ENGINEERING UPDATE FOR 2.0 PRINTGEN

TABLE OF CONTENTS

1.0 PURPOSE and DEFINITION OF PRINTGEN

PrintGen is a tool kit which assists the programmer in creating a new Generic Print System (GPS) device driver. GPS is an abbreviation for the Generic Print System. PrintGen 2.0 addresses the issues of creating a GPS 2.0 device driver.

PrintGen is –

- Documentation relevant to the description, definition and creation of device-dependent GPS device drivers.
- Object modules for the device-independent portion of a GPS device driver.
- Various examples of device-dependent code for GPS device drivers.
- A collection of tools to simplify the creation of a new device driver for GPS.

PrintGen is NOT –

- An application or utility program which will create GPS device drivers.
- A tool for the "Printing Administrator".
- A menu-driven program to modify existing drivers or to create new GPS drivers.

PrintGen's goal is to enable an experienced CTOS system programmer to create a new GPS device driver within four weeks. Many GPS device drivers can be implemented in less time.

More technically stated, the object of PrintGen is to 1) assist the programmer in creating new device-dependent code, 2) provide all of the GPS-Core modules to the programmer, and 3) assist the programmer in combining the provided GPS-Core modules with the newly created device-dependent modules.

COMPATIBILITY NOTE

This version of PrintGen is an "engineering–update" version to be used with the Currently–released GPS 2.0. It entirely replaces the 1.1 version of PrintGen as well as the s1.01 "engineering_special" version. This "engineering–update" version provides the facility to build device drivers that are compatible with GPS 2.0 and that include improvements made in GPS 2.0 device drivers. Any drivers built by PrintGen for use with GPS 2.0 must be built with this version of PrintGen.

2.0 STRUCTURE and CONTENTS OF PRINTGEN

PrintGen consists of three basic components:

1) The PrintGen manual.

2) PrintGen files.

3) PrintGen examples.

PrintGen Manual –

This manual:

a. Defines what functions must be performed by the device-dependent portion of any GPS device driver.

b. Defines what procedural interfaces must be present in any GPS device driver.

c. Defines the GPS data-output procedures that must be used by the device-specific portion of any GPS device driver.

d. Describes and defines the various GPS data structures that are of concern to the device-dependent code in any GPS driver.

e. Comments upon the example device-driver source files that are included in PrintGen.

PrintGen Files –

The PrintGen files provide the device-driver programmer with:

a. The device-**independent** portion of any GPS device driver.

b. Data-structure definitions which may be included in the programmer's device-dependent code.

c. Commands and file-lists that control and simplify the linkage of a new GPS device driver.

PrintGen Examples –

Device–dependent source code for different types of GPS device drivers is included with the PrintGen product. These examples are taken from the set of Convergent Technologies supported GPS device drivers. They complement the definitions in the PrintGen Manual by showing the implementation of working GPS device drivers. Comments upon the driver examples are found in the PrintGen Manual.

Related Documentation –

Printing Guide.
Describes installation and use of GPS devices.

Generic Print System Programmer's Guide.
Describes the components and architecture of GPS.

The CTOS Operating System Manual, Volume I.
Describes Byte Streams, and Byte–Stream Image Modes.

3.0 CONTENTS OF PRINTGEN DISTRIBUTION DISKETTES

This release of PrintGen is contained on two diskettes. Most files are contained in an archive file on the two diskettes.

PrintGen–related files on the PrintGen–Installation diskettes --

[diskette #1]

<Sys>HdInstall.sub	PrintGen–installation file.
<CT>2.0PrintGen.doc	This document (Document Designer 2.0 format).

[diskette #1]<CT>.01 and [diskette #2]<CT>.02 (Archive File)

CtosTypes.h String.h String.mdf CtosLib.edf CtosTypes.edf	Ctos structure and subroutine definitions.
CommStatus_pre10.obj CommNub_pre10.obj InitComm_pre10.obj InitComm.obj	Byte-Stream routines included in GPS device drivers.
2.0DevDr.lib	Library of GPS-Core routines.
2.0Gpam.lib	Library containing GPAM routines. (Version 2.0)
GpsDb.Idf DD.lit structGps.h	Defines data structures used by a GPS device driver.
DdParams.asm DdParams.idf DdParams.h	Template for the device-description parameter file that must be provided with each GPS device driver.
DdSam.idf DdSam.h	Procedural definitions of the GPS-Core routines called by device-dependent code for data output.
GpsErc.idf GpsErc.h	Definitions of erc codes generated by GPS device drivers.

MinMax.idf Convert.idf CompileOptions.idf DdChangePWheel.idf DdUtilities.idf FontUtil.h FontUtil.idf DdVp.idf DRLTypes.h DDLits.h types.h ulos.idf Util.idf XUtilDD.idf DdFntLit.h DdVpData.h	Definitions of Utility routines used by the device-specific source code of the example device drivers.
LinkGpsDriver.sub	Definitions of Submit file which implements the <u>Link GPS Driver</u> command.
AltVer.run	Utility used when creating a GPS device driver that can be loaded with the PMOS server. Run file for <u>AltVer</u> command.
linkGpsFirst.fls	List of GPS-Core routines that must be the first object modules in the list of object modules specified to the Linker when linking a GPS device driver.
linkGpsGr.fls	List of GPS-Core graphics modules.
linkGpsGrStub.fls	List of GPS-Core modules that must be included when graphics-processing is not supported by a driver.
linkGpsReqRes.fls	List of GPS-Core modules that are required in all GPS device drivers.
linkDaisy.fls	List of device-specific object modules for the Daisy GPS device driver.

linkDaisy.libr.fls	List of device specific Library files, if any, for the Daisy GPS device driver.
linkDaisy.sub	Daisy submit file that links the Daisy driver.
DaisyParams.asm Daisy.c Daisy_Data.c DdConfig.c DdFontChangeG.plm DdSheet.c Concat2.c Concat3.c Daisy.h DdFDef.h Ulcmp.c Uc.c	Daisy-driver source code.
linkHPLaserJet.fls	List of device-specific object modules for the HPLaserJet GPS device driver.
linkHpLaserJet.libr.fls	List of device specific Library files, if any, for the HPLaserJet GPS device driver.
linkHPLaserJet.fls	HPLaserJet submit file that links the HPLaserJet device driver.
HpParams.asm HpLaserJet.plm	HPLaserJet-driver source files.
linkImagen8300.fls	List of device-specific object modules for the Imagen8300 GPS device driver.
linkImagen8300.libr.fls	List of device specific Library files, if any, for the Imagen8300 GPS device driver.
linkImagen8300.sub	Imagen8300 submit file that links the Imagen8300 device driver.
DdImagen8300.asm DdImagenDriver.plm DdImagenFont.c	Imagen-driver source files.

linkLptSimple.fls	List of device-specific object modules for the LptSimple GPS device driver.
linkLptsimple.libr.fls	List of device specific Library files, if any, for the LptSimple GPS device driver.
linkLptSimple.sub	Submit file that links the LptSimple Device Driver by means of the Link GPS Driver command.
LptSimple.c LptParams.asm	Simple-ASCII-printer driver source files.
Compile_Examples.sub	Submit file that compiles the source files of the example device drivers.
Compile_Dev.sub	The Submit file that is called for each of the four example device drivers.
PrtGen.fls	List of all the files in the Archive file in the CT directories of the PrintGen installation diskettes.

4.0 PRINTGEN INSTALLATION

PrintGen may be installed onto a workstation which has been booted from a hard-disk -- it cannot be installed upon a system that boots from a floppy disk. CTOS Standard Software 10.3, or later is required for PrintGen installation and use.

The installation procedure:

a) Restores the PrintGen files from the distribution diskettes archive file to various directories on the currently pathed device. The directories will be created if they do not already exist. Note that the user must assure that existing directories are of the size indicated below or the compilation and link of example drivers may fail. These directories are:

<2.0Daisy>	200 Files
<2.0DaisyBld>	75 Files (default)
<2.0HP>	200 Files
<2.0HPBld>	75 Files (default)
<2.0Imagen>	200 Files
<2.0ImagenBld>	75 Files (default)
<2.0Lpt>	200 Files
<2.0LptBld>	75 Files (default)
<2.0DevDrBld>	75 Files (default)
<2.0Gpam>	75 Files (default)
<2.0GpamBld>	75 Files (default)
<2.0GpsDef>	200 Files
<2.0PrtGenBld>	75 Files (default)

b) Copies command files to the "system" directory. This is usually the **[Sys]<Sys>** directory. Another "system" directory may be specified -- see Installation-Procedure step 3 for the description of that installation parameter.

c) Adds PrintGen commands to a command file. The command file to which commands are added is usually **[Sys]<Sys>Sys.cmds.** Another command file may be specified -- see Installation-Procedure step 3 for the description of that installation parameter.

PrintGen is installed onto a hard-disk workstation by performing the following procedure:

1. Place the PrintGen-distribution diskette into floppy drive **[f0]**.

2. Use the PATH command to set the path to the volume where PrintGen directories will reside. Set default password to the **volume password** for **the PrintGen volume**. If any installation parameters other than the defaults are used, other volume passwords may need to be set, or perhaps none at all.

3. Use the SUBMIT command to submit **[f0]<Sys>HdInstall.-sub.** The INSTALL command will work here; however, there is no way to set installation parameters if the install is done via the INSTALL command.

   No parameters need be set, but there are two installation parameters that may be specified by the installer.

   Installation parameters --

   #1) **Installation Floppy Drive -** This is the device from which the PrintGen files are restored. It defaults to **[f0]**. A different drive can be specified with this SUBMIT parameter.

   #2) **Command File** This is the command file to which the PrintGen commands are added. It defaults to **[Sys]<Sys>Sys.cmds.** A different command file (e.g., [Sys]<Sys>Special.cmds) can be specified with the third SUBMIT parameter.

   Any of these parameters must be on the SUBMIT command's "Parameters" line. Position is important. If a value is to be entered on the second parameter, but not for the first, a null parameter (i.e., '') must precede the second-parameter string on the "Parameters" line.

4. The Restore command will prompt the user to mount/insert **[f0]<ct>.01**. This is the first of the two distribution diskettes, and is already inserted in the drive -- the user need only press **<GO>** at this point. When the Restore command prompts the user for **[f0]<ct>.02**, he should remove the first diskette, insert the second diskette and press **<GO>**.

5. After the "installation-completed" message is displayed, the PrintGen-distribution diskette is removed from **[f0]**.

6. On previous releases, the user could elect to copy the Example files from a separate diskette. In this release, all files are restored, including the example drivers and their source.

7. All of the example GPS device drivers can be compiled by submitting the following submit file: **Compile_Examples.sub**. This file invokes another submit file, **Compile_Dev.sub**, to compile each of the example drivers. Examination of **Compile_Dev.sub** will reveal the parameters required to compile and link the desired driver.

5.0 ADDITIONS and CHANGES

Fonts and character–translation:

Among the major new features introduced in GPS 2.0 are: 1) the support of various fonts within a document; and 2) character translation. These new device–driver features are described in the following sections and subsections of this document:

- DEVICE–DRIVER FONT PROCESSING (page 18)
- Font–Related Utilities (page 53)

Page orientations and sizes:

GPS 2.0 has added a number of additional features to GPS in the area of **page orientations** and **page dimensions**. Landscape and portrait pages can now be printed in the same document. Device page sizes can be specified at device–driver installation.

See the PAGE ORIENTATIONS and DIMENSIONS section (page 21) for more information about page orientations and sizes.

Device–driver initialization:

Facilities have been added for device–dependent code that must perform initilizations and resource allocations during installation, before the device driver is converted to a CTOS system service. See the description of the **DdBeforeConvertG** routine (page 34) in the Required Device–Dependent Procedures subsection.

Debugging aids:

A number of features have been added to the GPS 2.0 device drivers that aid the debugging of device-dependent code. These are new features which have either been added explicitly for debugging, or which also aid debugging in addition to their primary purpose.

Debugging the device-dependent code of a GPS device driver is aided by the following features:
- Debugger entry at device-driver installation.
- **DdBeforeConvertG** and **DdFirstChanceG** DDP routines.
- File output of output-device commands.

See the DEBUGGING AIDS section (page 104) of this document for detailed information about these debugging aids.

GPS 1.0 → GPS 2.0 compatibility and conversion:

GPS 1.0 → GPS 2.0 compatibility issues are discussed in detail at the appropriate places throughout this document. In addition, the GPS 1.0 TO GPS 2.0 CONVERSION section (page 94) is devoted to GPS 1.0 → GPS 2.0 conversion. That section discusses several different conversion approaches. The different approaches require different amounts of conversion effort and yield different levels of GPS 2.0 functionality.

6.0 STRUCTURE OF A GPS DEVICE DRIVER

This section of the PrintGen Manual provides insight into the architecture of a GPS device driver. This overview will aid the programmer in understanding how the device-dependent portion of a GPS device driver fits into the rest of the driver. It will also aid the programmer in understanding which functions the device-dependent code must perform and which functions are already implemented in the device-independent portion of a GPS device driver.

Two basic components comprise a GPS device driver. Those components are:
- GPS device-driver Core.
- GPS device-dependent portion (DDP).

The Core consists of all code that is used by all GPS device drivers, independent of the actual output device being driven. All code in any driver that is device-dependent comprises the DDP code.

The object of PrintGen is to: 1) assist the programmer in creating new DDP code, 2) provide all of the GPS-Core modules to the programmer, and 3) assist the programmer in combining the provided GPS-Core modules with the newly created device-dependent modules.

One of the design goals of GPS, was to greatly simplify the task of implementing new drivers. Consequently, the majority of a GPS device driver's code is in the GPS Core -- relatively little DDP code need be created to implement a new GPS device driver.

The DDP of a GPS device driver translates GPS data into device-specific data. It converts locations in GPS units of measure to the appropriate device-specific locations, in the device's units of measure. It converts GPS character-attribute flags into device-specific command sequences. It converts GPS color values into the appropriate, device-dependent command sequences, etc.

From the perspective of the DDP, the GPS Core may be thought of as performing two functions. First, it provides character and graphics information to the DDP for translation into a device-specific format; and, secondly, it takes the device-specific data from the DDP and outputs it to that physical device.

Therefore, a GPS device driver is architecturally composed of two parts: the GPS Core; and the DDP. But, using a data-flow model, the DDP is sandwiched in the middle of the GPS Core -- it receives GPS data from the Core, translates it into device-specific data, and sends that new data to other half of the GPS Core for output.

You can view the complete GPS device driver in terms of the "Jam Sandwich" model. The bread is the GPS Core, and the jam is the DDP. The "jam" is surrounded by the "bread" -- one slice of the GPS Core sends (GPS) data to the DDP while the other slice of the GP1S Core receives (device-specific) data from the DDP of the GPS device driver.

The text data received by the DDP from the GPS Core consists of GPS Character Records. One of these records is received for each character to be printed. A GPS Character Record specifies which character is to be printed, and where it is to be printed on the page. This record is fully defined in the <u>GPS CHARACTER-RECORD</u> section (page 83).

The graphics data received by the DDP from the GPS Core define various graphics objects, in terms of GPS units. The graphics-data structure is defined for each graphics-processing routine in the <u>DEVICE-DEPENDENT FUNCTIONS</u> section (page 33).

Output data emitted by the DDP of the GPS device driver is a stream of bytes, in whatever format is proper for the particular device supported by the driver.

The DDP does not always translate the output data. If TRANSPARENT, IMAGE or BINARY image modes are in effect, all of the data is sent directly from the input slice of the GPS Core to the output slice, completely bypassing the DDP of the driver. (These "image modes" are defined in the GPAM and CTOS-bytestreams documentation.)

Some of the GPS Core code is used only in the processing of graphics data. A GPS device driver which supports only textual data need not be burdened with graphics-processing code which it will never use. Exclusion of the Core's graphics code may be requested when the device-driver runfile is being linked. The details of this are described in the LINKING A NEW GPS DRIVER section (page 100).

7.0 DEVICE-DRIVER FONT PROCESSING

Among the major new features introduced in GPS 2.0 are:

1) The support of various fonts within a document.
2) Character translation.

This section discusses what a GPS 2.0 device driver must accomplish to support font translation, what information a device driver obtains from the Convergent Font Database, and what font-translation actions are the responsibility of the device-dependent code in any GPS 2.0 device driver. This section describes the "font philosophy" of a GPS 2.0 device driver. Subsequent sections detail the procedures providing font-translation services to device-dependent code. This supplements the Font Database and Font Service information provided in the Printing Guide.

Character translation is required for the use of various fonts within a document, but is also a useful feature even with devices that can output only one font. For example, the output character code used to print a ä, varies from device to device. The character translation feature in the GPS 2.0 device driver can be used to translate the document's character code for ä into the character code required by the particular device to print the ä character. This section discusses the character translation feature in the more general case of multiple-font translations, but is relevant for those GPS 2.0 device drivers that support only one font and require some character-code translations.

The following definitions are used in the rest of this section and in some subsequent sections.

A **glyph** is mark made by an output device. Often a printed character and a glyph are the same thing, but not always. Ä is an output character. Depending upon the output device's capabilities, this character could be composed of either one or two glyphs. Some printers have a command that results in an ä glyph being printed, but on some other printers the ä is printed by commanding the printer to output an a glyph and an umlaut glyph at the same location.

