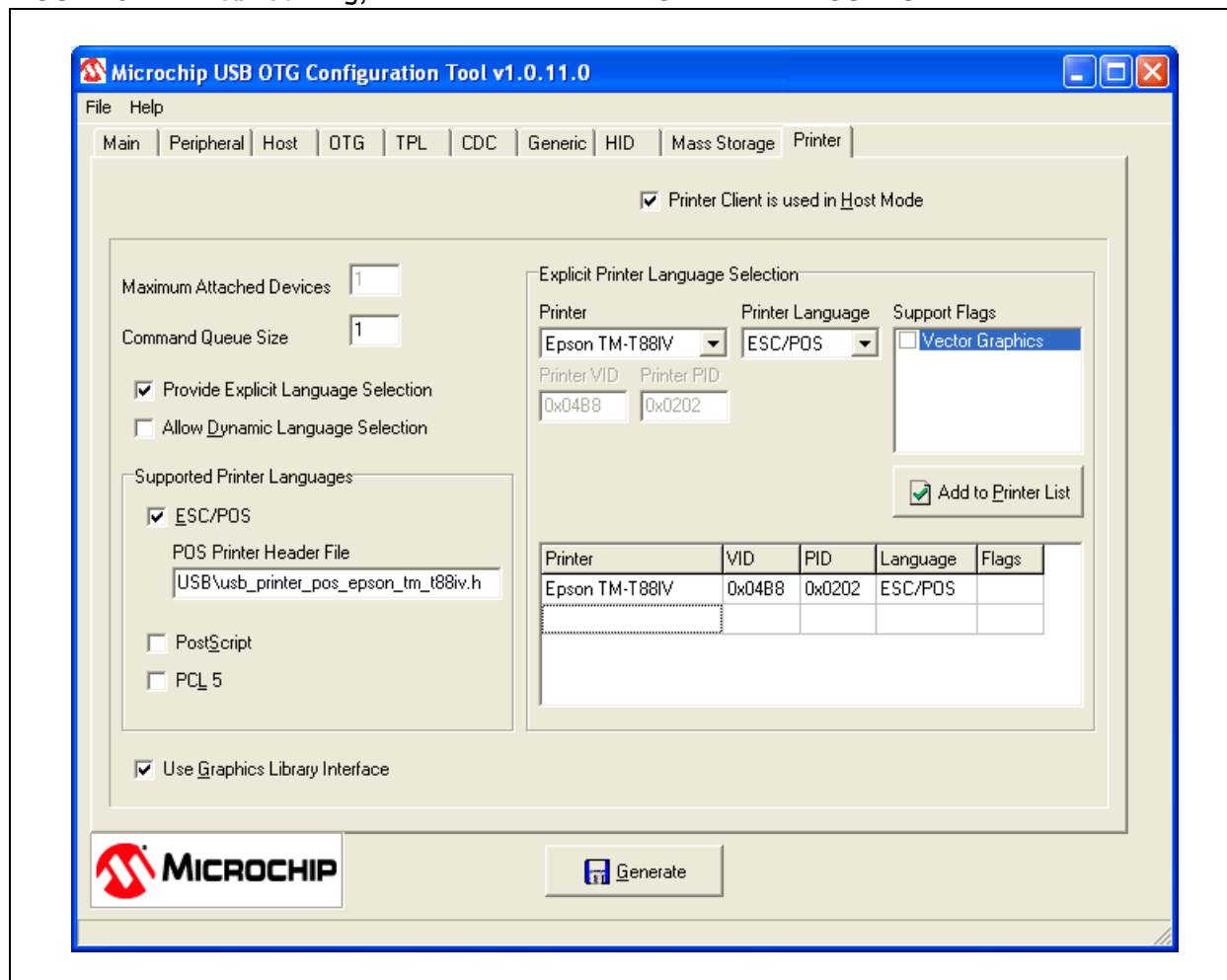
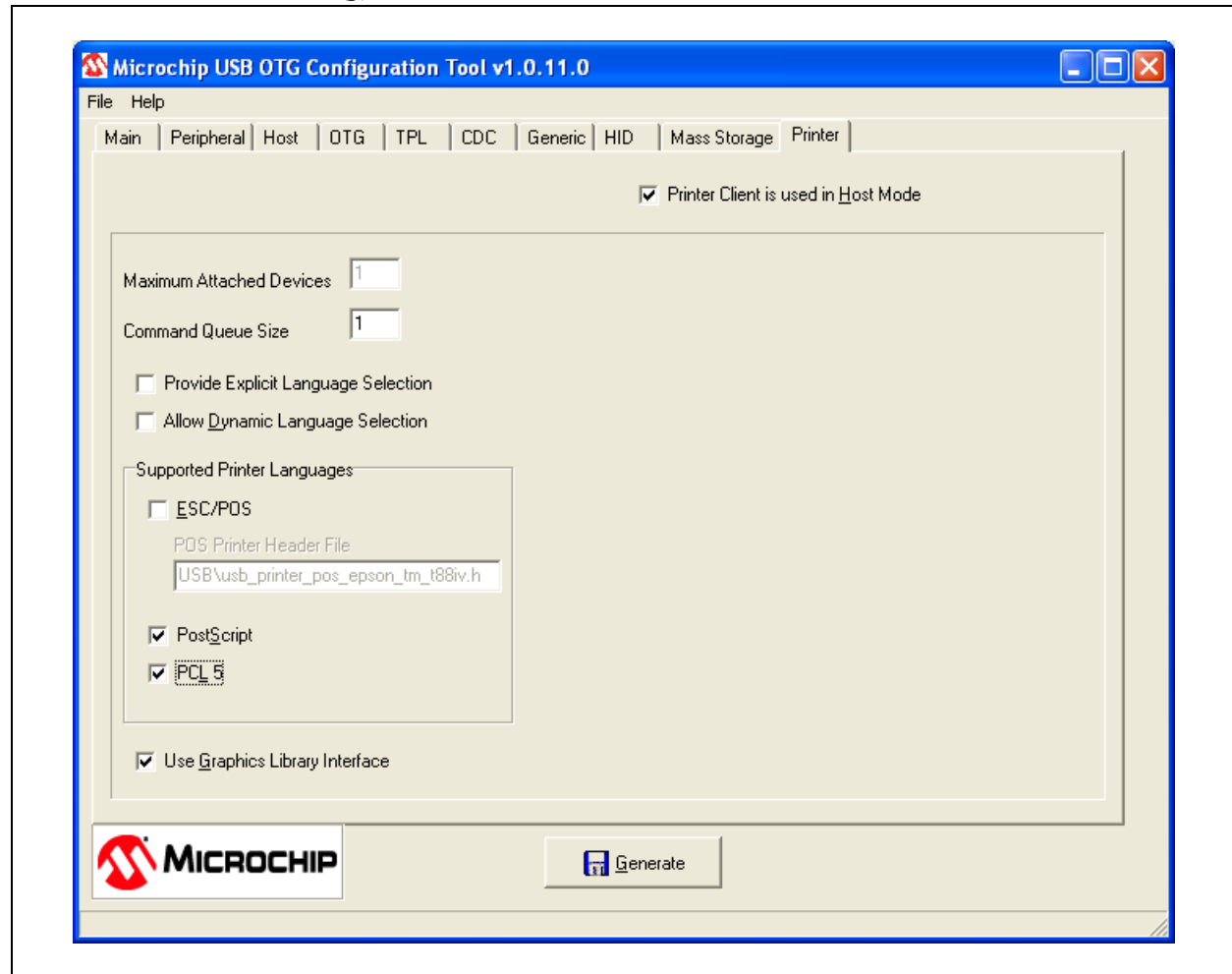