Let α	represent the character code for a character in a document.
Let β	represent the code for a glyph that can be printed by a device.
Let χ	represent the command string which must be sent to a device to cause a particular glyph to be printed.

The goal of a GPS 2.0 device driver, with respect to font translation, is to perform the following translations:

$$\alpha \rightarrow \beta \rightarrow \chi$$

For a given α, find its corresponding β, and then find the χ which corresponds to that β. Or to put it another way, for a given character code in a document (α), find the value (β) used to represent that glyph on the particular output device, and then find the device command (χ) that must be sent to the device to cause the desired glyph to be printed.

For example, assume the following:

1. The character code for ¢ in a document is 3.
2. A particular printer is capable of printing the ¢ glyph.
3. The code assigned to the ¢ glyph for this printer is 250.
4. The command that must be sent to this printer to print the ¢ glyph is SHIFT-IN/123/SHIFT-OUT.

So α is 3.
β is 250.
χ is SHIFT-IN/123/SHIFT-OUT.
And the desired translation to print a ¢ upon this particular printer is:

$3 \rightarrow 250 \rightarrow$ SHIFT-IN/123/SHIFT-OUT.

The translation sequence:

$$\alpha \rightarrow \beta \rightarrow \chi$$

is more manageable when separated into its two components:

1. $\alpha \rightarrow \beta$
2. $\beta \rightarrow \chi$

The first translation ($\alpha \rightarrow \beta$) is referred to as the *First-Level Translation*. The second ($\beta \rightarrow \chi$) is referred to as the *Second-Level Translation*.

In general, the First-Level Translation is actually not a 1-to-1 translation, but rather a 1-to-many translation. Consider the above case of a printer that prints an ä by outputting two glyphs (an a and an umlaut) at the same position. In this case the First-Level Translation must translate from the document code to the printer's glyph codes. (And then the printer commands to print each of these two glyphs are obtained by performing the Second-Level Translation upon each of these two resultant glyph codes.) So the First-Level Translation is really:

$$\alpha \rightarrow (\beta_1,\beta_2,\beta_3 \ldots,\beta_n).$$

The Font Service supports this 1-to-many mapping in the First-Level Translation.

A GPS 2.0 device driver is organized such that the driver Core controls the First-Level Translation, and the device-dependent code controls the Second-Level Translation. (Utility routines are provided for use by the device-dependent code to extract the Second-Level Translation information from the Font Database.) This division between the Core and device-dependent code means that the device-dependent code need not compose a character from individual glyphs, and need not translate from the document character code to the device's glyph codes. The Core passes to the device-dependent code a series of device-glyph codes and the location at which each glyph is to be placed upon the page. The device-dependent code is then responsible for correctly positioning the "printhead" and for obtaining and outputting the corresponding print-glyph command. Consider again the above case of a printer that prints an ä by outputting two glyphs (an a and an umlaut) at the same position. The Core will translate the document code for ä into the device's glyph code for an a and the device's glyph code for an umlaut. These two glyph codes will be passed to the device-dependent code, which is responsible for obtaining and outputting the two corresponding printer commands.

Design of a particular GPS 2.0 device driver and the information for that device in the Font Database are, therefore, interrelated. Generally, the device-driver designer should also "design" the data to be added to the Font Database for that device.

Subsequent sections in this document detail:

1. Routines called from device-dependent code to obtain Second-Level Translation data.
2. Font-related information passed from the Core to the device-dependent code for each printer glyph to be printed.
3. Examples of how some of the Convergent-supplied device drivers use these routines.

8.0 PAGE ORIENTATIONS and DIMENSIONS

GPS 2.0 has added a number of additional features to GPS in the area of **page orientations** and **page dimensions**. Landscape and portrait pages can now be printed in the same document. Device page sizes can be specified at device–driver installation.

To take full advantage of these new features, device–dependent code and data must added for each hardware device that can support them.

Page orientation and page dimensions are presented together in this section because page orientation is determined by page dimensions.

8.1 Page Dimensions

Several page–dimension sets exist in a GPS 2.0 device driver. They are:

1) **PrintGen** page–dimensions.
2) **Installation** page–dimensions.
3) **GPAM** page–dimensions.
4) **Current** page–dimensions.

These four page–dimensions are described in detail below.

PrintGen page–dimensions:

Each GPS 2.0 device driver includes a specification of the maximum–sized page that can be supported by the device driver. This page–dimension pair is specified in **DDwPgLength** and **DDwPgWidth**. These two PrintGen variables are defined in the GPS DEVICE–SPECIFICATION PARAMETERS section (page 87).

These two variables had a different usage in GPS 1.0 device drivers. In GPS 1.0 device drivers, they specified the driver's page size for each page of each document. In GPS 2.0 device drivers, these variables specify the driver's maximum page size. The three other page–dimensions cannot exceed this driver–maximum page size.

Installation page–dimensions:

Device page size is specified at device–driver installation. This **installation** page–dimension becomes:

a. The **default** page size for all documents printed by this driver.
b. The **maximum** page size for all documents printed by this driver.

This installation–time, page–size specification allows a single device driver to support different–sized paper in different installation instances.

Installation page–size is limited by the **PrintGen** page–dimensions. An installation page–size that is larger than the PrintGen–specified page–size is not accepted. During installation, each **installation** page–dimension that exceeds the corresponding **PrintGen** page–dimension is replaced by the corresponding **PrintGen** page–dimension. For example, if the **PrintGen** page–size is 8½ X 11 and a page–size of 14 X 10 is specified at device–driver installation, then the resultant installation, default page–size will be 8½ X 10.

Installation page–dimensions are specified by entering values into the Length and Width fields of the Page Dimensions for GPS Output section of Print Manager's device–driver installation form, and by entering a value into the Chars Per Line (non–GPAM output) field of the same Print Manager form. The "Chars Per Line" value is a special–case, page–width specification, and is discussed later in this subsection.

GPAM page–dimensions:

GPAM's **GPAMBeginPage** call may include page–dimension specifications. This page–dimension information specifies the page–dimensions to be used while printing the page data that follows the **GPAMBeginPage** call. Any **GPAM** page–dimensions replace the **Installation** page–dimensions for the page that follows.

GPAM page–dimensions are limited by the **Installation** page–dimensions. A **GPAM** page–size that is larger than the installation–specified page–size is not accepted. During **GPAMBeginPage**–processing, each **GPAM** page–dimension that exceeds the corresponding **Installation** page–dimension is replaced by the corresponding **Installation** page–dimension.

Current page–dimensions:

The page–dimensions currently in effect are the **Current** page–dimensions. These are the page–dimensions actually used by the GPS Core in processing print data. **Current** page–dimension values are a function of the preceding three types of GPS device–driver page–dimensions.

Usually, the **Current** page–dimensions are equal to either the **Installation** page–dimensions or to the **GPAM** page–dimensions. When printing a GPAM document, the **Current** page–dimensions are usually equal to the **GPAM** page–dimensions. When printing a non–GPAM document, the **Current** page–dimensions are usually equal to the **Installation** page–dimensions. But they are sometimes not equal to either the **GPAM** or **Installation** page–dimensions. As described above, **Installation** page–dimensions can be altered by **PrintGen** page–dimension limitations, and **GPAM** page–dimensions can be altered by **Installation** page–dimension limitations. Also, the actual **Current** page–dimensions are affected by the page orientation requested. This is described in the following Page Orientations subsection (page 27).

When device–driver installation begins, the **PrintGen** page–dimensions are used as the **Current** page–dimensions. After the **Installation** page–dimensions have been obtained (and perhaps altered by **PrintGen** page–dimension limitations), they become the new **Current** page–dimensions. When any **GPAM** page–dimensions are encountered, their (perhaps altered) values become the new **Current** page–dimension values. If a page–orientation request necessitates a change in page–dimensions, then the new, changed values become the new **Current** page–dimension values.

Page–dimension usage:

Page–dimension values have two primary uses in a GPS device driver:

1. Page–orientation determination.
2. Device–dependent information.

Page–orientation determination is discussed in the Page Orientations subsection (page 27), which follows. Page–dimension information is passed to the device–dependent code at the beginning of each page. (See the description of **DdNewPageG** in the DEVICE–DEPENDENT FUNCTIONS section, page 40.)

Generally, the page–dimensions do not cause the Core to restrict where characters or graphics may be placed on a page. A **GPAMReposition** may be specified to any location within the 45″ X 45″ GPS page space, regardless of the page–dimensions. And characters and graphic objects may be placed at any location within the GPS page space. There are some special cases in which the GPS Core does force placement of characters to be within the **Current** page–dimensions. These involve the printing of non–GPAM and unformatted documents, and are discussed below.

Chars–Per–Line and non–GPAM documents:

When a non–GPAM document is being printed, the page width is based upon number of columns instead of the specified page width. This number–of–columns value is obtained from the Chars Per Line (non–GPAM output) field of the Print Manager's device–installation form.

When a non–GPAM document is being printed, the page width is based upon how many columns are available. When a GPAM document is being printed, the page width is based upon the specified size in inches.

A GPS 2.0 implementation detail is significant here. The GPS Core's **Current** page–dimensions are always inches, never columns, even when printing a non–GPAM document. When the printing of a non–GPAM document is begun, the specified Chars Per Line value is multiplied by the device driver's default character width (**DDwWidth**) to generate a page width in inches that the Core can use for part of its page–dimensions. If a device driver has a particularly large default character width and a large enough "columns" value is specified, then the Core will attempt to generate a page width that exceeds the 16–bit limit of GPS units. If such overflow occurs, the resultant output is undefined. For example, if a device driver's default character width is one–tenth inch (144 GPS units), then the maximum valid value of Chars Per Line is about 445.

Line wrapping and automatic FormFeeds:

In a limited number of cases, the GPS Core will prevent an attempt to place a character past the right edge of a page or beyond the bottom of a page. These cases occur only in unformatted documents.

An unformatted document is a document that does not contain character–positioning commands. An ASCII text file is therefore an unformatted document (LineFeed, CarriageReturn, Tab and FormFeed are not character–positioning commands for this classification). A GPAM file may or may not be a formatted document. If it contains no character–positioning commands, then it is an unformatted document.

When any attempt is made to place a character of an unformatted document past the bottom of the page, a new page is begun, and that character is positioned at the beginning of this new page. The offpage detection and new-page generation is performed by the GPS Core. The page-length component of the **Current** page-dimensions determines the location of the bottom edge of the page.

Attempts to place an unformatted-document character past the right edge of the page may or may not be allowed. The default is to allow such character placement. However, if **Wrap:** was entered on the Device Setup field of Print Manager's device-driver installation form, then such past-the-right-edge placement is not allowed by the Core. Rather the offending character is placed at the beginning of the next line. The page-width component of the **Current** page-dimensions determines the location of the right edge of the page.

So for an ASCII text file that contains no FormFeeds, a new page will be automatically created by the Core each time the current page is filled. If **Wrap:** is not specified at installation and any lines of the text are wider than the printer, the appearance of the output depends only upon what the particular printer does when an output line exceeds its capacity. If **Wrap:** is specified, then these too-long text lines will be automatically "wrapped" onto the next printer line by the GPS Core.

GPS 1.0 device drivers perform quite differently in this respect. They force line wrapping and automatic page eject for all documents -- even formatted documents. By entering **Wrap**: on the Device Setup field of Print Manager's device-driver installation form, a GPS 2.0 device driver can be made to behave like a GPS 1.0 device driver, when processing unformatted documents.

Page borders:

The Page Dimensions for GPS Output section of Print Manager's device-driver installation form includes *Border* fields. Border values are for special cases of printing hardware. Border values are often required for laser-printer drivers. Even for laser printers, these fields can usually be left blank.

Many of the laser printers are unable to print on all of their page's character positions. Although these printers allow the commands to print on these positions, they are unable to actually make any marks on their page's at those locations. These *dead zones* are along one or more of the page's four edges. The size of each edge's dead zone usually differs from the sizes of that page's other dead zones. Their values vary among the different brands of laser printers, and even vary among a given brand.

It is not unusual, for example, for the first one to three columns of a laser printer to be unusable.

Default border values must be supplied as part of each GPS device driver. If no border values are entered at device-driver installation, then all of the driver's default border values are used. For printers that do not have this *dead-zone* problem (non-laser printers), the default border values should be 0. For laser printers, nominal border values for the supported printers should be included in the GPS device driver. If the proper default border values are built into the device driver, the user will rarely need to enter any border values when installing the driver. Some experimentation with several copies of the given printer is usually required to obtain these values.

A GPS device driver's default border values are stored in the following device-driver variables:

● **DDwLeftBorder**	Width of the page's left-edge dead zone (GPS units).
● **DDwRightBorder**	Width of the page's right-edge dead zone (GPS units).
● **DDwTopBorder**	Height of the page's top-edge dead zone (GPS units).
● **DDwBottomBorder**	Height of the page's bottom-edge dead zone (GPS units).

These device-driver variables are listed in the GPS DEVICE-SPECIFICATION PARAMETERS section (page 87).

The border values affect the Core's **Current**-page-dimensions calculations. They affect the page's default first-character position, where auto-line-wrap occurs, where auto-page-eject occurs, auto-sizing of graphics, auto-centering of graphics, etc.

These printer border values should not be confused with formatted-documents' margins. **Values should never be entered into the border fields at device-driver installation time in an attempt to create document margins!**

8.2 Page Orientations

Two page orientations are supported by the GPS 2.0 Core. Page–orientation information is passed from the GPS Core to the device–dependent code at the beginning of each page.

A given printer may or may not produce output in the two orientations. Whether or not it does is determined by:

1) Whether or not the printer hardware can print in two orientations.
2) Whether or not page–orientation support has been included in the GPS device driver's device–dependent code.

Page–orientation definitions:

The two page orientations are named:

- Portrait
- Landscape

When a page's height (or length) is greater than its width, then that page's orientation is **Portrait.** (The painting of a portrait is usually taller than it is wide.)

When a page's width is greater than the page's height (or length), then that page's orientation is **Landscape.** (The painting of a landscape is usually wider than it is tall.)

Desired page orientation may be specified by two means in a GPS 2.0 device driver.

First, a **Portrait** or **Landscape** orientation may be requested in the **GPAMBeginPage** call. The **bAspect** field of the parameters structure may be set to **Landscape** or **Portrait**.

Secondly, the **Current** page dimensions may be used to specify the page orientation. If **GPAMBeginPage**'s **bAspect** field is set to **default**, then the **Current** page dimensions are evaluated by the GPS Core. If the **Current** page dimensions have a height greater than their width, **Portrait** orientation will be requested. If the width is greater, **Landscape** orientation will be requested. The **Current** page dimensions are also used to determine page orientation when a document contains no **GPAMBeginPage** specifications.

A **bAspect** value of **Landscape** or **Portrait** takes precedence over **Current** page dimensions and **GPAMBeginPage** page dimensions.

GPAM applications may use either means of specifying page orientation. The Convergent Document Designer, for example, does not utilize the **bAspect** field at all. Rather every page of a

Document Designer document has specific dimensions associated with it, and these dimensions are included in the **GPAMBeginPage** calls. So at the beginning of each page, the **GPAMBeginPage** page dimensions are used to specify the desired orientation for that page.

A GPAM document may contain pages of both orientations. This is true because at the beginning of each GPAM page, a new page orientation may be requested. This is accomplished through use of either the **GPAMBeginPage bAspect** value or the **GPAMBeginPage** page dimensions.

Only one page orientation will exist in a non-GPAM document. Since a non-GPAM document does not include **GPAMBeginPage** calls, it cannot specify page orientation or page-dimensions. Therefore the **Installation** page-dimensions will be used to select page orientation, and the page orientation will not change within the document.

For the same reasons, only one page orientation will exist in GPAM documents that do not include **GPAMBeginPage** calls.

Such default page orientations may differ for non-GPAM and GPAM documents, however. This is because the <u>Chars Per Line</u> value specifies the installation page width for non-GPAM documents, while <u>Width</u> (in inches) specifies it for GPAM documents. The two are generally not the same, and may result in different page orientations being requested. For example, assume that a driver is installed with the following page dimensions: 1) Width = 8½", Length = 11" and Chars-Per-Line = 132. Assume also that the device driver's default character width is 0.1 inch. Then the default **Current** page-dimensions will differ for the two types of documents. For a GPAM document the dimensions are 8½ X 11, which specifies portrait orientation. For a non-GPAM document the dimensions are 13.2 X 11, which specifies landscape orientation (132 * 0.1" = 13.2").

DDP's page-orientation responsibilities:

A GPS device driver's device-dependent code must:

- Specify to the GPS Core whether or not this device driver supports more than one page orientation.
- Access the page-orientation information provided by the Core.
- Issue to the printer the necessary change-page-orientation commands.

DdDevOrient values specify the device driver's page-orientation capabilities.

Page-orientation information is passed to the DDP at the beginning of each page.

The following PageL and PageP subsection describes the one case in which the device-dependent code is not responsible for generating the printer's change-page-orientation commands.

DdDevOrient values:

A data structure in each GPS device driver specifies the page-orientation capabilities of that device driver. The data structure is **DdDevOrient**. See the GPS DEVICE-SPECIFICATION PARAMETERS section (page 92) for the **DdDevOrient**-structure definition and field values.

Multiple page-orientation processing will be done by the Core only if the **bOrients** field specifies that the device is capable of outputting in two orientations. If this field is set to one, the Core will never notify the DDP of requests for page-orientation changes.