Figure 6 illustrates how to populate the tab to provide generic printer support, corresponding to the TPL shown in Figure 4. This method is not recommended for POS printers.

FIGURE 6: USBConfig, PRINTER TAB – GENERIC PRINTER SUPPORT



Printer Client Driver Events

The Printer Client Driver is an event driven USB Embedded Host client driver. It utilizes transfer events generated by the USB Embedded Host driver and sends printer events to the application. The Printer Client Driver generates the following events:

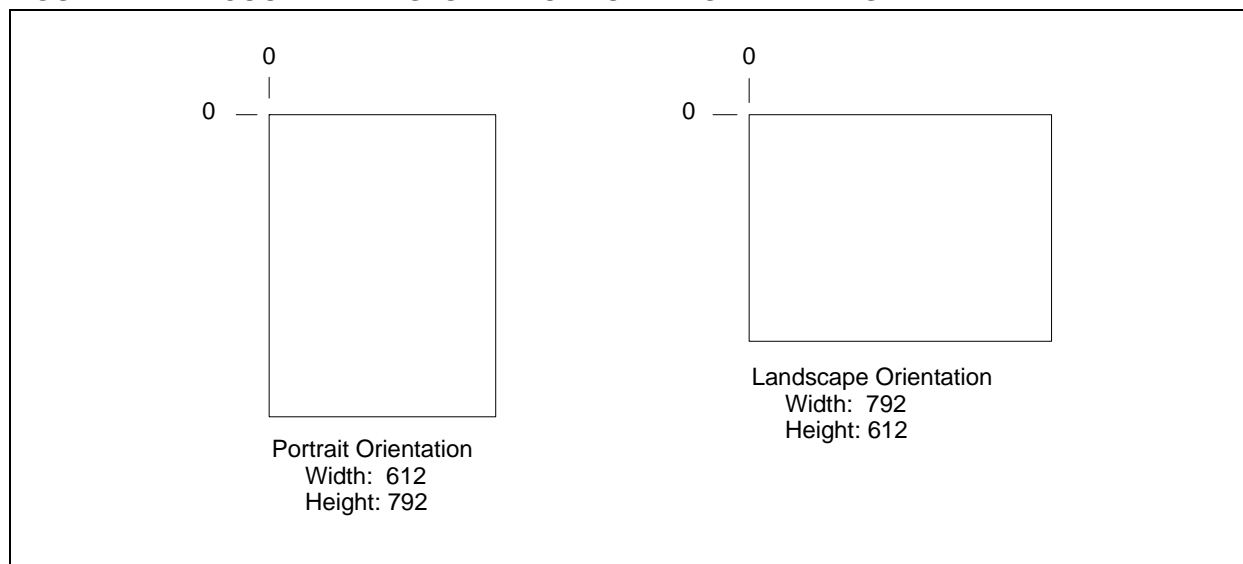
- **EVENT_PRINTER_ATTACH** – This event indicates that a printing device has successfully attached: the device enumerated correctly and utilizes one of the printer languages included in the application. This event also provides the address of the printer on the USB bus, which the application must use when sending commands to the printer.
- **EVENT_PRINTER_DETACH** – This event indicates that the printing device that was previously on the bus at the specified address is no longer attached.
- **EVENT_PRINTER_REQUEST_DONE** – This event indicates that the class-specific request initiated by the application is complete. Class-specific requests issued by the Printer Client Driver itself do not generate this event.
- **EVENT_PRINTER_RX_DONE** – This event indicates that the Bulk IN transfer initiated by the application is complete.

- **EVENT_PRINTER_TX_DONE** – This event indicates that the Bulk OUT transfer initiated by the application is complete. Refer to the “**Creating Printed Output**” section for more information on generating and suppressing this event.
- **EVENT_PRINTER_UNSUPPORTED** – This event indicates that a printer tried to attach, but either the application does not contain the printer language needed to communicate with the printer, or the application does not have enough dynamic memory (heap space) available to support the device.

The Full Sheet Printed Page

The printed output can be oriented as either portrait or landscape. The position on the paper is described by (X,Y) coordinates, with the X-axis being the horizontal position on the paper and the Y-axis being the vertical position on the paper. The origin (0,0) is located at the upper left corner of the page, regardless of the orientation (see Figure 7). The coordinate system is specified in terms of points; there are 72 points per inch. Location (72,72) is located one inch down from the top edge of the paper and one inch to the right of the left side of the paper.

FIGURE 7: COORDINATE SYSTEM FOR FULL PAGE PRINTERS



The POS Printed Page

The orientation of the POS printer output is always horizontal, beginning at the upper left corner of the paper. Currently, (X,Y) coordinates cannot be used to specify the location on the paper (this feature is planned for later implementation for applications such as label printing).

Creating Printed Output

After the application receives the event, `EVENT_PRINTER_ATTACH`, and receives the USB address of the printer, the application can generate printed output. All printing commands utilize a simple API.

Before issuing a command, see if there is room in the printer command queue to issue another printer command by calling the function:

```
BOOL USBHostPrinterCommandReady( BYTE
deviceAddress );
```

If this function returns `TRUE`, then issue the printer command by calling the function:

```
BYTE USBHostPrinterCommand( BYTE
deviceAddress, USB_PRINTER_COMMAND
command, void *data, DWORD size, BYTE
flags );
```

Note: Refer to the Help file documentation installed with the USB Embedded Host Stack for the complete list of printer commands, their usage and their required parameters and data structures.

If `USBHostPrinterCommand()` is called and there is no room in the printer command queue, the function will return an error.

If the application blocks other execution while printing, it can use a macro that combines these two commands into a single call:

```
void USBHostPrinterCommandWithReadyWait
( BYTE &returnCode, BYTE deviceAddress,
USB_PRINTER_COMMAND command,
USB_DATA_POINTER data, DWORD size, BYTE
flags );
```

While there is no space available in the printer command queue, this macro calls `USBTasks()` to perform required USB maintenance tasks. When space is available for another printer command, the macro will then call `USBHostPrinterCommand()` with the indicated parameters. Upon completion, the `returnCode` parameter contains the return code from `USBHostPrinterCommand()`.

To start a print job, issue the command, `USB_PRINTER_JOB_START`. This command tells the printer to reset back to its default state.

When printing text and bit-mapped images, the data source can be located in either RAM or ROM. The location of the data pointed to by the `data` parameter is specified in the `flags` parameter. If the data is located in RAM, the application can also request that the client driver make a copy of the data, so the application can overwrite the data area immediately after the function returns. The various allowed flags for data specification are:

- `USB_PRINTER_TRANSFER_COPY_DATA` – The client driver will make a copy of the data, allowing the application to immediately overwrite the data buffer.
- `USB_PRINTER_TRANSFER_STATIC_DATA` – The client driver will send the data directly from the specified data buffer. **If the user overwrites the data buffer before the command is actually sent to the printer, then the new data will be sent, not the data that existed when `USBHostPrinterCommand()` was called.**
- `USB_PRINTER_TRANSFER_FROM_ROM` – The data parameter points to data in ROM. In this case, the client driver will automatically make a RAM copy of the device because the USB OTG peripheral can only access data in RAM. Not all commands support transfers from ROM, so be sure to check the Help file for the allowable command parameters.
- `USB_PRINTER_TRANSFER_FROM_RAM` – (default) The data parameter points to data in RAM.