The device's default page orientation is specified by the value set in the **bDefault** field. If the driver supports only one page orientation (**bOrients = L1ORIENTATION**), the value placed into the **bDefault** field is irrelevant.

Laser printers' page borders (described in the previous subsection) necessitate the **bPtoL** field. The Core must calculate the first and last "columns" that will actually show up on a page. It must also calculate the first and last usable lines. These values are determined by page-border values. When page orientation changes, these values are recalculated. If page orientation changes from its default orientation, is the new Left Border now the old Top Border or the old Bottom Border? It varies from printer to printer. The value that is set into this **bPtoL** field specifies whether it is the old Top Border or the old Bottom Border. The other, needed border-rotation rules are also specified by this field's value.

A great variety of orientation-change possibilities exist among the various devices that support multiple page orientations. Therefore the following should be done to determine the correct value for the **bPtoL** field:

1. Set **bPtoL** to one of its two valid values. Valid **bPtoL** values are 1 and 2. Symbol values defined in **DdParams.asm** are **LMINUS90** and **LPLUS90.**
2. Print a non-GPAM document in the non-default page orientation. (This document should **not** begin with spaces.)
3. If any of the first characters of each line do not show up on the output page, then set **bPtoL** to its other valid value.

If the driver supports only one page orientation (**bOrients** = **L1ORIENTATION**), the value placed into the **bPtoL** field is irrelevant. If the driver supports a multi-orientation printer that does not have dead-zones on its page's edges, set **bPtoL** to either of the two valid values.

The **fLFntDev** must always be set to **FALSE** in a GPS 2.0 device driver.

Page-orientation information passed to DDP:

Each time that a page is begun, the GPS Core calls the DDP's **DdNewPageG** routine. This call includes a pointer to the new page's **Page Descriptor**. The **Page Descriptor** is a structure containing information about the new page's dimensions and orientation. It is defined in the GPS DATA STRUCTURES OF INTEREST section (page 78). The **bOrient** field of the structure specifies the page's orientation.

It is the responsibility of the device-dependent code to interpret the **bOrient** field, and issue the needed orientation-change commands to the output device. Any font-selection operations that are a function of page orientation must be handled by the device-dependent code. Printing documents using both portrait and landscape orientations is supported only to the extent that the same printer fonts are available in both orientations.

PageL and PageP:

The GPS Core does provide an orientation-change-command output facility. It is called **PageL/PageP**. Its intended use is for temporary, page-orientation support of new printers. The **PageL/PageP** feature should be no longer needed after the driver has been updated to support that new printer.

PageL/PageP is a GPS-Core feature that outputs change-page-orientation commands to the device. The commands that is sends to the printer are specified by the user at device-driver installation. **PageL/PageP** is relevant only if the DDP specifies that it supports both page orientations.

Two different page-orientation command strings may be entered into Print Manager's device-driver installation form. One is the command which causes the device to switch to landscape orientation -- this string is called the **PageL** string. The other is the command which causes the device to switch to portrait orientation -- it is called the **PageP** string.

A **PageL** string is specified by typing **PageL:<hex-string>** into the DEVICE SETUP field of Print Manager's device-installation form. A **PageP** string is specified by typing **PageP:<hex-string>** into that field. *<hex-string>* is the command string that is to be sent to the output device.

The **PageL** string is output to the device, by the GPS Core, after calling **DdNewPageG** if all of the following are true:

1. A **PageL** string was specified at device-driver installation.
2. The new page's orientation is landscape.
3. Both page orientations are supported by the device driver.

The **PageP** string is output to the device, by the GPS Core, after calling **DdNewPageG** if all of the following are true:

1. A **PageP** string was specified at device-driver installation.
2. The new page's orientation is portrait.
3. Both page orientations are supported by the device driver.

It is not recommended that the **PageL/PageP** facility be considered as a permanent solution to any device-support requirement. A better approach is to have a list of device names that are recognized by the device-dependent code. The user enters one of these names into the DEVICE SETUP field of Print Manager's device-installation form when installing the device driver. The DDP uses this information to select the device commands that it should, itself, output when **DdNewPageG** is called. This approach is less error prone, is easier for the user, and enables the device-dependent code to select the appropriate set of device-dependent commands.

Another example of the use of **PageL** and **PageP** strings is in the Convergent-supplied Daisy device driver. The goal was to make it likely that many of the laser "daisy emulators" would work with Convergent's driver, and that the page-orientation capabilities of these daisy emulators could be used. This was accomplished by specifying that the Daisy driver supports both page orientations (**DdDevOrient.bOrients = L2ORIENTATIONS**), and by utilizing the **PageL/PageP** facility.

If a user specifies **PageL** and **PageP** strings when installing the Convergent-supplied Daisy device driver, the appropriate command string will be output (by the Core) to the attached printer each time that page orientation changes.

If such a laser printer was to be used regularly with the Daisy driver, PrintGen should be used to create a new version of the device driver. This new version would recognize a new device name that represents this laser daisy emulator (just as it currently recognizes names that represent the Diablo630, Qume, etc.) Occurrence of this new device name would signal the device-dependent code to output the required change-orientation commands when its **DdNewPageG** is called by the Core.

9.0 DEVICE-DEPENDENT FUNCTIONS

This section lists the functions that must be provided by the DDP's device-dependent code. These procedures are called by the GPS device-driver Core to pass characters to the DDP, pass graphics objects to the DDP, begin and end documents, begin and end pages, etc.

Not all of these functions must always be implemented. The procedure specifications in this section are divided into two groups:

- ALWAYS REQUIRED,
- REQUIRED ONLY FOR **GRAPHICS PROCESSING.**

Each procedure's specification consists of:

1) **A procedure declaration.**

    This declaration includes the name of the procedure; the procedure's parameters; and type of value (if any) returned by the procedure.

    PLM syntax is used for the procedure declarations. If a procedure expects parameters, they are enclosed in parentheses. If a data-type is assigned to the procedure, then the procedure is called as a function and returns a value of the specified data-type.

    Most of the required device-dependent procedures are called as functions, and most of these functions return an error code. This is a one-word value. It is equal to zero for no error, or equal to a CTOS, GPS or user-defined error code. Error codes **15300** to **15319** may be used by GPS device-driver writers.

2) **Description of the function to be performed by the procedure.**

3) **Description of each input parameter.**

    For each input parameter, its use and data-type are described.

    Please note that some input parameters are used to specify where the procedure's output data is to be placed.

9.1 Required Device-Dependent Procedures

The following procedures must always be implemented in the DDP of any GPS device driver.

**DdBeforeConvertG: PROCEDURE(pInfo, pInfoRet)
ERCTYPE PUBLIC REENTRANT;**

This device-dependent routine is called **after** all of the installation parameters have been obtained and processed by the Core, and **before** the device driver is converted to a system service.

GPS 1.0 device drivers did not require this routine.

The purpose of this routine is to allow device-dependent processing to be performed before **ConvertToSys**. Whether or not this procedure does anything is determined by the device-driver designer.

An error code must be returned by this procedure. If it is a non-zero value, installation of the device driver will terminate.

Parameters –

pInfo:	A pointer to an array of bytes. The first two bytes contain the version word. This is the version value obtained from the GPS request file. No other fields are defined in this "Info" byte structure for the 2.0 release of GPS.
pInfoRet:	A pointer to an array of bytes. Information that is to be provided by the device-dependent code should be placed into this byte array. No such information is defined in this 2.0 release of GPS. **Therefore, pInfoRet should NOT be used by GPS 2.0 device drivers!**

A stub for this procedure is included in the GPS 2.0 device-driver Core library. If **DdBeforeConvertG** will do no processing in a particular device driver, then the already-defined stub may be included. It is included by referencing this library module in the list of device-specific object modules.

DdBeginDocumentG: PROCEDURE(psbDocName, psbUserName) ErcType PUBLIC REENTRANT;

This routine is called at the beginning of each document to notify the DDP that a new document is being begun. Document and User names are provided for possible use by the DDP. An "error code" (one word; equal to zero if no error) is returned by this routine.

A "beginning of document" form-feed should **NOT** be done by this routine.

Parameters –

psbDocName:	A pointer to the "sb" string containing this new document's name.
psbUserName:	A pointer to the "sb" string containing the name of the user associated with this document.

DdEndDocumentG: PROCEDURE ErcType PUBLIC REENTRANT;

This routine is called at the end of each document, in case there is any end-of-document processing that is to be done by the DDP of the GPS device driver. If the end-of-document processing requires that bytes be output to the port, it is the responsibility of this routine to make that output happen (i.e., there will be no subsequent call to **DdFlushBufferG** by the GPS Core to force anything out). An "error code" is returned by this routine.

An "end of document" form-feed should **NOT** be done by this routine.

Parameters – none

DdFirstChanceG: PROCEDURE ERCTYPE PUBLIC REENTRANT;

This device-dependent routine is called at the very beginning of device-driver installation (right after the conditional INTERRUPT-3 instruction).

GPS 1.0 device drivers did not require this routine.

The intended purpose of this routine is for debugging by users of PrintGen.

An error code must be returned by this procedure. If it is a non-zero value, installation of the device driver will terminate.

Parameters - none

A stub for this procedure is included in the GPS 2.0 device-driver core library. If **DdFirstChanceG** will do no processing in a particular device driver, then the already-defined stub may be included. It is included by referencing this library module in the list of device-specific object modules.

DdFlushBufferG: PROCEDURE ErcType PUBLIC REENTRANT;

This procedure is called when any data currently buffered within the DDP of the GPS device driver must be output to the device. An "error code" is returned by this routine.

Parameters - none

DdFormFeedG: PROCEDURE ErcType PUBLIC REENTRANT;

This routine is called when the GPS Core wishes a form-feed operation to be performed. The DDP of the GPS device driver has the responsibility of flushing its buffers first, if necessary. An "error code" is returned by this routine.

This is device-dependent function is called at the end of each page (and maybe at the beginning of a document). Another device-dependent function (**DdNewPageG**) is called at the beginning of each page.

Parameters – none

DdInitializeG: PROCEDURE(pbModeRet) ErcType PUBLIC REENTRANT;

This routine is called when the port is acquired for the device driver. This occurs when the first document is output to a spooled device, or each time that a client acquires a direct-print device. Any required output-device initialization or any initialization of the DDP should be done by this routine. An "error code" is returned by this procedure.

A value must be returned to GPS-Core by this routine -- the returned value is the output image-mode that the DDP will be using. (See the parameter description below for a definition of image-mode values.) Both the DDP and Core output data to the device. (The GPS Core outputs data directly during *Transparent* mode, for example.) While the Core is doing its output, it uses the image-mode that it needs. When it has completed its data output, it restores the port's image-mode to the value desired by the DDP. The value returned by this **DdInitializeG** procedure is this image-mode value that will be restored by the Core when necessary.

Even though this routine must specify its desired image mode to the GPS Core, the DDP of the GPS device driver must, itself, initially set the port to the image mode that it desires.

Part of the initialization done by the **DdInitializeG** routine should include a call to **SetDdSamMode** to set the output port to the image-mode value desired by the device-dependent code. Generally, an image mode of *Binary* should be specified. Devices that require control sequences, must use the *Binary* image mode. If the *Normal* image mode is specified by this procedure, then any RETURN's or TAB's output by the DDP are interpreted as specified in the device-driver installation form. This interpretation is performed by the CTOS output service.

Parameters –

pbModeRet:	A pointer to the byte that the desired image-mode is to be written into by this procedure. The valid image-mode values are: 0 Normal, 1 Image, and 2 Binary. (See the definitions of these image modes under **Printer Byte Streams** and **Spooler Byte Streams** in CTOS Volume I.) Since the GPS Core does the interpretation and extraction of embedded escape sequences, the Image and Binary modes are equivalent for the output (device-dependent) portion of a GPS device driver.

DdNewLineG: PROCEDURE ErcType PUBLIC REENTRANT;

This routine is called to cause device output to advance to the next line. An "error code" is returned by this procedure.

Parameters - none

DdNewPageG:
PROCEDURE (pPD) ErcType PUBLIC REENTRANT;

This routine is called to define the beginning of a new page of output. The device-dependent routine should perform whatever functions are necessary to begin a new page of output. Page coordinates should be reset. A form-feed operation should **NOT** be performed by this routine. The input parameter points to an updated Page-Descriptor. An "error code" is returned by this procedure.

(See also the above description of the **DdFormFeedG** procedure.)

Parameters -

pPD:	A pointer to the Page-Descriptor data structure (defined in the GPS DATA STRUCTURES OF INTEREST section, page 78.)

DdPutCharG: PROCEDURE (pToken) ErcType PUBLIC REENTRANT;

This routine is called to give the next output character to the device driver's DDP. The data passed to the device-dependent code is a **GPS Character Record**. An "error code" is returned by this procedure.

Parameters -

pToken:	A pointer to the *GPS Character Record*. The GPS Character Record (also referred to as the character token) contains the character to be output, its page position, its attributes, its font, etc. The GPS Character Record is described in the GPS CHARACTER-RECORD section (page 83).

9.2 Graphics–Processing Device–Dependent Procedures

The following procedures must always be implemented in the DDP of a GPS device driver that outputs graphics. A GPS device driver that does not process graphics need not include these routines.

DdGrBeginG: PROCEDURE ErcType PUBLIC REENTRANT;

This procedure is called at the beginning of each graphics object, or picture. This enables the DDP to perform any necessary text–processing termination and/or graphics–processing initialization. An "error code" is returned by this routine.

Parameters – none

DdGrColorTableG: PROCEDURE(iColorTable, pbColorTable, cwColorTable) ErcType PUBLIC REENTRANT;

This routine is called by the Core portion of the GPS driver to set some or all of the device's color table. See the description of the **GPAMGrColorTable** GPAM call in the Generic Print System Programmer's Guide. The DDP of a GPS, graphics device driver must include this function definition, even if it does not support "colors". An "error code" is returned by this routine.

Parameters –

iColorTable:	This word is the index of the first palette entry to be changed by this call to this procedure.
pbColorTable:	A pointer to the array of new color values to placed into the device's color table.
cwColorTable:	This word contains a count. It is a count of the number of words in the array referenced by **pbColorTable**.

DdGrEndG: PROCEDURE ErcType PUBLIC REENTRANT;

This procedure is called at the end of each graphics picture that is sent to the DDP. This enables the DDP to perform any necessary graphics-processing termination and/or text-processing initialization. An "error code" is returned by this routine.

Parameters - none

DdGrPolyGonG: PROCEDURE(pPointsBfr, cPoints, sPointsBfr, fPerimeterVisible, iPerimeterColor, iInteriorStyle, iHatch, iFillColor) ErcType PUBLIC REENTRANT;

This function is called to draw a polygon. The parameters include a list of points to be connected to form the polygon. The point-coordinate units are 1440ths of an inch. Coordinates (0,0) address the upper left of the page. An "error code" is returned by this routine.

Parameters –

pPointsBfr:	A pointer to the array of points defining the polygon.
cPoints:	This word is a count of the number of points defining the polygon. **Counts of zero and two points are valid values -- the DDP must cope with these possible values!**
sPointsBfr:	This word is a count of the number of bytes in the **PointsBfr**.
fPerimeterVisible:	If this flag is set to TRUE, then the line defining the polygon's perimeter should be drawn.
iPerimeterColor:	If the polygon's perimeter is to be drawn, then it is to be drawn with the color specified by this color-table index. **Not yet implemented! Currently, this parameter is always zero. This routine should check for a value of zero in this parameter. If it is equal to zero, then use the fill color for the perimeter. Otherwise use the perimeter color specified by this parameter.**
iInteriorStyle:	This word index specifies what type of fill (if any) is to be used in the polygon's interior. See the <u>Generic Print System Programmer's Guide</u> for the definition of interior styles.
iHatch:	If the polygon's interior is to be "hatched", then this word specifies which of the hatch types is to be used. See the <u>Generic Print System Programmer's Guide</u> for the definition of hatch types.
iFillColor:	If the polygon's interior is to be filled, then it is to be drawn with the color specified by this color-table index.

DdGrPolyLineG: PROCEDURE(pPointsBfr, cPoints, sPointsBfr, iColor, iLineType)
ErcType PUBLIC REENTRANT;

This function is called to draw a polyline. The parameters include a list of points to be connected to form the polyline. The point-coordinate units are 1440ths of an inch. Coordinates (0,0) address the upper left of the page. An "error code" is returned by this routine.

Parameters –

pPointsBfr:	A pointer to the array of points defining the polyline.
cPoints:	This word is a count of the number of points defining the polyline. **Counts of zero and two points are valid values -- the DDP must cope with these possible values!**
sPointsBfr:	This word is a count of the number of bytes in the **PointsBfr**.
iColor:	The line is to be drawn with the color specified by this color-table index.
iLineType:	This word specifies the type of line that is to be used to draw this polyline. See the Generic Print System Programmer's Guide for the definition of line types.

10.0 DEVICE-DRIVER STATUS MESSAGES

A large amount of status data is available for each installed GPS device driver. Most of this data is generated, collected and maintained by the GPS Core. Applications obtain device-driver status information through **GetGPSStatus** calls to the Generic Print System.

Some of the status data can be directly generated by the device-dependent code. These items are three device-status messages. The messages are:

1) PAUSE message.
2) FONT-REQUIRED message.
3) FORM-REQUIRED message.

These messages can also be generated by the Core. Their is no conflict between the Core-generation and DDP-generation of these messages.

10.1 PAUSE Message

The reason that a device has been paused can be stated in this message. For example, if a device has timed out, the Core places an appropriate string into this status message.

See the description of the **DdManualIntervention** utility (page 71). The **DdManualIntervention** utility always "activates" the PAUSE message before pausing the device, and then "de-activates" the PAUSE message after the device has been restarted.

DDsbPauseMessage:	Byte string containing the current PAUSE message. String is 61 bytes long. The first byte of the string is a count of the number of following bytes that comprise the PAUSE message.
DDfShowPauseMessage:	When set to **TRUE**, the string in **DDsbPauseMessage** is part of the device driver's current status -- the PAUSE message is "activated". When set to **FALSE**, the information in **DDsbPauseMessage** is not part of the current status -- the PAUSE message is "de-activated".

10.2 FONT-REQUIRED Message

Message string which specifies the font that must be supplied by an operator. This is not a complete font specification, and should not be confused with a font key. The name of a print wheel required by a Daisy-Wheel-printer, or the number of a required HPLaserJet font cartridge would be placed into this message string.

This message string is a special case of the general PAUSE message, and is redundant. It is used, however, because some existing applications depend upon its presence. When some operator action is needed to effect a font change, appropriate messages should be placed into both the PAUSE message and the FONT-REQUIRED message, and both of the messages should be "activated". Only the font name (e.g., wheel name or cartridge number) should be placed into the FONT-REQUIRED message. A complete statement of what is to be done by the operator should be placed into the PAUSE message.

DDsbFontNeeded:	Byte string containing the current FONT-REQUIRED message. String is 41 bytes long. The first byte of the string is a count of the number of following bytes that comprise the FONT-REQUIRED message.
DDfNeedFontChange:	When set to **TRUE**, the string in **DDsbFontNeeded** is part of the device driver's current status -- the FONT-REQUIRED message is "activated". When set to **FALSE**, the information in **DDsbFontNeeded** is not part of the current status -- the FONT-REQUIRED message is "de-activated". When set to **TRUE**, operator intervention is required to effect the requested font change. When set to **FALSE**, no "font-change" intervention is required of the operator.

10.3 FORM-REQUIRED Message

Message string which specifies the form that must be supplied by an operator.

This message string is a special case of the general PAUSE message, and is often redundant. It is used, however, because some existing applications depend upon its presence, and it contains only a form–name string. When some operator action is needed to effect a forms change, appropriate messages should be placed into both the PAUSE message and the FORM–REQUIRED message, and both of the messages should be "activated". Only the form name (e.g., "PayrollChecks") should be placed into the FORM–REQUIRED message. A complete statement of what is to be done by the operator (e.g., "Load PayrollChecks forms and restart printer".) should be placed into the PAUSE message.

DDsbFormNeeded:	Byte string containing the current FORM–REQUIRED message. String is 21 bytes long. The first byte of the string is a count of the number of following bytes that comprise the FORM–REQUIRED message.
DDfNeedFormsChange:	When set to **TRUE**, the string in **DDsbFormNeeded** is part of the device driver's current status -- the FORM–REQUIRED message is "activated". When set to **FALSE**, the information in **DDsbForm-Needed** is not part of the current status -- the FORM–REQUIRED message is "de-activated". When set to **TRUE**, operator intervention is required to effect the requested forms change. When set to **FALSE**, no "forms–change" intervention is required of the operator.

11.0 DEVICE-DEPENDENT OUTPUT ROUTINES

This section lists the routines that are to be used for device output by the DDP of the GPS device driver.

All device output performed by the DDP of the GPS device driver must be done via the procedures listed in this section. **The DDP must never attempt output to the GPS device by directly accessing I/O ports or making calls to the CTOS I/O routines!**

Each output procedure's specification consists of:

1) **A procedure declaration.**

   This declaration includes the name of the procedure; the procedure's parameters; and type of value (if any) returned by the procedure.

   PLM syntax is used for the procedure declarations. If a procedure expects parameters, they are enclosed in parentheses. If a data-type is assigned to the procedure, then the procedure is called as a function and returns a value of the specified data-type.

   Most of these output procedures used by the DDP of the GPS device driver are called as functions, and most of these functions return an error code. This is a one-word value. It is equal to zero for no error, or equal to one of CTOS or Generic Print System error codes.

2) **Description of the function that will be performed by the procedure.**

3) **Description of each parameter**

   For each parameter, its use and data-type are described.

   Please note that some parameters are used to specify where the called procedure will place "returned" data values.

PLM declarations of these modules are in the PLM "INCLUDE" file **DdSam.idf**, which comes with the PrintGen package.

SetDdSamMode: PROCEDURE (bMode) ErcType PUBLIC REENTRANT;

This procedure can be called by the DDP to set the device-output image mode. (See the parameter description below for details.) This procedure must be called at least once by the DDP's **DdInitializeG** routine (see page 38). An "error code" (one word; equal to zero if no error) is returned by this routine.

Parameters -

bMode: A byte specifying the image-mode value for the GPS output port. The valid image-mode values are:

0 *Normal*, and
2 *Binary*.

Avoid using the *Normal* value, unless there is a compelling reason to do so.
(See the definitions of these image modes under **Printer Byte Streams** and **Spooler Byte Streams** in The CTOS Operating System Manual, Volume I.)

WriteDdSamByte: PROCEDURE (bChar) ErcType PUBLIC REENTRANT;

Call this procedure to output one byte to the GPS device. An "error code" is returned by this routine.

Parameters -

bChar: The byte that is to be output.

WriteDdSamRecord: PROCEDURE (pb, cb, pCbRet) ErcType PUBLIC REENTRANT;

Call this procedure to output a string of bytes. The parameters used when calling this routine specify the address of the string of characters, the number of characters to be output, and the address of the word to receive the successful–output count. An "error code" is returned by this procedure.

Parameters –

pb:	A pointer to the vector of bytes to be output to the GPS device.
cb:	A word containing a count of the number of bytes in the byte vector pointed to by the parameter **pb**.
pCbRet:	A pointer to a word that will be written into by the **WriteDdSam–Record** procedure. **WriteDdSamRecord** writes into this word a count of the number of bytes that were actually output.

WriteDdXlatedByte

See the description of this "output" utility in the FONT–RELATED UTILITIES section (page 66).

12.0 AVAILABLE GPS UTILITIES

This section lists the functions that may be used by the device driver's DDP. Some of these are part of the GPS Core, and some are separate modules that may be linked into a GPS device driver.

Utilities are grouped into the following subsections:
- Font–Related Utilities
- Device–Setup–Field Utility
- Miscellaneous Utility Functions

Each output procedure's specification consists of:

1) **A procedure declaration.**

   This declaration includes the name of the procedure; the procedure's parameters; and type of value (if any) returned by the procedure.

   PLM syntax is used for the procedure declarations. If a procedure expects parameters, they are enclosed in parentheses. If a data–type is assigned to the procedure, then the procedure is called as a function and returns a value of the specified data–type.

   Most of these output procedures used by the DDP of the GPS device driver are called as functions, and most of these functions return an error code. This is a one–word value. It is equal to zero for no error, or equal to one of CTOS or Generic Print System error codes.

2) **Description of the function that will be performed by the procedure.**

3) **Description of each parameter**

   For each parameter, its use and data–type are described.

   Please note that some parameters are used to specify where the called procedure will place "returned" data values.

12.1 Font-Related Utilities

The utilities described in this section are those which may be called to obtain font-related information. The primary font information required by device-dependent code is Second-Level Translation data. Several routines described here provide this information. The other routines return additional font-related information.

The details presented in this section supplement the Font Database and Font Service information provided in the Printing Guide, and the information presented in the DEVICE-DRIVER FONT PROCESSING section (page 18) of this document.

Font Handles:

Since a document may consist of a variety of fonts, the desired font must be specified when calling most of these font-data utilities. A font is specified by a **font handle**.

A font handle is a 16-bit datum. It is passed from the Core to the DDP. It is part of the Character Record. Each character or glyph passed to the device-dependent code for output has a font handle associated with it. In this document, a variable or parameter that contains a font handle is usually named **iFont**. Font handles are created by the device-driver Core. Font handles should not be changed by the device-dependent code, and no meaning should be attached to any particular font-handle values.

Font handles are not reassigned within a given output document. Therefore, various font data may be stored by the DDP, and requested only when a new font handle value is encountered. Such storage should be reset at the beginning of each output document.

Variety and Complexity of Font Utilities:

Not all of the routines listed here need be used to implement font and character translation. In fact, use of them all in a single device driver is usually erroneous. A variety of font utilities have been provided to service the variety of printer sophistications and complexities.

At the simplest, there is a routine which may be called by the device-dependent code to do all of the Second-Level Translation and output work. At the other end of the complexity spectrum are the rarely-used routines which return detailed, specialized font-related information. Most customer-written device drivers will utilize neither the simplest nor the most complex utilities.

The following font utilities can be roughly grouped as follows:

Simple/general –	WriteDdXlatedByte
Medium –	DdChXlate DdFntHandles DdFntKey DdGetAlias
Complex/Specialized –	DdChWidth DdDocToNative DdFntConfig DdGetChSet

The font-utility descriptions which follow describe the intended usage of each routine. Each description states whether or not the routine is "simple" or "complex".

Glyph and **Character** are used interchangeably in the following descriptions. This is done because all of the "characters" passed from the Core to the DDP for output are actually glyphs. (Glyph is defined in the DEVICE-DRIVER FONT PROCESSING section of this document, page 18.) The term **Character–Set** is used but not defined in the descriptions of the following utilities. The definition and explanation of Character-Set is found in the Font Database documentation.

DdChWidth: PROCEDURE(iFont, bChar, pwWidth)
ErcType PUBLIC REENTRANT;

Returns the width of the specified character for the specified Font. An "error code" is returned by this procedure.

Device-dependent code should rarely find use for this routine, as the width of each glyph or output character is included in that glyph's Character-Record.

This routine could be used, for example, to obtain the width of the device's underline glyph in a specific font.

Parameters -

iFont:	Font handle from Character-Record. (This uniquely identifies the font and CharacterSet.)
bChar:	Byte containing glyph code for which the width is requested.
pwWidth:	The width value will be placed into the word pointed to by this parameter.

DdChWidth is one of the "complex" font utilities.

DdChWidth is part of the device-driver Core, and is always available for use.

DdChXlate: PROCEDURE(iFont, bChar, prgbXltn, cbXltnMax, pcbXltnRet) ErcType PUBLIC REENTRANT;

Called to get the (Font Service-provided) translation for a character. The device-dependent code calls this routine to find out what device-specific commands are required to print the glyph specified in the Character-Record. An "error code" is returned by this procedure.

When this routine is called with a Second-Level font handle, a Second-Level Translation is returned. The Second-Level Translation for an output glyph is the printer-command string which causes that glyph to be printed. All of the font handles passed to the DDP in Character-Records are Second-Level font handles.

> This routine is not limited to performing Second-Level Translations. It also performs First-Level Translations. Which translation it performs is controlled by the font-handle value passed to it. All of the font handles passed to the DDP in Character-Records would cause Second-Level Translations to be performed. Font handles which cause First-Level Translations to be performed are those font handles which represent document characters (the α characters, as described in the DEVICE-DRIVER FONT PROCESSING section (page 18). Other font utilities, described in this section, may be used by device-dependent code to obtain such First-Level font handles.
>
> When the font-handle value (**iFont**) passed to **DdChXlate** is that of a First-Level font handle, **DdChXlate** assumes that **bChar** contains an α value, and it performs a First-Level Translation ($\alpha \rightarrow \beta$). When the font-handle value (**iFont**) passed to **DdChXlate** is that of a Second-Level font handle, **DdChXlate** assumes that **bChar** contains a β value, and it performs a Second-Level Translation ($\beta \rightarrow \chi$).

Parameters -

iFont:	Font handle of font for which the translation is to be performed. This is usually the font handle from the Character-Record. (This uniquely identifies the font and CharacterSet.)
bChar:	Byte containing the value of the character or glyph to be translated.

prgbXltn:	Points to the device-dependent code's string that is to receive the translation data. This memory must be defined and allocated by the DDP. Format of the translation data is described in the FONT ESCAPE SEQUENCES section (page 74) of this document, and in the Printing Guide.
cbXltnMax:	Size of **prgbXltn** in bytes. This is the maximum size of any translation string that will be returned by **DdChXlate**. Assure that this string is large enough hold the longest translation string specified for this device in the Font Database.
pcbXltnRet:	Actual number of returned translation-data bytes will be placed into the word referenced by this pointer.

DdChXlate is part of the device-driver Core, and is always available for use.

DdDocToNative: PROCEDURE(bCSID, bCharDoc, iFont, pbCharN, piFontN, pwWidth) ErcType PUBLIC REENTRANT;

Given a "document" character code (α) and a valid font handle, this routine finds and returns the corresponding device glyph code (β), its width and its font handle. It performs the First-Level Translation, and obtains the width of the printer glyph. An "error code" is returned by this procedure.

> This routine is a special combination of the **DdChWidth** and **DdChXlate** routines. The font handle passed to **DdDocToNative** may be either a First-Level or Second-Level font handle. **DdDocToNative** always assumes that **bCharDoc** contains an α value, and always performs a First-Level Translation.

DdDocToNative is usually called by the device-dependent code to obtain information about particular printer glyphs that were never passed to it by the Core. If, for example, the DDP was to use the printer's underline character to underline some other glyph, **DdDocToNative** could be called to obtain the appropriate underline-glyph information from the Font Database. The value of the **bCharDoc** parameter would be *5Fh*, which is the code assigned to the underline character in CT documents. The value of the **iFont** parameter would be the font-handle value found in the Character-Record of the glyph that is to be underlined. The corresponding printer glyph code and its font handle would be returned in the locations pointed to by **pbCharN** and **piFontN**, respectively. These two returned values can then be used in a call to **DdChXlate** to obtain the printer command string which will cause the printer's underline glyph to be printed.

Restrictions -

1. Currently, only the CT "document" character set is supported. Therefore, zero should be the value of the **bCSID** parameter.

2. For a 1-to-many First-Level Translation, only the first char is returned to the location pointed at by **pbCharN**.

3. If no translation can be found; **iFont**, 0 and 0x20 are returned in the locations pointed to by **piFontN**, **pwWidth** and **pbCharN** respectively.

Parameters -

bCSID:	Byte value specifying which "document" character set.

bCharDoc:	Byte containing the code of the "document" character (α) for which the corresponding "native"-character (β), width and font-handle are desired.
iFont:	A valid font-handle for the desired font. May be a Second-Level font handle.
pbCharN:	The code for the corresponding "native" printer glyph (β) will be placed into the byte pointed to by this pointer value.
piFontN:	The font handle for the printer glyph will be placed into the font-handle variable pointed to by this pointer value.
pwWidth:	The printer glyph's width will be placed into the word pointed to by this pointer value.

DdDocToNative is one of the "complex" font utilities.

DdDocToNative is not part of the device-driver Core. If **DdDocToNative** is to be used, it must be added to the device-driver link. The **DdDocToNative** routine is contained in the **Font2Util** module of the Core library.

**DdFntConfig: PROCEDURE(psbDevType, cbDevTypeMax)
FlagType PUBLIC REENTRANT;**

Called to obtain some detailed information relevant to the Font Service and how it is being used by this device driver. If the Font Service is installed and in use, this function returns TRUE. The (font) device type currently being used by the device driver is copied into the client-specified string.

The Font Service is always installed when a GPS 2.0 device driver is running. With GPS 2.0, **DdFntConfig** will always return TRUE.

The most likely reason that device-dependent code would call this routine is to determine its Font-Device-Type. See the description of **DDsbFDevice**, in the GPS DEVICE-SPECIFICATION PARAMETERS section (page 92) for an explanation of Font-Device-Type.

Parameters -

psbDevType:	A pointer to the device-dependent code's string that is to receive the Font-Device-Type string. (Note that this is an "sb" structure.)
cbDevTypeMax:	Word containing a count of the maximum number of bytes that may be stored in the data structure pointed to by **psbDevType**.

DdFntConfig is one of the "complex/specialized" font utilities.

DdFntConfig is not part of the device-driver Core. If **DdFntConfig** is to be used, it must be added to the device-driver link. The **DdFntConfig** routine is contained in the **Font2Util** module of the Core library.

DdFntHandles: PROCEDURE(iFont, phXlate, phWidth, pwNumerator, pwDenominator, pfMonoSpace) ErcType PUBLIC REENTRANT;

This utility returns information about the **iFont**–specified font. The returned information includes "handles" and characteristics that may be used to uniquely identify the translation and width sets associated with the font. An "error code" is returned by this procedure.

Within a given document, a specific table of translations is uniquely identified by a "translation handle". Within a given document, a specific table of widths is uniquely identified by the triple: ("width handle", numerator, denominator). The width handle is inadequate for unique identification because "scaled" fonts may use the same width table for many point sizes.