After issuing all printing commands, issue the command, `USB_PRINTER_JOB_STOP`, to print the current page, eject it and terminate the print job. By default, the event, `EVENT_PRINTER_TX_DONE`, is issued by the Printer Client Driver upon completion of the `USB_PRINTER_JOB_STOP` command only. If the application needs to see this event upon completion

of other printer commands, include `USB_PRINTER_TRANSFER_NOTIFY` in the flags parameter of the `USBHostPrinterCommand()` command. For example, to eject the current page and receive notification when the command is successfully sent to the printer, issue the following command:

EXAMPLE 1:

```
if USBHostPrinterCommandReady( deviceAddress )
{
    USBHostPrinterCommand( deviceAddress,
        USB_PRINTER_EJECT_PAGE, USB_NULL, 0,
        USB_PRINTER_TRANSFER_NOTIFY );
}
```

General Printing – Full Sheet Only

Before issuing any printing commands, establish the page orientation by issuing either the command, `USB_PRINTER_ORIENTATION_PORTRAIT`, or the command, `USB_PRINTER_ORIENTATION_LANDSCAPE`. If printing commands are issued before establishing the paper orientation, the printed output may not be correct.

Before printing text, images and many graphic items, tell the printer the page location of each item with the command, `USB_PRINTER_SET_POSITION`.

If the application prints multiple output pages within a single print job, issue the command, `USB_PRINTER_EJECT_PAGE`, to print the current page, eject it and proceed to the next page. After this command, all orientation, font and line type commands must be reissued. If the output consists of only a single page, this command should be omitted; issuing this command may result in an extra, blank page.

General Printing – POS Only

With POS printers, items are printed as soon as they are received by the printer. Therefore, it is important to send all setup commands before sending any printing commands.

POS printers are fundamentally single line printers, though graphics printing is also supported. The printer can automatically print text, graphics and bar codes (if supported) as either left, center or right justified by sending one of the following commands:

- `USB_PRINTER_POS_JUSTIFICATION_LEFT`
- `USB_PRINTER_POS_JUSTIFICATION_CENTER`
- `USB_PRINTER_POS_JUSTIFICATION_RIGHT`

Many POS printers feature an automatic cutter. Use the command, `USB_PRINTER_POS_CUT_PARTIAL` or `USB_PRINTER_POS_CUT`, to feed and partially, or completely, cut the paper.

Printing Text – Full Sheet Only

Before printing text, issue the following commands to describe the text:

- `USB_PRINTER_FONT_NAME` – Name of the font. Refer to the Help file for the list of available fonts.
- `USB_PRINTER_FONT_SIZE` – Size of the font in points.
- `USB_PRINTER_FONT_ITALIC` or `USB_PRINTER_FONT_UPRIGHT` – Font inclination.
- `USB_PRINTER_FONT_BOLD` or `USB_PRINTER_FONT_MEDIUM` – Font weight.

Upon receiving these commands, the printer will select the best match from its internally supported fonts. If the printer does not support the explicit font specified, it will select the closest possible match. The exact choice is printer dependent. In general, PostScript printers provide the most complete font support.

For maximum compatibility with various printers, text printing requires three commands:

- `USB_PRINTER_TEXT_START` – Prepares the printer for printing text.
- `USB_PRINTER_TEXT` – Sends the actual text to print.
- `USB_PRINTER_TEXT_STOP` – Terminates the text print.

These commands must be issued contiguously. No other commands can be inserted between these commands.

The text will be printed using the currently selected font, at the current printer position. The location of the actual text to print is specified by the flags passed to the printer command function.

Embedded carriage returns and line feeds are handled differently by the different printer languages. For maximum compatibility, use the text printing sequence for

each line of text, specifying the text location with the command, `USB_PRINTER_SET_POSITION`, before each line.

EXAMPLE 2:

```
USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_JOB_START, USB_NULL, 0, 0 );

USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_FONT_NAME, USB_NULL, USB_PRINTER_FONT_COURIER, 0 );
USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_FONT_SIZE, USB_NULL, (DWORD)24, 0 );
USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_FONT_BOLD, USB_NULL, 0, 0 );

USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_SET_POSITION, USB_NULL,
    USBHostPrinterPosition(100, 100), 0 );
USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_TEXT_START, USB_NULL, 0, 0 );
USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_TEXT, USB_DATA_POINTER_RAM(buffer), strlen(buffer), 0 );
USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_TEXT_STOP, USB_NULL, 0, 0 );

USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_JOB_STOP, USB_NULL, 0, 0 );
```

Printing Text – POS Only

Before printing text, issue the following commands to describe the text:

- `USB_PRINTER_FONT_NAME` – Name of the font.
POS printers also specify the size of the font in the font name. Refer to the Help file for the list of available fonts.
- `USB_PRINTER_FONT_BOLD` or
`USB_PRINTER_FONT_MEDIUM` – Font weight.

The font specification will apply until the printer receives another font command.