An example user of **DdFntHandles** is the Imagen8300 device–dependent code. The Imagen printer hardware performs character translations itself, due to the very large number of possible glyphs that it prints. Before a document can be printed on an Imagen (using imPress), "maps", which convert 7–bit commands into arbitrary, 16–bit glyph numbers, must be sent to the Imagen printer. A variety of Imagen maps may be defined at the same time, and need not be redefined within a document. There are many cases where the same Imagen map may be used for all of the different point sizes of a particular family. Whenever the Imagen8300 device driver must print a glyph whose font handle it has not already processed, the device–dependent code calls **DdFntHandles** to get the translation handle corresponding to this new font handle. If that translation handle has never before been encountered, then the DDP uses Font Service translation data to define another Imagen map. If, however, this translation handle has been encountered before, then the device–dependent code just switches to the already–defined, corresponding Imagen map. It does not need to do the work of creating the map again.

Parameters –

iFont:	Font handle which specifies the font for which font information is desired. This is usually a font handle from a Character–Record. (This uniquely identifies the font and CharacterSet.)
phXlate:	Handle of the translation table associated with **iFont** will be placed into the word referenced by this pointer.

PhWidth:	Handle of the Width table associated with **iFont** will be placed into the word referenced by this pointer. (This handle, along with font numerator and denominator, uniquely identifies the width set associated with **iFont**.)
pwNumerator:	Each original width-table entry has been multiplied by the value placed into the word referenced by this pointer. The Font Service and Core perform this scaling. The device-dependent code need not do any scaling.
pwDenominator:	Each original width-table entry has been divided by the value placed into the word referenced by this pointer. The Font Service and Core perform this scaling. The device-dependent code need not do any scaling.
pfMonoSpace:	The byte referenced by this pointer is set to TRUE if font is monospaced.

DdFntHandles is not part of the device-driver Core. If **DdFntHandles** is to be used, it must be added to the device-driver link. The **DdFntHandles** routine is contained in the **Font2Util** module of the Core library.

DdFntKey: PROCEDURE(iFont, pFontKey, cbFontKeyMax, pcbFontKeyRet) ErcType PUBLIC REENTRANT;

Obtains the font key corresponding to a font handle. An "error code" is returned by this procedure.

A font key is a set of values that request or specify a font in the Font Database. A font key is used by the Core to request font information from the Font Service. The Font Service returns another font key (along with the requested font data) which specifies that font for which the data has been returned. If the specified font cannot be exactly matched in the Font Database, the Font Service determines which font data should be returned, and this second font key describes that font. The Core assigns font handles to the obtained data, and passes font handles to the device-dependent code. When **DdFntKey** is called with such a font handle, **DdFntKey** obtains and returns the corresponding font key.

Parameters –

iFont:	Font handle (from Character-Record) that uniquely identifies the font and CharacterSet.
pFontKey:	The corresponding font key will be placed into the data area pointed to by this parameter.
cbFontKeyMax:	Word containing a count of the number of bytes in the data area that is to receive the font key.
pcbFontKeyRet:	Actual byte count of the returned font key will be placed into the word pointed to by this parameter.

DdFntKey is part of the device-driver Core, and is always available for use.

**DdGetAlias: PROCEDURE(iFont, prgbAlias, cbAliasMax,
pcbAliasRet)
ErcType PUBLIC REENTRANT;**

Routine returns the "alias" string corresponding to the specified font-handle. The Font Database defines an "alias" string for each font/CharacterSet pairing. An "error code" is returned by this procedure.

The font alias is a string of arbitrary function and format. Its definition and usage is device dependent.

The Daisy driver, for example, uses it to store the name of the print wheel needed to print characters in the requested font. The simple ASCII driver does not use it at all.

Parameters -

iFont:	Font handle specifying the font/CharacterSet whose alias string is to be obtained.
prgbAlias:	Points to the device-dependent code's string that is to receive the alias string.
cbAliasMax:	Word containing a count of the maximum number of bytes that may be placed into the string referenced by **prgbAlias**.
pcbAliasRet:	Actual number of bytes in the returned alias string will be placed into the word referenced by this pointer.

DdGetAlias is part of the device-driver Core, and is always available for use.

DdGetChSet: PROCEDURE(iFont, bCharSet, piFont)
ErcType PUBLIC REENTRANT;

This routine may be called to obtain the font handle for an additional CharacterSet of the specified font. Given a valid font handle and CharacterSet identifier, the font handle specifically for that CharacterSet will be returned. (CharacterSets and valid CharacterSet identifiers are defined in the Font Database documentation.) An "error code" is returned by this procedure.

If a font handle has not yet been defined for the specified font/CharacterSet combination, the Core will create it and update its tables before returning the new font handle to the caller of this utility.

Parameters -

iFont:	A valid font handle, used to specify the desired font. It does not matter which of the font's CharacterSets is referenced by this font handle. Usually this will be a font-handle value obtained via a Character-Record from the Core, but it may be any Core-defined font handle for the desired font.
bCharSet:	Byte containing a CharacterSet identifier. This specifies which CharacterSet to get the font handle for.
piFont:	Pointer to a word. A font handle will be placed into this word by **DdGetChSet**. This new font handle will reference the same font that **iFont** does, but it will also be the font handle assigned to the **bCharSet**-specified Character-Set.

DdGetChSet is one of the "complex/specialized" font utilities.

DdGetChSet is part of the device-driver Core, and is always available for use.

WriteDdXlatedByte: PROCEDURE(iFont, bChar) ErcType PUBLIC REENTRANT;

This utility provides the simplest means of outputting printer glyphs from the device-dependent code. It does all of the Second-Level Translations, and outputs the resultant printer commands to the device. (More flexibility and control may be obtained by using some of the other utilities described in this section.) An "error code" is returned by this procedure.

WriteDdXlatedByte takes a font handle and a printer-glyph code (β) as input parameters. It performs the Second-Level Translation, and outputs the resultant printer commands (χ) to the hardware device. If this routine is called with the **iFont** and **bChar** values from a given Character-Record, the appropriate printer commands are output by it. Thus, the device-dependent routine is relieved of the responsibility for obtaining and outputting the printer commands (χ) corresponding to each printer-glyph code (β) received from the Core.

(**WriteDdXlatedByte** is very much a GPS output utility, and could have been described in the DEVICE-DEPENDENT OUTPUT ROUTINES section (page 49). It is described here, however, since it also involves font translations.)

An example of a routine that would use this routine is the Daisy device driver. Assume that one of its glyph codes (β) is 250. Assume also that the corresponding Second-Level Translation converts 250 to the three command bytes:

(SHIFT-IN, 123, SHIFT-OUT).

Then calling **WriteDdXlatedByte** with **bChar** = 250, will result in the three byte-values:

SHIFT-IN
123
SHIFT-OUT

being output to the printer for the device-dependent routine.

An example of a routine that would **not** (in fact, could not) use this routine is the Imagen8300 device driver. Before it can output any characters of a particular font, the driver must construct and download a "mapping" table for that entire font. Thus, the Imagen driver must collect all of a font's Second-Level Translations at one time to construct this Imagen "mapping" command. The Imagen driver makes use of the **DdChXlate** routine to accomplish this.

See translation-escape rules 3, 5 and 6 in the FONT ESCAPE SEQUENCES section (page 74) for details of **WriteDdXlatedByte**'s Second-Level-Translation processing.

Parameters –

iFont:	Font handle corresponding to the printer glyph that is to be printed. This font handle is usually from the Character–Record that contained the printer glyph to be output.
bChar:	Byte containing the code for the printer glyph that is to be printed by this routine.

WriteDdXlatedByte is a "simple/general" font utility.

WriteDdXlatedByte is part of the device–driver Core, and is always available for use.

12.2 Device-Setup-Field Utility

The Print Manager's device-driver installation form includes a "Device Setup" field. Device-dependent parameter strings may be entered in this string when installing a GPS device driver. All data entered into the "Device Setup" field will be found in the device-driver string **DdInstall.sbDevParams**.

The following utility may be used to test for arbitrary keywords in the "Device Setup" field, and to obtain the associated data strings.

**DdGetSetupKey: PROCEDURE(pS, cS, pV, cVMax,
pcVRet) WORD PUBLIC REENTRANT;**

This routine finds a key–word in the device–parameters string, and returns the string following that key–word.

A caller–defined string is passed to this routine. It searches for that string in the "Device Setup" field. If found, the function value returned by **DdGetSetupKey** is an index into **DdInstall.sbDevParams**. This is the index of the first byte following the matched string. If the caller–defined string is not found, the function value returned by **DdGetSetupKey** will be 0FFFFh.

The caller of this utility also specifies a recipient string. If the caller–specified key–word string is found, then the string following this key word is copied by **DdGetSetupKey** into the receiving string. This "following" string is terminated by the first space after the key word.

Parameters –

pS:	A pointer to the key–word string to search for.
cS:	Word containing the count of the number of bytes in the string referenced by **pS**.
pV:	Pointer to a data area defined by the caller. If the key–word string is found, then the string following it is copied into this data area.
cVMax:	Word containing a count of the maximum number of bytes that may be copied to the data area referenced by **pV**.
pcVRet:	Pointer to a word in which will be placed the length (bytes) of the string following the key–word.

DdGetSetupKey is part of the device–driver Core, and is always available for use.

12.3 Miscellaneous Utility Functions

This section describes a variety of functions available for use by the writer of a GPS device driver. In some cases, these are routines which should be used to request some data or action from the GPS Core. In other cases, they are just general utilities which are already implemented in the Core.

DdHexToB: PROCEDURE(prgbH, bcbH, prgbB, bcbBMax, pbcbBRet) ErcType PUBLIC REENTRANT;

This routine will convert a hexadecimal-character string into a string of binary, byte values.

If the character string does not have an even number of bytes, or has characters that are not hex digits, then an error is returned.

Parameters –

prgbH:	Pointer to the string of hexadecimal characters that are to be converted into binary values.
bcbH:	A byte containing the number of characters in the string referenced by **prgbH**.
prgbB:	Pointer to a string. Resultant binary bytes are placed into this string.
bcbBMax:	A byte containing the maximum number of bytes in the string referenced by **prgbB**.
pbcbBRet:	Pointer to a byte. Number of binary bytes placed into the string referenced by **prgbB** is written into this byte.

DdHexToB is part of the device-driver Core, and is always available for use.

DdManualIntervention: PROCEDURE
ErcType PUBLIC REENTRANT;

This routine will cause device output to pause until a restart operation is performed upon that device.

Before calling this routine, the device-dependent code should usually set up some status messages that explain what operator action is required.

Before pausing device output, the **DdManualIntervention** utility "activates" the device's "pause message". Before returning to the caller, it "de-activates" the "pause message". If other of the driver's messages are to be displayed, they should be "activated" before calling **DdManualIntervention**, and "de-activated" after the call to **DdManualIntervention**. (See the DEVICE-DRIVER STATUS MESSAGES section (page 46) for a definition of these messages and their usage.)

If the print job has already been cancelled at the time that **DdManualIntervention** is called, then **ErcDdCancelled** is returned as the function's value, and the printer is not paused.

Parameters - none

DdManualIntervention is part of the device-driver Core, and is always available for use.

DdChangeForm: PROCEDURE (psbForm)
ErcType PUBLIC REENTRANT;

This routine may be called by the device-dependent code to initiate and perform "forms change" processing by an operator. The device-driver Core detects forms-change requests and processes them. Therefore, it is usually not necessary for the DDP to worry about processing forms changes. The **DdChangeForm** utility may, however, be used by device-dependent code if necessary.

DdChangeForm does the following:

- Nothing if the requested form is already the current form.
- Sets up the various "forms needed" messages to be displayed with the device's status.
- Puts the device into its paused state, until a restart operation is performed upon the device.
- Updates the Core's "current form" information.

If the name of the requested form is the null string, then the form specified in **DdsbFormDoc** is the form that is requested. If **DdsbFormDoc** contains a null string, then the "standard" form is requested of the operator.

Parameters –

psbForm:	Pointer to an "sb" string of bytes. This string contains the name of the form to be loaded before resuming output. This form name may be the null string. Length of this string must not be greater than 21 bytes.

DdChangeForm is part of the device-driver Core, and is always available for use.

MovB: PROCEDURE(prgbSource, prgbDest, cb)
PUBLIC REENTRANT;

This routine is the equivalent of the built-in, PLM **MovB** procedure. If device-dependent code is written in PLM, there is little point in using this GPS-driver routine. When using some other language such a C, this is a handy routine that is already a part of the GPS device driver.

MovB may be called to copy a specified number of bytes from one location to another.

No defined function value is returned by **MovB**.

Parameters -

prgbSource:	Pointer to a byte. This byte contains the first of the bytes that are to be copied to another location.
prgbDest:	Pointer to a byte. This byte is where the first of the copied bytes will be placed.
cb:	Word containing a count of the number of bytes to be copied.

MovB is part of the device-driver Core, and is always available for use.

13.0 FONT ESCAPE SEQUENCES

GPS's interpretation of First-Level-Translation data and Second-Level-Translation data is discussed in this section.

The purpose and formats of these font translations is discussed in the Font Database documentation, and in the DEVICE-DRIVER FONT PROCESSING section (page 18) of this document. Reading and understanding that information is prerequisite to understanding this section.

Translation data is returned to GPS by the Font Service. Some the returned translation values have special meaning for GPS. These special values are called **translation escapes**, or just **escapes**. All font translations are strings of one or more bytes. All **translation escapes** recognized by GPS are byte values.

Translation escapes are relevant to GPS device driver designers for several reasons. The designer is necessarily involved in the definition of the format and data used in the Font Database's font translations for her device. The GPS Core uses the First-Level-Translation data to define the printer glyphs to be output for each character received. Certain translation values (escapes) have pre-defined meanings for the GPS Core. The device-driver designer must not attempt to reuse these values for some other purpose, and she must understand the Core's interpretation of them so that she may properly command the Core via her First-Level-Translation data. Some utilities are provided to the GPS device-driver designer to perform Second-Level Translation functions and to output the Second-Level-Translation data. These utilities associate specific meaning with certain of the Second-Level values. Again, the GPS driver designer must know which these are and how they will be interpreted.

"Actual" and "suggested" translation escapes are described in the Font Database documentation.

Actual translation escapes are those used and generated by the Font Tool and Font Database. Included in this set would be the following escapes:

- Nil
- Alternate Character Set 1
- Alternate Character Set 2
- Alternate Character Set 3

Suggested translation escapes are those which may be entered by the Font-Tool user. Included in this set would be the following "escapes":

- Quote
- 16 Bit Code
- Shift Left
- Shift Out
- Printer Escape

The two sets may intersect -- i.e., the Font-Tool user may enter translation-escape values which are the same as those generated by the Font Tool.

GPS adds more translation-escape rules. These rules state what escapes are recognized by the GPS device driver, and how it interprets them. The rest of this section lists these translation-escape rules.

The basic translation-escape rule is:

The GPS Core and font utilities will recognize all documented actual and suggested translation escapes.

Detailed GPS translation-escape rules:

1. The Font-Tool user must enter all desired translation escapes, <u>except</u> those which are the "actual" Font Service escapes (e.g., **Nil**, **Alternate-Character-Set-1**, **Alternate-Character-Set-2** and **Alternate-Character-Set-3**).

2. The GPS routine **DdChXlate** returns a translation string. The returned translation string includes the translation escapes contained in the font translations.

   <u>Examples</u>:

   → A translation of **0105** is entered by the Font-Tool user. **DdChXlate** returns the 2-byte string **{01,05}**.

   → A translation of **05** is entered by the Font-Tool user. **DdChXlate** returns the 1-byte string **{05}**.

   → A translation of **2505** is entered by the Font-Tool user. **DdChXlate** returns the 2-byte string **{25,05}**.

   → A translation of **0000** is entered by the Font-Tool user. **DdChXlate** returns the 2-byte string **{00,00}**.

- → A translation of **070100** is entered by the Font-Tool user. **DdChXlate** returns the 3-byte string **{07,01,00}**.

3. The GPS routine **WriteDdXlatedByte** obtains and outputs the Second-Level-Translation data corresponding to the specified printer-glyph code. All translation escapes in the obtained data are interpreted; not output! (Although the interpretation of some of the "escapes" does result in the output of their value -- e.g., **Shift-Out**.)

   Examples:

   - → A translation of **0105** is entered by the Font-Tool user. **WriteDdXlatedByte** outputs the 1-byte string **{05}**.

   - → A translation of **05** is entered by the Font-Tool user. **WriteDdXlatedByte** does not output anything (because 05 is the Shift-Up escape, a First-Level Translation, which is an ignored error).

   - → A translation of **2505** is entered by the Font-Tool user. **WriteDdXlatedByte** outputs the 1-byte string **{25}** (because 05 is the Shift-Up escape, a First-Level Translation, which is an ignored error).

   - → A translation of **0000** is entered by the Font-Tool user. **WriteDdXlatedByte** outputs nothing (because 00 is the Nil escape, which is an ignored error).

   - → A translation of **070100** is entered by the Font-Tool user. **WriteDdXlatedByte** outputs the 1-byte string **{00}** (because 07 is the Alternate-Character-Set-1 escape, a First-Level Translation, which is an ignored error; and 01 "quotes" the following 00 byte.)

   - → A translation of **0F0100** is entered by the Font-Tool user. **WriteDdXlatedByte** outputs the 2-byte string **{0F,00}** (because 0F is the Shift-In escape, which is output as 0F; and 01 "quotes" the following 00 byte.)

4. Translation escapes that are legal in First-Level Translations:
   - Quote
   - Shift-Left
   - Shift-Right
   - Shift-Up
   - Shift-Down
   - Alternate-Character-Set-1
   - Alternate-Character-Set-2
   - Alternate-Character-Set-3
   - Cell-Width

   Any other translation escapes in the First-Level-Translation data are ignored when the Core is processing First-Level Translations.