Since POS printers primarily print one line of text at a time, use the command, `USB_PRINTER_POS_TEXT_LINE`, to print a single, null-terminated line of text and feed the specified number of lines after the text. For compatibility, the three-command sequence used for full sheet printing, described above, can also be used. Use the command, `USB_PRINTER_POS_FEED`, to feed a specified number of blank lines.

The text will be printed using the currently selected font, at the current printer position, with the currently selected justification (left, center or right).

EXAMPLE 3:

```
USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_JOB_START, USB_NULL, 0, 0 );

USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_POS_JUSTIFICATION_CENTER, USB_NULL, 0, 0 );
USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_FONT_NAME, USB_NULL, USB_PRINTER_FONT_POS_18x36, 0 );
USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_POS_TEXT_LINE, USB_DATA_POINTER_RAM("Hello world"), 1,
    USB_PRINTER_TRANSFER_COPY_DATA );
USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_POS_CUT_PARTIAL, USB_NULL, 100, 0 );

USBHostPrinterCommandWithReadyWait( &returnCode, printerInfo.deviceAddress,
    USB_PRINTER_JOB_STOP, USB_NULL, 0, 0 );
```

Printing Bit-Mapped Images – Full Sheet Only

For maximum compatibility with various printers, bit-mapped (raster) image printing requires four commands:

- `USB_PRINTER_IMAGE_START` – Initializes the printing of a bit-mapped image.
- `USB_PRINTER_IMAGE_DATA_HEADER` – Indicates a new row of bit map data.
- `USB_PRINTER_IMAGE_DATA` – Sends one row of the actual bit map data.
- `USB_PRINTER_IMAGE_STOP` – Terminates the image print.

The command, `USB_PRINTER_IMAGE_START`, requires a pointer to a structure of type, `USB_PRINTER_IMAGE_INFO`, containing information about the bit-mapped image, and where and how to print the image. This structure contains the position on the page to print the image, so the command, `USB_PRINTER_SET_POSITION`, is not required before printing an image. Printers can automatically resize bit-mapped images. Some printers utilize a resolution value (dots per inch) to specify the size of the printed image, and some use a scale factor. Some printers support only certain scale or resolution values, and some support a wide range of values. For maximum compatibility with various printers, specify both the `resolution` and the `scale` members of the printer information structure. Refer to Table 3 for example resolution and scale values to generate similarly sized output.

TABLE 3: COMPATIBLE RESOLUTION AND SCALE FACTORS

Resolution (DPI)	Scale
75	1.0
100	0.75
150	0.5
200	0.37
300	0.25
600	0.13

Bit map data is byte-based, with the Most Significant bit representing the left most pixel in the image row. A value of '0' indicates a black pixel and a value of '1' indicates a white pixel. The printer language will format the data as required for the particular language. The bit map data created by the font and bit map converter utility supplied with the Microchip Graphics Library is compatible with the required format.

Bit map data must be sent to the printer, one row at a time. Before each row, issue the command, `USB_PRINTER_IMAGE_DATA_HEADER`, then issue the `USB_PRINTER_IMAGE_DATA` command, with the `transferFlags` parameter set appropriately for the location of the bit-mapped data (RAM or ROM). After all rows of data have been sent, terminate the image print with the `USB_PRINTER_IMAGE_STOP` command.

The code in Example 4 illustrates how to send a complete image stored in ROM to a full sheet printer.

EXAMPLE 4:

```
WORD          currentRow;
USB_PRINTER_IMAGE_INFO imageInfo;
BYTE          returnCode;
WORD          widthBytes;

#if defined (__C30__)
    BYTE __prog__ *ptr;
    ptr = (BYTE __prog__ *)myImage.address;
#elif defined (__PIC32MX__)
    const BYTE *ptr;
    ptr = (const BYTE *)myImage.address;
#endif

// Extract the image height and width
imageInfo.width  = ((WORD)ptr[5] << 8) + ptr[4];
imageInfo.height = ((WORD)ptr[3] << 8) + ptr[2];

ptr += 10; // skip the header info

widthBytes = (imageInfo.width + 7) / 8;

USBHostPrinterCommandWithReadyWait( &returnCode,
    printerInfo.deviceAddress, USB_PRINTER_IMAGE_START,
    USB_DATA_POINTER_RAM(&imageInfo),
    sizeof(USB_PRINTER_IMAGE_INFO), 0 );

for (currentRow=0; currentRow<imageInfo.height; currentRow++)
{
    USBHostPrinterCommandWithReadyWait( &returnCode,
        printerInfo.deviceAddress, USB_PRINTER_IMAGE_DATA_HEADER,
        USB_NULL, imageInfo.width, 0 );

    USBHostPrinterCommandWithReadyWait( &returnCode,
        printerInfo.deviceAddress, USB_PRINTER_IMAGE_DATA,
        USB_DATA_POINTER_ROM(ptr), imageInfo.width,
        USB_PRINTER_TRANSFER_FROM_ROM );

    ptr += widthBytes;
}

USBHostPrinterCommandWithReadyWait( &returnCode,
    printerInfo.deviceAddress, USB_PRINTER_IMAGE_STOP,
    USB_NULL, 0, 0 );
```

Printing Bit-Mapped Images – POS Only

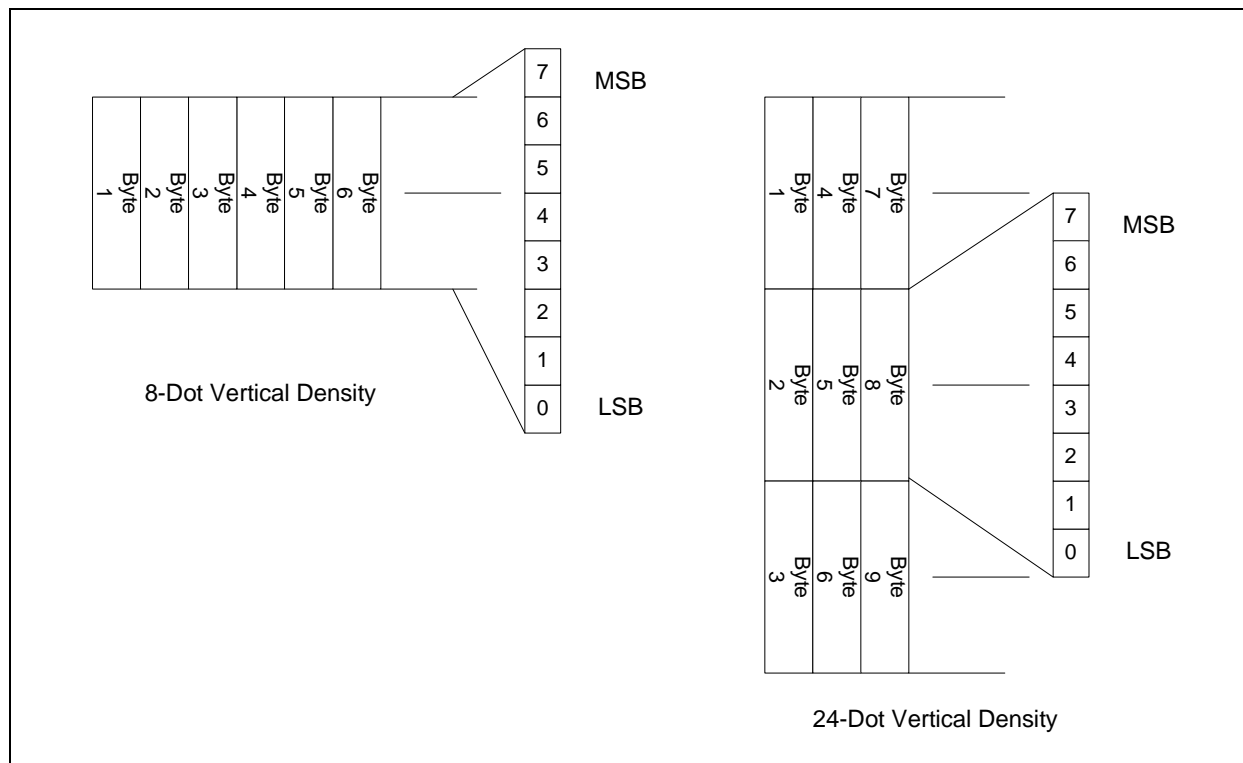
Image printing on POS printers is very similar to printing on full sheet printers. The major differences are the scale specification and the data format.

POS printers utilize the `densityVertical` and `densityHorizontal` members of the `USB_PRINTER_IMAGE_INFO` rather than the `scale` and `resolution` members. All POS printers that support image printing support 8-bit vertical density, and many also support 24-bit. Horizontal density can be either single or double. If it is supported, 24-bit vertical density

and double horizontal density is recommended, as it best maintains the original aspect ratio of the image. If 24-bit vertical density is not supported, then 8-bit vertical and single horizontal density best maintains the original aspect ratio of the image.

Image data must be sent to a POS printer in a specific format based on the vertical density. The image must be sent one row at a time, like the full sheet printers, but each row has a depth of 8 or 24 pixels (1 or 3 bytes), as opposed to a depth of one pixel for full sheet printers. This format is illustrated by Figure 8.

FIGURE 8: 8-DOT AND 24-DOT DATA FORMATS



Since this format differs from the way graphic images are stored, use the following function to format one row of data:

```
USB_DATA_POINTER USBHostPrinterPOSImage-
DataFormat( USB_DATA_POINTER image, BYTE
imageLocation, WORD imageHeight, WORD
imageWidth, WORD *currentRow, BYTE
byteDepth, BYTE *imageData )
```

This routine will take the data at the location specified by `image` and `imageLocation` with the specified `imageWidth`. It formats the data to the specified `byteDepth` (8-dot density = 1 byte, 24-dot density = 3 bytes) and stores it in the RAM location specified by `imageData`. Upon return, the current row of pixels (`currentRow`) is updated to reflect the next row of pixels and the function returns a pointer to the next byte of image data.

The code in Example 5 illustrates how to send a complete image stored in ROM to a POS printer.