5. Translation escapes that are legal in Second-Level Translations:
   - Quote
   - 16-Bit
   - Shift-Out
   - Shift-In
   - ESC (1Bh)

   Any other translation escapes in the Second-Level-Translation data are ignored by the **WriteDdXlatedByte** utility.

6. Any of the following Font Service "escapes" imply Quote also:
   - Alternate-Character-Set-1
   - Alternate-Character-Set-2
   - Alternate-Character-Set-3
   - Cell-Width

   Therefore, glyph code **04** in Alternate-Character-Set-3 should be specified in a First-Level Translation by entering **0904**, not **090104**.

   This also implies that the Alternate-Character-Set-Selection translation escape should immediately precede the printer-glyph codes that it affects.

7. **10h** is recognized and processed as a valid First-Level translation escape. This is the "Cell-Width" escape. When this escape is used, there should be a printer-glyph code immediately following.

   The width associated with this printer-glyph defines the width of the character cell. If any whitespace underlining or overstriking is in effect, a whitespace token for this width is generated by the Core to underline or overstrike the output character.

14.0 GPS DATA STRUCTURES OF INTEREST

GPS data structures that are used by the DDP of a GPS device driver are documented in this section.

PLM definitions for these data structures are included in **GpsDB.idf**. C-language definitions are in **StructGps.h**.

Please note that the following other data structures are described in other sections of this manual:

- GPS Character Record
- GPS Device-Specification parameters

Font Data Structures:

Font Service data structures of relevance to a GPS device driver are documented in the Printing Guide. The structures of particular interest to device-dependent code are: 1) **font key**, and 2) **alias string**.

Page Descriptor:

The information in this structure is passed to the DDP of a GPS device driver at the beginning of each page. Page-size and orientation information are contained in this structure.

This structure is defined by **PageDescDDType** in **GpsDB.idf**.

The Page Descriptor has been expanded for GPS 2.0. Each GPS 1.0 Page Descriptor was 8 bytes long. It is now 11 bytes. The additional information is in the structure's **bOrient**, **bVer** and **bStamp** fields.

Definition – Field Name	Size (bytes)	Usage
wLMx	2	Location of page's left margin.
wRMx	2	Location of page's right margin.
wTMy	2	Location of page's top margin.
wBMy	2	Location of page's bottom margin.
bOrient	1	Specifies the requested page orientation. Valid values: 1 = Landscape 2 = Portrait
bVer	1	Page–Descriptor format version. The value contained in the **bVer** field for this (GPS 2.0) format of the Page Descriptor is **20**.
bStamp	1	A "check" byte which should contain **05Ah**. If it does not contain **05Ah**, then the **bVer** field is not valid.

Page width may be obtained by subtracting the contents of **wLMx** from the contents of **wRMx**. Page length may be obtained by subtracting the contents of **wTMy** from the contents of **wBMy**.

Break–at–Installation Flag:

See the DEBUGGING AIDS section (page 104) for information about the use of this flag in debugging GPS device drivers.

Cancellation Flag:

The flag **DDfCanceling** is set to TRUE when the current document has been cancelled. It is reset to FALSE when the end of the cancelled document is encountered by the GPS Core. Useless processing can often be avoided by the device-dependent code if it tests **DDfCanceling** after each call to the GPS Output routines.

Document Type:

By examining the byte **DDbDocType**, the DDP can discover whether it is processing data from a GPAM or non-GPAM document. It is not necessary for the DDP to make this determination, but if there is a desire to do so, **DDbDocType** contains the information.

Valid **DDbDocType** values:

1 =	GPAM document.
2 =	non-GPAM document.
3 =	non-GPAM document.

A correct value in **DDbDocType** is not assured until the document's first call to **DdPutCharG**. If the document being processed is a non-GPAM document, and image-mode is not NORMAL, then **DdPutCharG** will not be called and **DDbDocType**'s validity cannot be assured.

DdShare:

DdShare is a large structure that contains a great deal of document-status information. Most of it is used by the GPS Core and should not be depended upon by the DDP. **DdShare** fields that contain valid data of use to the DDP are listed and described below.

This structure is defined by **SharedDDType** and **BeginPageDDType** in GpsDB.idf.

DdShare fields that may be used by the device-dependent code:

Field Name	Size (bytes)	Usage
bQuality	1	(See GPAMBeginPage description in the Generic Print System Programmer's Guide.)
wBinOut	2	This field is mis-named in the structure definition used by GPS device drivers. It is the field actually named **wBinIn**. (See GPAMBeginPage description in the Generic Print System Programmer's Guide.)
bSides	1	(See GPAMBeginPage description in the Generic Print System Programmer's Guide.)
fStagger	1	(See GPAMBeginPage description in the Generic Print System Programmer's Guide.)

If these DdShare values are to be used by device-dependent code, they should be examined when **DdNewPageG** is called. The above-listed values are then valid for that new page. Examination of DdShare values when **DdPutCharG** has been called, generally yields invalid data.

Driver-Process Stacks:

A GPS device driver will have up to three stacks. It always has at least two stacks. If it processes graphics, then it must have three stacks.

The three stacks are:

- Control-Process stack.
- Data-Process stack.
- VDM-Interpreter stack.

One of the stack's sizes is specified in the LINK command. The other two stacks' sizes are specified in the device driver's Device-Specifications-Parameters module. (See the GPS DEVICE-SPECIFICATION PARAMETERS section, page 87.)

The size of the **Control–Process stack** is specified in the device-driver LINK command. The Control Process generally requires the least stack space. Usually, the LINK command's default stack size is more than adequate for any GPS device driver.

All device-dependent code runs in the Data Process. All data output is performed by the device driver's Data Process. Therefore, the Data Process often requires more stack than the Control Process. How many bytes are needed for the **Data-Process stack** varies with each GPS device driver since the requirement is dependent upon the implementation of the DDP in each device driver. The space to be allocated for the Data-Process stack is specified in the driver's Device-Specifications-Parameters module. The symbol **sDpStack** specifies the number of bytes to be allocated to the Data-Process stack. The recommended starting value for **sDpStack** is **2048**.

Any stack-overflow problems will almost certainly involve only the Data-Process stack. To change the size of this stack: 1) the value of **sDpStack** must be changed; 2) the Device-Specifications-Parameters module must be re-assembled; and 3) the device driver must be re-linked.

If the GPS device driver in question does not process graphics data, the **VDM-Interpreter stack** is not required. If graphics data is to be processed, then space must be allocated for the **VDM-Interpreter stack**. The space to be allocated for the VDM-Interpreter stack is specified in the driver's Device-Specifications-Parameters module. The symbol **sStackVI** specifies the number of bytes to be allocated to the VDM-Interpreter stack. If graphics is not supported, set the **sStackVI** symbol to **1**. Set the symbol to **2048**, if graphics is supported by the device driver.
(Note: the symbol **sVdmWA** must also be set to specific values for the "graphics" and "non-graphics" cases.)

15.0 GPS CHARACTER-RECORD

All of the information about a character is passed from the GPS Core to the DDP in a data structure called the GPS Character Record. This data structure is also sometimes referred to as the "character token". This section defines the GPS Character Record. Each time that the device-dependent's **DdPutCharG** procedure is called, it receives one character from GPS Core via a GPS Character Record.

The following is a PLM definition of the GPS Character Record, and some of the character-attribute symbols used in it. Following the PLM definition are detailed explanations of the record's fields and subfields. This definition also occurs in the files **GpsDB.idf** and **structGps.h**, which are distributed with the PrintGen package.

NOTE: Some changes have been made to the Character-Record definition in GPS 2.0. The GPS 2.0 Character Record differs from the GPS 1.0 Character Record.

```
DECLARE TokenDDType LITERALLY 'STRUCTURE (
     wX                 WORD
    ,wY                 WORD
    ,bChar              BYTE
    ,bOverstrikeChar    BYTE
    ,iFont              WORD
    ,wSpaceWidth        WORD
    ,bAttr              BYTE
    ,wColor             WORD
    ,wBlkIdNext         WORD)'

    ,sTokenDDType       LITERALLY   '15'

    ,mskBold            LITERALLY   '1000$0000B'
    ,mskUnder           LITERALLY   '0100$0000B'
    ,mskDUnder          LITERALLY   '0010$0000B'
    ,MSKITALIC          LITERALLY   '0001$0000B'
    ,mskOStrike         LITERALLY   '0000$1000B'
    ,mskWhiteSpace      LITERALLY   '0000$0010B'

    ;
```

Fields of the GPS Character Record are defined as follows --

wX	X-position of this character or white-spaces' left edge (in GPS units).
wY	Y-position of this character or white-spaces' baseline (in GPS units).
bChar	Code of printer glyph (β) to be output, if an output glyph is associated with this token.
bOverstrikeChar	Relevant only if **mskOStrike** is set. If **mskOStrike** is set, then this is the printer glyph that **bChar** should be overstruck with. If **mskOStrike** is set and **bOverstrikeChar** = 0, then the device's default overstrike-glyph is to be used. The Character Record's font handle **(iFont)** specifies the font/CharacterSet for the overstrike printer glyph -- i.e., **iFont** applies to both **bChar** and **bOverstrikeChar**.
iFont	A font handle. This font-handle value specifies the font and CharacterSet of the printer glyphs in the Character Record. It is used in calls to the font-related GPS utilities. Font-handle values should never be modified by device-dependent code, and no meaning should be associated with any particular font-handle value. For a given font/CharacterSet combination, the font handle assigned to it will remain constant for an entire document. All font handles are redefined at the beginning of each document.
WSpaceWidth	Width (in GPS units) of the printer glyph (**bChar**) to be output, if the **mskWhiteSpace** attribute bit is not set. If the **mskWhiteSpace** attribute bit is set, then this field contains the width of the whitespace being defined. Width is in GPS units.
bAttr	A byte of attribute flags. These attribute bits specify the attributes associated with the printer glyph, and are described, separately, below.
wColor	Number of color that is to be used for this character.
wBlkIdNext	GPS internal use.

The attribute bits found in the **bAttr** field are defined as follows --

mskBold	When set, indicates that BOLD has been requested for this printer glyph, **and that a bold font is not available**. The DDP may opt to do something (such as increasing hammer energy) to cause the printer glyph to appear bold. When not set, indicates: 1) that BOLD has not been requested for this printer glyph; or 2) that BOLD was requested for this printer glyph and a bold font has been assigned to this printer glyph.
mskUnder	When set, indicates that UNDERLINING is in effect for this printer glyph or whitespace. When not set, indicates that UNDERLINING is not in effect for this printer glyph or whitespace.
MskDUnder	When set, indicates that DOUBLE-UNDERLINING is in effect for this printer glyph or whitespace. When not set, indicates that DOUBLE-UNDERLINING is not in effect for this printer glyph or whitespace. This bit is relevant only when **mskUnder** is set.
MSKITALIC	When set, indicates that ITALIC has been requested for this printer glyph, **and that a italic font is not available**. The DDP may opt to do something (such as tilting the glyph) to cause the printer glyph to appear italic. When not set, indicates: 1) that ITALIC has not been requested for this printer glyph; or 2) that ITALIC was requested for this printer glyph and an italic font has been assigned to this printer glyph.

MskOStrike When set, indicates that OVERSTRIKING is in effect for this printer glyph or whitespace. When not set, indicates that OVERSTRIKING is not in effect for this printer glyph or whitespace. What printer glyph will be used for overstriking depends upon the contents of the **bOverstrikeChar** field of this token.

MskWhiteSpace When set, indicates that this token defines a whitespace. The width of the whitespace is found in the **wSpaceWidth** field of this token. This token does NOT contain an output character (although it may include an "overstrike character"). The contents of **bChar** are undefined. The other bits in the **bAttr** field specify all of the attributes currently in effect.

See below for more information about whitespace.

Whitespace must be explicitly defined in a GPS document. Whitespace is a blank area that can be underlined and/or overstruck. Whitespace is defined two ways:

1) "Blank" characters output by a **WriteRecord** or **PlaceCharacter** (spaces and tabs).
2) Use of the **GPAMDefineWhiteSpace** operation.

If underscoring and/or overstriking of a blank area is desired, this blank area upon the page must be a whitespace area defined by one of the two methods described above. Blank areas that result from a Reposition alone cannot be underscored or overstruck.

There are no subscript or superscript attributes associated with any characters presented to the DDP of the GPS device driver. The GPS Core positions subscript or superscript characters appropriately, and passes them to the DDP of the GPS device driver with those positions specified in their GPS Character Record. Therefore, an underlined superscript, for example, may result in two GPS Character Records being sent to the DDP -- one containing the superscript character itself, and the other containing the underlined whitespace.

16.0 GPS DEVICE-SPECIFICATION PARAMETERS

A number of parameters that depend upon the particular output device are grouped together by GPS into one module. This module and its parameters are discussed here.

DdParams.asm is an example of this module. It is also the template for this module, and contains a number of definitions that need not and should not be changed by the device-driver writer. **DdParams.idf** is a PLM include file that may be used to define this module's declarations. **DdParams.h** is the C-language include file.

Each GPS device driver must have a Device-Specification Parameters module that has been customized for the particular device that is being controlled. This new module must be included in the linker's list of device-dependent object modules.

Note that many of the parameter variables are set to symbolic values within the Device-Specification Parameters file. Usually, a parameter variable within this module can be set by redefining the appropriate symbol in the Device-Specification Parameters module.

> **Not all of the parameters appearing in the Device-Specification Parameters module are described here. Those that are not described should not be changed in any new Device-Specification-Parameters module.**
>
> **Some changes have been made to the Device-Specifications-Parameters module definition in GPS 2.0. The GPS 2.0 Device-Specifications-Parameters module differs from the GPS 1.0 Device-Specifications-Parameters module.**
>
> **If a GPS 1.0 device driver is being converted to GPS 2.0, the GPS 2.0 DdParams.asm should be used as the basis for the driver's Device-Specifications-Parameters module. The existing parameters module should <u>not</u> just be modified in accordance with the following item descriptions! New parameters and values not described below are included in the GPS 2.0 version of DdParams.asm.**

Some of the following "fields" are described as "Symbols". These are assembler symbols (as defined by EQU directives). Anytime that any such "symbol" parameters are changed, the device driver's Device-Specifications-Parameters module must be re-assembled and linked before the change will take effect.

Field Name	Definition – Usage
sStackVI	(Symbol) Number of bytes to be allocated to the GPS device driver's VDM–Interpreter stack. Recommended value for a device that supports graphics is **2048**. Recommended value for a device that does not support graphics is **1**.
sVdmWA	(Symbol) Number of bytes to be allocated to the GPS device driver's VDM–Interpreter work area. Recommended value for a device that supports graphics is **2048**. Recommended value for a device that does not support graphics is **1**.
sDpStack	(Symbol) Number of bytes to be allocated to the stack of the GPS device driver's Data–Process stack. (A GPS device driver has two processes –– the Data Process and the Control Process.) The size of the Control–Process stack is set in the LINK command form when the driver is linked. The size of the Data–Process stack cannot be set through a LINK–form entry. The Data–Process stack size can only be changed by changing the **sDpStack** symbol in the Device–Specifications–Parameters module and re-assembling it. The Data Process is more sensitive to stack size than is the Control Process. All device–dependent code runs in the Data Process. The recommended starting value for **sDpStack** is **2048**.

DDsHeap	Contains number of bytes to be allocated for the GPS Core's "character heap". The device-driver Core uses this data area to sort and store Character Records before sending them to the DDP. The default value (4096) is recommended. When the end of a page is encountered, any Character Records still in this heap are passed to **DdPutCharG**. Anytime that another Character Record must be created and this heap is already full of Character Records, the first Character Record is output to **DdPutCharG** to make room for the new Character Record. Increasing the size of this heap allows more Character Records to be created and sorted before any are sent to the DDP. Recommended minimum value for this symbol is **2048**. Recommended value for laser-printer drivers is **4096**.
DDwWidth	Device's default character width (in GPS units). The default character width used by the GPS Core during character processing depends upon the character width of the current font.
DDwHeight	Device's default character height (in GPS units). This value is used as the point size when requesting the device's default font at the beginning of each document.
DDwSLDDistance	Device's default "line feed" distance. This value is used unless some other value has been specified in a **GPAMSetSLD** command.
DDyDUnderScore	This variable is included for optional use by the device-dependent code. It is intended to be used as the "offset from baseline" for placement of double underlines. For single-font devices, this value can be "tuned" from the debugger to obtain the desired offset. For multi-font device's, this value might be a factor to be scaled to get the correct offset for a given font size.
DDxTabWidth	Distance between horizontal tabs. This value is used if actual TAB codes are received via **GPAMWriteRecord** or **GPAMPlaceCharacter** calls.