EXAMPLE 5:

```
WORD          currentRow;
BYTE          depthBytes;
BYTE          *imageDataPOS;
USB_PRINTER_IMAGE_INFO  imageInfo;
BYTE          returnCode;
#if defined (__C30__)
    BYTE __prog__ *ptr;
    ptr = (BYTE __prog__ *)myImage.address;
#elif defined (__PIC32MX__)
    const BYTE *ptr;
    ptr = (const BYTE *)myImage.address;
#endif

imageInfo.densityVertical = 24;    // 24-dot density
imageInfo.densityHorizontal = 2;    // Double density

// Extract the image height and width
imageInfo.width = ((WORD)ptr[5] << 8) + ptr[4];
imageInfo.height = ((WORD)ptr[3] << 8) + ptr[2];

depthBytes = imageInfo.densityVertical / 8;
imageDataPOS = (BYTE *)malloc( imageInfo.width *
                                depthBytes );

if (imageDataPOS == NULL)
{
    // Error - not enough heap space
}

USBHostPrinterCommandWithReadyWait( &returnCode,
    printerInfo.deviceAddress, USB_PRINTER_IMAGE_START,
    USB_DATA_POINTER_RAM(&imageInfo),
    sizeof(USB_PRINTER_IMAGE_INFO),
    0 );

ptr += 10; // skip the header info

currentRow = 0;
while (currentRow < imageInfo.height)
{
    USBHostPrinterCommandWithReadyWait( &returnCode,
        printerInfo.deviceAddress,
        USB_PRINTER_IMAGE_DATA_HEADER, USB_NULL,
        imageInfo.width, 0 );

    ptr = USBHostPrinterPOSImageDataFormat(
        USB_DATA_POINTER_ROM(ptr),
        USB_PRINTER_TRANSFER_FROM_ROM, imageInfo.height,
        imageInfo.width, &currentRow, depthBytes,
        imageDataPOS ).pointerROM;

    USBHostPrinterCommandWithReadyWait( &returnCode,
        printerInfo.deviceAddress, USB_PRINTER_IMAGE_DATA,
        USB_DATA_POINTER_RAM(imageDataPOS), imageInfo.width,
        USB_PRINTER_TRANSFER_COPY_DATA);
}

free( imageDataPOS );

USBHostPrinterCommandWithReadyWait( &returnCode,
    printerInfo.deviceAddress, USB_PRINTER_IMAGE_STOP,
    USB_NULL, 0, 0 );
```

Printing Graphics – Full Sheet Only

Many printers support vector graphics, allowing applications to print large, complex images using a series of graphics commands rather than requiring large bit-mapped images. Various graphic objects that can be produced are:

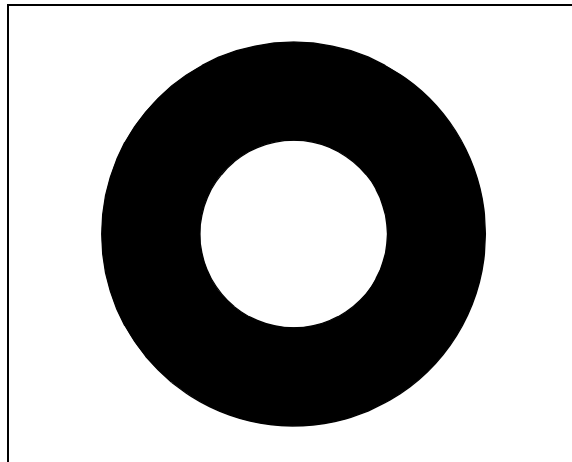
- Lines
- Arcs
- Circles (filled and outlined)
- Bevels or rounded rectangles (filled and outlined)
- Rectangles (filled and outlined)
- Polygons (outlined)

Before printing graphic items, issue the following commands to describe the line type used to create the item:

- `USB_PRINTER_GRAPHICS_LINE_TYPE` – Select a solid, dashed or dotted line.
- `USB_PRINTER_GRAPHICS_LINE_WIDTH` – Select a thin (1 point) or thick (3 point) line.
- `USB_PRINTER_GRAPHICS_LINE_END` – Select the style to use when terminating lines.
- `USB_PRINTER_GRAPHICS_LINE_JOIN` – Select the style to use when connecting lines.

Graphic objects can be printed in either black or white. Use the command, `USB_PRINTER_GRAPHICS_COLOR`, to select the desired color. Graphic objects are printed using opaque colors. For example, if a black circle with a radius of two units is printed at a specified center location, and subsequently, a white circle with a radius of one unit is printed with the same center location, the final output will appear as a ring or donut shape.

FIGURE 9: CONCENTRIC CIRCLES WITH ALTERNATE COLORS



Graphics objects are drawn with the currently specified line type and color. They are printed at the current printer position unless otherwise indicated by the command description. For command name consistency, the command, `USB_PRINTER_GRAPHICS_MOVE_TO`, is provided with the same functionality as the command, `USB_PRINTER_SET_POSITION`.

Graphic items typically require several parameters to specify them completely. The union, `USB_PRINTER_GRAPHICS_PARAMETERS`, provides structures that support the various graphics commands. Example 6 shows how to draw a circle.