DDwPgLength	Maximum page length for the device. Actual page length is specified at device-driver installation. See the PAGE ORIENTATIONS and DIMENSIONS subsection (page 21) for more information about page dimensions.
DDwPgWidth	Maximum page width for the device. Actual page width is specified at device-driver installation. See the PAGE ORIENTATIONS and DIMENSIONS subsection (page 21) for more information about page dimensions.
DDwLeftBorder	Defines the default "left border" width for the device. This value may be changed at device-driver installation time. See the PAGE ORIENTATIONS and DIMENSIONS section (page 21) for more information about page borders.
DDwRightBorder	Defines the default "right border" width for the device. This value may be changed at device-driver installation time. See the PAGE ORIENTATIONS and DIMENSIONS section (page 21) for more information about page borders.
DDwTopBorder	Defines the default "top border" width for the device. This value may be changed at device-driver installation time. See the PAGE ORIENTATIONS and DIMENSIONS section (page 21) for more information about page borders.
DDwBottomBorder	Defines the default "bottom border" width for the device. This value may be changed at device-driver installation time. See the PAGE ORIENTATIONS and DIMENSIONS section (page 21) for more information about page borders.

DDwDataPriority	Driver's default Data-Process priority. This value may be changed at device-driver installation. The Data Process is the driver process that does most of the work, and performs output. The default Data-Process priority should be such that it does not block interactive applications running on the same workstation. The number in this variable should never be less than 21.
DDfAlpha	Must be set to TRUE if output device supports textual output. Set to FALSE, otherwise.
DDfGraph	Must be set to TRUE if output device supports graphics output. Set to FALSE, otherwise.
DDnwSpeed	Device's speed in characters per second.
DDnwColors	Number of colors supported by this device. Set to zero if single-color device. This is configuration information, for use by the DDP. The Core does not use it.
DDcNulls DDbNull	Currently, several combinations of ByteStream code, processor-module type and device speed result in the last couple of characters being left in the serial I/O chip. So, any time that a port is being released, the character **DDbNull** is automatically output **DDcNulls** times. If the device is a strictly [lpt]-port device, **DDcNulls** may be set to 0, and none of these "flush" characters will be output. The value placed in **DDbNull** should be a byte treated as a null by the output device -- i.e., it should not result in any positioning of the output device, and it should not result in the printing of any glyphs.
bcNFonts	(Symbol) Specifies the number of entries that will be allocated in the "nominal fonts" table. The minimum value for this symbol is **3**. Its default value is **15**. Increasing its value may speed the processing of some multi-font documents.

bcTBMin	(Symbol) Specifies the Minimum number of heap blocks which must be reserved for font tables. The minimum value for this symbol is **4**. Increasing its value may speed the processing of some multi-font documents. If its value is increased, the default font-heap size may no longer be large enough. The size of the font-heap can be increased at device-driver installation.
DDsbFDevice	The device's default font-device type is placed into this sb string. This string is referenced in calls to the Font Service. It specifies for which type of output device the requested font data is obtained. The default font-device type may be overwritten at device-driver installation. If any string is placed into the device-driver installation form's FONT DEVICE field, that string will overwrite the string in **DDsbFDevice**. Therefore, **DDsbFDevice** must be defined to be a certain fixed length, rather than just the length of the device's default font-device-type string. The **DDsbFDevice** example in **DdParams.asm** is set up with the proper assembler directives to assure this.
DdDevOrient	This is a structure defined in **DdParams.asm**. It contains fields that define the device's page-orientation capabilities. Since it is a "structure", it should be included in each GPS device driver in the same order that it is in **DdParams.asm**. No **DdDevOrient** field can be left out, and the order of the fields should not be changed. See the <u>PAGE ORIENTATIONS and DIMENSIONS</u> section (page 21) for more information about page orientations. Descriptions of **DdDevOrient**'s fields follow:
bOrients	**(DdDevOrient** field) Specifies how many orientations the device is capable of. If it can print in both portrait and landscape page orientations, the value of this field will be 2. Symbol values defined in **DdParams.asm** are **L1ORIENTATION** and **L2ORIENTATIONS**.

BDefault	(**DdDevOrient** field) If **bOrients** = **L2ORIENTATIONS**, this field specifies the device's default page orientation. Symbol values defined in **DdParams.asm** are **LLANDSCAPE** and **LPORTRAIT**.
bPtoL	(**DdDevOrient** field) If **bOrients** = **L2ORIENTATIONS**, this field specifies the direction of text rotation when changing from portrait to landscape orientation. Symbol values defined in **DdParams.asm** are **LMINUS90** and **LPLUS90**. See the PAGE ORIENTATIONS and DIMENSIONS section (page 21) for more information about the use of this field.
fLFntDev	(**DdDevOrient** field) This field is reserved for future use, and must now be set to **FALSE**.

17.0 GPS 1.0 TO GPS 2.0 CONVERSION

This section contains guidelines for the conversion of customer-written GPS 1.0 device drivers to GPS 2.0 device drivers. It addresses varying levels of conversion. The levels vary from *patch-existing-driver-to-make-it-work*, to *modify-driver-to-add-GPS 2.0 capabilities.*

17.1 Patching Existing GPS 1.1 Drivers

Temporarily, it may be necessary to use a GPS 1.1 device driver with the GPS 2.0 software. Because of the new Request.9.sys file, this is not directly possible. The GPS 1.1 device driver run file must be patched with the Debug File utility so that it will work with the new request file.

In order to patch the GPS 1.1 driver, the symbol file created when the device driver was built must be available. In the example below, the name of the driver is LptFrobnitzDd. The characters in **bold face** indicate what the user typed in. Special characters are enclosed in angle brackets, and are capitalized, as in **<GO>**. If either of the values to be changed do not have the correct values (000B and 11D1), you should abort the Debug File command without making the changes, since you did not specify the correct symbol file.

```
Command             Debug File  <RETURN>
  File name         LptFrobnitzDd.Run  <RETURN>
  [Write?]          Yes  <GO>
  [Image mode?]

Debugger 10.3  (File Mode)
%'LptFrobnitzDd.Sym' <CODE-F>
%DDwVersionMax <RIGHT-ARROW> 000B  14 <RETURN>
%VpAdd+41E <RIGHT-ARROW> 11D1  0 <RETURN>
% <FINISH>
```

When these two changes have been made in the run file, the device driver should work with the GPS 2.0 request file. Documents should print, though any font changes will be ignored.

Note that this procedure is valid only for GPS 1.1 device drivers, and is not valid for GPS 1.0 drivers.

17.2 Changes Required for GPS 2.0 Link

A number of the new features and bug fixes of GPS 2.0 can be taken advantage of in the customer–written device driver by linking it with the new, GPS 2.0 Core routines. Such new features include some of the GPS 2.0 character–translation features. Before this linkage can be performed, several changes must be made to the GPS 1.1 device–dependent code. The following is a guide to the changes that must be made to GPS 1.1 device–dependent code before it can be linked with the GPS 2.0 Core routines.

New, required device–dependent–code procedures:

The DDP of each GPS 2.0 device driver must include two new procedures. These two procedures are the device–dependent procedures **DdBeforeConvertG** and **DdFirstChanceG**. These procedures are explained in the Required Device–Dependent Procedures section (page 34).

Core Utilities no longer available:

The **GetFontInfo** Core–utility should not be referenced by GPS 2.0 device drivers.

Changes in Char–Token structure:

Changes have been made to the Character Record passed to **DdPutCharG**. The GPS CHARACTER–RECORD section (page 83) describes font–related changes made to the Character Record. The Character Record's **font index** now has a format and interpretation which differs from those in a GPS 1.0 device driver. Unless the device–dependent code is modified to implement the GPS 2.0 version of fonts and character–translations, the DDP should not attempt to interpret or use the Character Record's **font index**.

Therefore, this "second level" of GPS 2.0 conversion is not practical for multi–font devices.

Device–Parameters changes:

The GPS DEVICE–SPECIFICATION PARAMETERS section (page 87) describes the Device–Specifications–Parameters module. Each GPS device driver must include this module. The GPS DEVICE–SPECIFICATION PARAMETERS section describes the required format and contents of a GPS 2.0 Device–Specifications–Parameters module.

Some variables have been removed from the GPS 2.0 Device–Specifications–Parameters module. Some have been added. The driver will not link if the driver's Device–Specifications–Parameters module is not of the described GPS 2.0 format. The driver will not install and run properly if that module does not contain the valid values listed in the GPS DEVICE–SPECIFICATION PARAMETERS section.

Device–Parameter Deletions:
DDsbSheetFeeder
DDrgbSheetFeeder

Device–Parameter Additions:
DDcNulls
DDbNulls
DDrgNFonts
DDbcNfonts
DDbcTBMin
DDsbFDevice

17.3 Adding GPS 2.0 Features

To fully convert a GPS 1.0 device driver to a GPS 2.0 device driver, implementation of the following new features must be considered:

1. Device–driver installation initializations.
2. Font and character translations.
3. Landscape and portrait page orientations.

Installation Initializations:

New options have been provided in GPS 2.0 for the initialization of device–dependent code when the device driver is being installed. These new options involve implementing device–dependent procedures **DdBeforeConvertG** and **DdFirstChanceG**. These procedures are explained in the Required Device–Dependent Procedures section (page 34).

Font and Character Translations:

The major new functionalities of variable fonts and character translation are described in the DEVICE-DRIVER FONT PROCESSING section (page 18). The Font-Related Utilities section (page 53) describes Core-utility routines which provide translation services to the device-dependent code. The GPS CHARACTER-RECORD section (page 83) describes font-related changes made to the Character Record passed to **DdPutCharG**. The Printing Guide describes what data should be entered into the Font Database, and how to do it.

Landscape and Portrait Page Orientations:

The PAGE ORIENTATIONS and DIMENSIONS section (page 21) describes the page-orientation options supported by GPS 2.0. A device driver that is to support rotated page orientations should examine the page-orientation field of the page-description record each time that **DdNewPageG** is called. The page-orientation field specifies the page orientation desired by the document creator.

The device's parameter module must be changed to signal support of multiple page orientations. See the description of **DdDevOrient** in the GPS DEVICE-SPECIFICATION PARAMETERS section (page 92).

Device-Parameters changes:

The GPS DEVICE-SPECIFICATION PARAMETERS section (page 87) describes the Device-Specifications-Parameters module. Each GPS device driver must include this module. The GPS DEVICE-SPECIFICATION PARAMETERS section describes the required format and contents of a GPS 2.0 Device-Specifications-Parameters module.

Some variables have been removed from the GPS 2.0 Device-Specifications-Parameters module. Some have been added. The driver will not link if the driver's Device-Specifications-Parameters module is not of the described GPS 2.0 format. The driver will not install and run properly if that module does not contain the valid values listed in the GPS DEVICE-SPECIFICATION PARAMETERS section.

Device-Parameter Deletions:
- **DDsbSheetFeeder**
- **DDrgbSheetFeeder**

Device-Parameter Additions:
- **DDcNulls**
- **DDbNulls**
- **DDrgNFonts**
- **DDbcNfonts**
- **DDbcTBMin**
- **DDsbFDevice**

18.0 IMPLEMENTATION LANGUAGES

Any of the CTOS-environment languages supported by Convergent Technologies can be used to write the DDP for a GPS device driver. The recommended programming languages are C and PASCAL.

The key requirement is conformance to the calling and parameter-passing protocol used by the GPS-Core routines. The protocol used by the GPS-Core routines is the same as that defined for accessing CTOS services in each of the programming-languages' reference manuals.

For languages that have that option, "medium model" must be used. This is particularly important for the workstation C language, as the parameters will be passes in the wrong order if "medium model" is not used.

Device I/O need not (and generally **should** not) be done from the device-dependent code.

19.0 LINKING A NEW GPS DRIVER

A new GPS device driver is linked by combining the new DDP object modules with the GPS device-driver Core object modules. PrintGen provides a special link command, Link GPS Driver, which performs this linkage.

This section describes the Link GPS Driver command. The command differs somewhat from that in GPS 1.0 releases.

When you use this command, your default path should be the path to your PrintGen directory.

Before you use the Link GPS Driver command, all of the device-dependent source modules must be compiled or assembled.

The Link GPS Driver command has the following form:

```
Link GPS Driver
    Device Dependent Object Files
    Device Driver Root Name
    Library Version Prefix
    [version]
    [Graphics ? (default = no)]
    [Text ? (default = yes)]
    [Stack Size (1024)]
    [Swapping ? (default = no)]
    [V6 runfile ? (default = no)]
```

Parameter-definitions for the Link GPS Driver command:

Device Dependent Object Files

Device-dependent object files. This string should name a file that contains the names of the device-dependent object files. These are the object files resulting from the compilation or assembly of the source files written for a specific output device. This parameter should **NOT** be the name(s) of actual object module file(s) -- it must the name of a text file **containing** the object filenames. Generally there will be at least two device-dependent object-files listed in the specified names file: 1) the device-specification parameter file, and 2) the device-dependent processing code.
This parameter must be present.

Device Driver Root Name

Root of the name that will be given to the device-driver runfile. The string DD.run will be appended to the device-driver name specified by this parameter. If the value of this parameter is **Epson**, then the resulting device driver will be named: **EpsonDD.run.**
This parameter must be present.

Library Version Prefix.

This string will have ≤ prefixed, DevDrBld≥ appended, and will specify the directory where the master build library is found.
This parameter must be present.

[version]

Version string that will be assigned to the device-driver runfile. If this parameter is left blank, the current date-time string will be used for the runfile's version string.

[Graphics ? (default = no)]

Allows the user to specify whether or not to include the device-driver Core's graphics modules. If this parameter is blank, the graphics modules are not included in the device-driver runfile -- if **any** characters are specified for this parameter (**including** "NO"), the graphics modules **are** included. If the driver's device does not support graphics, this parameter should be left blank to reduce the size of the device driver. But if the device does support graphics, this parameter should be set to **"YES"**, so that the required graphics routines will be included in the resultant device driver.

[Text ? (default = yes)]

This parameter should always be left blank when linking a GPS 2.0 device driver.

[Stack Size (1024)]

Stack size (in bytes) for the device-driver runfile. This parameter defaults to 1024. This is the Control-Process stack. (See the <u>**Driver-Process Stacks**</u> subsection, page 81, for information about a GPS device driver's various stacks.)

[Swapping ? (default = no)]

This parameter should always be left blank when linking a GPS 2.0 device driver.

[V6 runfile ? (default = no)]

This flag controls whether a Version-6 run file will be built. The Default is No. This parameter should be set to "**YES**", except when the driver must run on a system that does not support the Version-6 run file (e.g., SRP-CTOS-3.2). If the Version-6 run file is **not** selected, the string <u>m</u> will be prepended to the driver's run-file name. If the value of this parameter is blank and the Root-Name parameter is **Epson**, then the resulting device driver will be named: **mEpsonDD.run**.

The user is able to specify libraries that are to be referenced in the device-driver link. (**[Sys]<Sys>CTOS.lib** is always included, automatically.) The Link GPS Driver command will reference the file **link***<root>***.libs.fls**, where *<root>* is the Root-Name parameter. For example, if the Root-Name parameter is **Epson**, then the Link GPS Driver command will reference the file **linkEpson.libr.fls.** This file should contain the names of the libraries that are to be included in the device-driver link. Such a "libraries" file must be present for the Link GPS Driver command to function properly. If no libraries are required, then an empty "libraries" file must be created.

After the <u>Link GPS Driver</u> command has been successfully executed, the resultant runfile may be installed as a GPS device driver. The command has not been successfully executed if any errors have been listed upon the workstation display or in the resultant runfile map (the resultant runfile map is the file whose name is the driver name root with the string <u>.map</u> appended to it). The new device driver is installed by entering its runfile name in the <u>Driver Run File</u> field of Print Manager's device-installation form.

A submit file, **[Sys]<Sys>LinkGpsDriver.sub**, implements the Link GPS Driver command. This submit file causes the user-specified device-dependent object files and the required device-driver core object files to be linked, and in the correct order. Generally there will be at least two device-dependent object files -- the device-specification parameter file, and the device–dependent processing code. There is no reason why the device–dependent processing code cannot consist of multiple object–file modules. There are some device–driver core files that are always included, and other device–driver Core files that may or may not be included, depending upon whether or not graphics was requested.

It is not required that the Link GPS Driver command be used to link a new GPS device driver. The user may sometimes need to set up his own link command -- although the supplied command will be adequate in most of the situations. If a customized device–driver link command must be developed, use [Sys]**<Sys>LinkGpsDriver.sub** as a guide.

> **The DDSegOrder module from the GPS device–driver must be first in the list of modules to be linked.**

20.0 DEBUGGING AIDS

Debugging the device-dependent code of a GPS device driver is aided by the following features:

- Debugger entry at device-driver installation.
- **DdBeforeConvertG** DDP routine.
- **DdFirstChanceG** DDP routine.
- File output.

Debugger entry:

Debugger entry can be forced at the beginning of device-driver installation. Entering the debugger at this point enables one to examine data structures and to set breakpoints before the device driver is converted to a system service.

A one-byte variable in the device driver controls this debugger entry. This flag variable is **fBreak**. If it is set to **FALSE** (0), the debugger is not invoked at device-driver installation. By default, **fBreak** is **FALSE**. If **fBreak** is set to **TRUE** (0FFh), the debugger is invoked when the device driver begins its installation processing.

It is not necessary re-compile and re-link a device driver just to change **fBreak**'s value. The DEBUG FILE command can be used to change **fBreak**'s value in the device-driver runfile.

DdBeforeConvertG routine:

DdBeforeConvertG is a required DDP routine. It is called during device-driver installation, before the device driver is converted to a system service. It is called **after** all of the installation parameters have been obtained and processed by the core.

A breakpoint at this routine may aid the debugging of driver-initialization bugs. A breakpoint at this routine enables one to examine all of the device-driver installation data.