EXAMPLE 6:

```
USB_PRINTER_GRAPHICS_PARAMETERS params;

params.sCircle.x    = 100;
params.sCircle.y    = 100;
params.sCircle.r    = 50;
USBHostPrinterCommandWithReadyWait(
    &returnCode,
    printerInfo.deviceAddress,
    USB_PRINTER_GRAPHICS_CIRCLE_FILLED,
    USB_DATA_POINTER_RAM(&params),
    sizeof(params.sCircle), 0 );
```

Printing Bar Codes – POS Only

Many POS have the built-in capability to easily print bar codes. Most printers that have bar code support can print the following types of bar codes:

- UPC-A
- UPC-E
- EAN 13
- EAN 8
- Code 39
- ITF
- CODEABAR

Some printers additionally support the following types of bar codes:

- Code 93
- Code 128
- EAN 128

To create a barcode, call the function, `USBHostPrinterCommand()`, or `USBHostPrinterCommandWithReadyWait()` with the command, `USB_PRINTER_POS_BARCODE`. The Data Pointer must point to a data structure of type `sBarCode` within the `USB_PRINTER_GRAPHICS_PARAMETERS` union. Example 7 shows how to print a simple barcode.

EXAMPLE 7:

```
char buffer[10];

strcpy( buffer, "Hello" );
params.sBarCode.height      = 75; // Height in dots.
params.sBarCode.type        = USB_PRINTER_POS_BARCODE_CODE39;
params.sBarCode.textPosition = BARCODE_TEXT_BELOW;
params.sBarCode.textFont    = BARCODE_TEXT_12x24;
params.sBarCode.data        = (BYTE *)buffer;
params.sBarCode.dataLength  = strlen(buffer);

USBHostPrinterCommandWithReadyWait( &returnCode,
    printerInfo.deviceAddress, USB_PRINTER_POS_BARCODE,
    USB_DATA_POINTER_RAM(&params.sBarCode),
    sizeof(params.sBarCode), 0 );
```

Getting Status Information

Two of the three Printer Class interfaces support receiving data from the printer. The availability and format of the data is printer dependent. If the application utilizes a specific target printer, it can use the following commands to request the information:

EXAMPLE 8:

```
if (!USBHostPrinterRxIsBusy( deviceAddress ))
{
    USBHostPrinterRead( deviceAddress, &buffer,
        sizeof(buffer), USB_PRINTER_TRANSFER_NOTIFY );
}
```

When the Printer Client Driver receives the requested data, it will generate the event, `EVENT_PRINTER_RX_DONE`. The application can then examine the received data, which has been placed in the specified buffer. To determine the amount of data received, use the command:

```
DWORD USBHostPrinterGetRxLength( BYTE
    deviceAddress );
```

DEMONSTRATION PROGRAMS

The USB Embedded Host Printer Client Driver is installed with the USB software support packages available for download at www.microchip.com/USB. After installation, the Printer Client Driver files will be located in the `.\Microchip\USB\Printer Host Driver` and `.\Microchip\Include\USB` directories, and three demonstration projects will be provided.

- `.\USB Host - Printer - Print Screen Demo`. This demo illustrates how to use the Graphics Library with the USB Printer Client Driver, capture a signature from a touch screen, and print it.
- `.\USB Host - Printer - Simple Full Sheet Demo`. This demo illustrates how to send output to a full sheet printer.
- `.\USB Host - Printer - Simple POS Demo`. This demo illustrates how to send output to a POS printer.

CONCLUSION

The USB Embedded Host Printing Device class provides a simple, standard interface to a printer. Although there is a wide variety of printer languages used by the multitude of available printers, some languages are considered standard and are supported by multiple manufacturers. Embedded applications can now easily take advantage of this, and provide hardcopy output for data presentation, snapshot reports, archival, receipts and labeling.

RESOURCES

- USB Embedded Host Library Help file, `.\Microchip\Help\`
- *AN1140, "USB Embedded Host Stack"*, <http://www.microchip.com>
- *AN1141, "USB Embedded Host Stack Programmer's Guide"*, <http://www.microchip.com>
- Universal Serial Bus web site: <http://www.usb.org>
- Microchip Technology Inc. web site: <http://www.microchip.com>

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

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. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, rfPIC, SmartShunt and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, In-Circuit Serial Programming, ICSP, ICEPIC, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, PICkit, PICDEM, PICDEM.net, PICtail, PIC³² logo, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rfLAB, Select Mode, Total Endurance, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2008, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2002 ==

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.



WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office

2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://support.microchip.com>
Web Address:
www.microchip.com

Atlanta

Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston

Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago

Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas

Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit

Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo

Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles

Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara

Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto

Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office

Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney

Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing

Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu

Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR

Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing

Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao

Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai

Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang

Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen

Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan

Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen

Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian

Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai

Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore

Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi

Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune

Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama

Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu

Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul

Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur

Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang

Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila

Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore

Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu

Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung

Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei

Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok

Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels

Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen

Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris

Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich

Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan

Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen

Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid

Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham

Tel: 44-118-921-5869
Fax: 44-118-921-5820