A detailed description of the **DdBeforeConvertG** routine is in the DEVICE-DEPENDENT FUNCTIONS section (page 34).

DdFirstChanceG routine:

DdFirstChanceG is a required DDP routine. It is called at the very beginning of device-driver installation (just after the invocation of the debugger, if **fBreak** is set).

A breakpoint at this routine may aid the debugging of driver-initialization bugs. This routine is called before the GPS Core has done its initialization processing, before the Data Process is created and before the device driver is converted to a system service.

A detailed description of the **DdFirstChanceG** routine is in the DEVICE-DEPENDENT FUNCTIONS section (page 36).

File output:

It is possible to output the printer-command data to a CTOS disk file, instead of to the printer port. Such a file can then be DUMPed to determine what exactly would have been sent to the printer. User-written debugging tools can easily read the data output to such files. ByteStream routines can be used to read these files.

Two things must be done to capture device-driver output on a CTOS file:

1) The device driver must be re-linked with a different SamGen file.
2) @ must be entered into the first character position of the Device Setup field in the device driver's installation form.

ByteStream support for disk files must be present for this file output to work. Such ByteStream support is not included in the usual device-driver run files, because it makes the driver larger. So a SamGen that does include the ByteStream disk-file support must be: 1) editted; 2) assembled; and 3) linked with the rest of the device-driver object modules. The new, resultant device driver is then capable of supporting disk-file output.

The new SamGen object module should replace the **SamgenDD** module specified in the list of device-driver link files. Assume, for example, that the SamGen module supporting disk ByteStreams is named **SamgenDDF.asm**. Then **SamgenDDF** would replace **SamgenDD** in the list of object files to link into the device driver.

The following is an example of such a SamGen specification that includes disk ByteStreams:

```
$INCLUDE (samgen.mdf)
%Init

%DeviceOpen([Lpt],OpenByteStreamLpt)
%DeviceOpen([Ptr],OpenByteStreamC)
%DeviceOpen([Disk],OpenByteStreamSD)

%tagProcs(tagDiskRead,FillBufferSD,FlushBufIllegal,CheckPointBsSD,
      ReleaseByteStreamSD,SetImageModeIllegal)
%tagProcs(tagDiskWrite,FillBufIllegal,FlushBufferSD,CheckPointBsSD,
      ReleaseByteStreamSD,SetImageModeIllegal)
%tagProcs(tagDiskModify,FillBufferSD,FlushBufferSD,CheckPointBsSD,
      ReleaseByteStreamSD,SetImageModeIllegal)
%tagProcs(tagLptWrite,FillBufIllegal,FlushBufferLpt,CheckPointBsLpt,
      ReleaseByteStreamLpt,SetImageModeLpt)
%tagProcs(tagPtrWrite,FillBufIllegal,FlushBufferC,CheckPointBsC,
      ReleaseByteStreamC,SetImageModeC)

%DevDepProc(GetBsLfa,GetBsLfaSync)
%DevDepProc(SetBsLfa,SetBsLfaSync)
%DevDepProc(SetImageMode,SetImageModeBrn)
%Final
```

If disk-file output is desired, @ must be in the first character position of the Device Setup field when the device driver is installed. Just having a SamGen that supports disk output will not cause all device-driver output to be sent to disk. The @ character specifies that disk output will be performed. If the device driver is installed without @ in Device Setup's first column, the driver will behave in the usual manner and send all output to its printer port.

Since the "disk output" switch (@) is specified in the device-driver's installation form, device-driver output can only be rerouted at device-driver installation. If disk output is desired, @ is placed into Device Setup's first column, and all subsequent print jobs will send all printer output to the disk file. To resume output to the printer, GPS must be removed and the device driver reinstalled without @ in the Device Setup field.

When file output is in effect, all printer output is written to **[Sys]<Gps>GPS.dmp**. Output cannot be directed to another file, but existing **[Sys]<Gps>GPS.dmp**'s can, of course, be renamed to other filenames between print jobs. More than one GPS device driver, sharing the same <Gps> directory, should not attempt concurrent file output.

21.0 EXAMPLES

Examples of the following GPS device drivers' DDP's are included in the PrintGen product:

- Daisy
- HPLaserJet
- Imagen8300
- LptSimple

For each of these example GPS device drivers, there is a brief description of its capabilities below, and a list of its source files. These source files are located in the Archive file on the [f0]<ct> directories of the PrintGen distribution diskettes. These files are restored from the distribution diskettes when the rest of the PrintGen files are restored. After PrintGen has been installed, the user should submit the **<2.0PrtGenBld>Compile_Examples.sub** file which will compile the device specific source files and create the four sample device drivers.

Daisy –

GPS daisy-wheel-device driver. Supports bold, strikethrough, underline, print-wheel mapping, multiple print-wheels, sheet-feeders, etc. Does not support graphics. Spoke-mapping and overstrike information that was formerly obtained from ".whl" files is now obtained from the Font Service. The font "alias" (obtained from the Font Database) is formatted in such a way that it specifies either a print wheel or a font cartridge. The alias may also contain command strings that are sent to the printer to switch to a different (cartridge) font. If the current font cartridge or print wheel contains the requested font, the user is not asked to load it again. These features facilitate the use of laser printers that emulate the Diablo630 printer.

Source files:

```
<2.0Daisy>Concat2.c
<2.0Daisy>Concat3.c
<2.0Daisy>Daisy.c
<2.0Daisy>Daisy_Data.c
<2.0Daisy>DaisyParams.asm
<2.0Daisy>DdFontChangeG.plm
<2.0Daisy>DdConfig.c
<2.0Daisy>DdSheet.c
<2.0Daisy>UlCmp.c
<2.0Daisy>Uc.c

<2.0DaisyBld>linkDaisy.fls
<2.0DaisyBld>linkDaisy.libr.fls
<2.0DaisyBld>linkDaisy.sub
```

<2.0GpsDef>CtosTypes.h
<2.0GpsDef>CompileOptions.idf
<2.0GpsDef>CtosLib.edf
<2.0GpsDef>CtosTypes.edf
<2.0GpsDef>daisy.h
<2.0GpsDef>DD.lit
<2.0GpsDef>DdFdef.h
<2.0GpsDef>DdParams.h
<2.0GpsDef>DdSam.idf
<2.0GpsDef>DdUtilities.idf
<2.0GpsDef>FontUtil.idf
<2.0GpsDef>gpserc.h
<2.0GpsDef>string.h
<2.0GpsDef>StructGPS.h
<2.0GpsDef>types.h

HPLaserJet –

GPS device driver for the HPLaserJet. Supports underlining, bold, overstrike, multiple fonts, etc. This version does not support graphics. The font "alias" (obtained from the Font Database) is formatted in such a way that it specifies the name of the cartridge containing the desired font, and the command string that must be sent to the printer to switch to that font. If the current font cartridge contains the requested font, the user is not asked to load it again.

Source files:

<2.0HP>HPParams.asm
<2.0HP>HPLaserJet.plm

<2.0HPBld>linkHPLaserJet.fls
<2.0HPBld>linkHPLaserJet.libr.fls
<2.0HPBld>linkHPLaserJet.sub

<2.0GpsDef>CompileOptions.idf
<2.0GpsDef>DdParams.idf
<2.0GpsDef>DdSam.idf
<2.0GpsDef>DdUtilities.idf
<2.0GpsDef>GpsDb.Idf
<2.0GpsDef>GpsErc.idf
<2.0GpsDef>Convert.idf
<2.0GpsDef>CtosLib.edf
<2.0GpsDef>CtosTypes.edf
<2.0GpsDef>DD.lit
<2.0GpsDef>DdVp.idf
<2.0GpsDef>FontUtil.idf
<2.0GpsDef>MinMax.idf
<2.0GpsDef>ulos.idf
<2.0GpsDef>Util.idf
<2.0GpsDef>XUtilDD.idf

Imagen8300 –

GPS device driver for text and graphics. Supports bold, strikethrough, underline, etc. It also supports both portrait and landscape page orientations. ImPress is used by the driver to control the Imagen printer. The Imagen8300 device driver uses the Font Database's Second-Level Translation to map from an 8-bit character code to the Imagen, 16-bit GASCII code. For each font used, the corresponding Second-Level Translation data for all characters is downloaded to the Imagen printer to create its mapping table. Since the DDP is receiving 8-bit character codes, and the Imagen can only accept 7-bit character codes, two Imagen "families" must be created for each font used. If the character code's high bit is set, the "upper" family of Imagen characters is selected before sending the lower seven bits of that character code to the Imagen printer.

Source files:

<2.0Imagen>DdImagen8300.asm
<2.0Imagen>DdImagenDriver.plm
<2.0Imagen>DdImagenFont.c

<2.0ImagenBld>linkImagen8300.fls
<2.0ImagenBld>linkimagen8300.libr.fls
<2.0ImagenBld>linkImagen8300.sub

<2.0GpsDef>CompileOptions.idf
<2.0GpsDef>CtosTypes.edf
<2.0GpsDef>ctostypes.h
<2.0GpsDef>DD.lit
<2.0GpsDef>DdChangePWheel.idf
<2.0GpsDef>DDParams.idf
<2.0GpsDef>DdSam.idf
<2.0GpsDef>DdUtilities.idf
<2.0GpsDef>FontUtil.h
<2.0GpsDef>GpsDB.idf
<2.0GpsDef>GpsErc.idf
<2.0GpsDef>UIOs.idf

LptSimple –

GPS device driver for simple ASCII, character printers. Does not support graphics, underlining, bold, etc.

Source files:

```
<2.0Lpt>LptParams.asm
<2.0Lpt>LptSimple.c

<2.0LptBld>linkLptSimple.fls
<2.0LptBld>linkLptSimple.libr.fls
<2.0LptBld>linkLptSimple.sub

<2.0GpsDef>CtosTypes.h
<2.0GpsDef>DdFDef.h
<2.0GpsDef>DDLits.h
<2.0GpsDef>DDParams.h
<2.0GpsDef>DdSam.h
<2.0GpsDef>DRLTypes.h
<2.0GpsDef>structGPS.h
<2.0GpsDef>DDVpData.h
<2.0GpsDef>DDFntLit.h
```

22.0 GPS DEVICE-DRIVER STATUS CODES

Some of the GPS and Font Service error codes are included in this PrintGen document. They do not replace the status codes published in the Status Codes manual and the GPS release notice. For some of the following status codes, more explanation is included in this document. The above two documents list such status codes as resulting from "internal errors". This PrintGen document lists DDP/Core interface errors that could cause these "internal errors".

22.1 GPS Status Codes

The set of GPS device-driver status codes include the following. They are listed in ascending order:

Code	Meaning
4531 to **4534**	Reserved for future use.
4535	Invalid "Hex" string. A hexadecimal string of byte values entered into the device-driver installation form is invalid because: 1. It contains non-hexadecimal digits. 2. It contains an odd number of hexadecimal digits. This erc is returned by the **DdHexToB** utility.
4536	GPAM data, or "*NORMAL* image-mode" data has been sent to the **BinaryMode** device driver that is provided with the Generic Print System. Only "*IMAGE* image-mode" data or "*BINARY* image-mode" data will be processed by the **BinaryMode** driver.
4537	An attempt was made to acquire a GPS device that is paused (a direct-print GPS device may be "paused" when it is not acquired to prevent its use by GPS). The device must be RESTARTed before it can be acquired for output.

4538 A **SetImageModeGPS** request sent to a GPS device driver while also sending it GPAM data. The image-mode of a GPS device driver cannot be set while it is processing a GPAM document. If a "pass-through" mode is required with GPAM data, consider using GPAM's **GPAMBegin-Transparent** and **GPAMEnd-Transparent** functions.

4539 More than one **SetGPSParams** request was sent to a GPS device driver for a single document. Or the **SetGPSParams** request was made after a **WriteGPSFile** request had already been made. The **SetGPSParams** request must occur between the **OpenGPSFile** request and the first **WriteGPSFile** request; and there cannot be more than one **SetGPSParams** request between these other two requests.

4540 The GPS device driver's output port is not currently acquired, and this request cannot be executed when the port is not acquired. If the device driver is installed for "direct" printing (not spooled), some other application may be currently printing to the device-driver's "shared" port.

4541 This device-control command cannot be executed now, because another device-control command has not yet completed execution. Wait a few seconds, and try the desired device-control command again.

4542 An ALIGN or RESTART command was attempted upon a device that is not paused. A GPS device must be paused before it will process an ALIGN or RESTART command.

4552 The specified restart location could not be found in the document. **(This is a change to the GPS 1.0 documentation of this error value.)**

4553 An ALIGN command was received by a GPS device that was not processing a document. **(This is a change to the GPS 1.0 documentation of this error value.)**

4561 This error should not occur in GPS 2.0 device drivers. Custom pre-2.0 GPS device drivers may report this error when processing data created by Document Designer 2.0.

4574 This error should not occur in GPS 2.0 device drivers.

22.2 Font Service Status Codes

The following are status codes used by the **Font Service**. They are listed in ascending order. (More GPS device–driver status code definitions follow this section.)

Code	Meaning
13900	**Font Service Not Running** – This is returned by Deinstall Font Service if there is no Font Service to deinstall.
13901	**Incorrect Version Font Database** – The Font Service returns this status code when it fails to install. The version of the runtime font database is incompatible with the version of the Font Service. Future releases of the font system will require that you regenerate the runtime font database, using the re–released Font Tool, to obtain a version of the font database compatible with the re–released Font Service. Verify that you have the correct version of font system software installed, and regenerate the runtime font database if necessary. It is also possible that the file provided to the Font Service is not a valid runtime font database at all.
13902	**Font Database Inconsistency** – The Font Service returns this status code when it discovers an internal inconsistency in the process of serving a request. If regenerating the font database does not clear up the problem, contact technical support.
13903	**Font Key Not Found** – The Font Service returns this status code when the reduced font key is not found in the font database. Verify that the regeneration of the runtime font does not result in errors (e.g., records flagged as invalid). If the problem persists, contact technical support.
13904	**Insufficient Font Service Buffer Space** – The Font Service returns this status code when it does not have sufficient buffer space to service the request. Reinstall the Font Service, specifying a larger amount of buffer space.
13905	**Insufficient Space to Return Font Info** – The Font Service returns this status code when the client of the Font Service calls it with less than minimal space to return the font data.

13906 **Cannot Install** – This is returned by the Font Service when it fails to install because the Font Service request codes are already being served by some service. Verify that the Font Service has not already been installed.
NOTE: Removing GPS does not remove the Font Service.

13907 **Cannot Deinstall** – This is returned by Deinstall Font Service under a single partition operating system.

13908 **Font Data Unavailable** – This is returned by the Font Service when the reduced font key does not refer to the type of data requested. For example, this status code is returned if a raster font is requested for a device type for which rasters are inapplicable or otherwise unavailable. Examine the returned font key. If the data should be available, check for an error in the editable font database (e.g., the raster font is not named in the font key, or the corresponding font file is not present in that path) and regenerate the font database.

13909 **Invalid Font Character Set Id** – The Font Service returns this status code when the request parameters contain a character set id which is generally invalid for the particular request (e.g., 0 is invalid on requests for the device alias), or invalid for the particular font key (e.g., 82 hex when the font key refers to only two character sets).

13910 **No Such Font Device Type** – GetFontDeviceList returns this status code when *iDevice* in the request is greater or equal to the total number of device types.

13911 **No Such Font Family** –GetFontFamilyList returns this status code when *iFamily* in the request is greater or equal to the total number of font families.

13912 **Invalid Font Key** – font key passed on Font Service request is invalid. For example, it may be of incorrect size.

13913 **Invalid Font Handle** – Handle passed on Font Service request does not match the handle for any item of the type (raster font or translation table) requested.

22.3 More GPS Status Codes

The set of GPS device–driver status codes include the following. They are listed in ascending order:

Code	Meaning
15300 – 15319	Reserved for use by the device–dependent portion of GPS device drivers. The meanings of these error codes will depend upon which GPS device driver is being used. **Refer to the device–driver descriptions for definition of these error codes.** Writers of device–dependent code may assign any needed error status codes from this range. Therefore, each code within this **15300 – 15319** range may have multiple meanings -- one for each of the device drivers that it is defined in. The Release Notice for each GPS device driver should include the definitions of the status codes in the **15300 – 15319** range for that particular driver. For example, the Convergent–supplied **Imagen8300DD.run** driver uses error codes in this range. They are as follows:

Imagen–Driver

Code	Meaning
15300	The document specifies more distinct fonts than can be output in one document by the Imagen8300 device driver. Reduce the number of fonts in the document.
15301	An erroneous 2nd–level–translation value has been encountered in the Font Database's font–translation data for the Imagen printer. Verify that the Imagen8300 device driver was installed with a valid **"Font Device Type"** string (leaving this field blank will cause the correct font data to be used if the Convergent–supplied Font Database is being used). Use the Font Tool to correct the Imagen's translation data in the Font Database.

15302 An erroneous 2nd-level-translation value has been encountered in the Font Database's font-translation data for the Imagen printer. Verify that the Imagen8300 device driver was installed with a valid **"Font Device Type"** string (leaving this field blank will cause the correct font data to be used if the Convergent-supplied Font Database is being used). Use the Font Tool to correct the Imagen's translation data in the Font Database.

15303 More Imagen-printer "Families" than can be used at one time, are required by this document. Reduce the number of fonts in the document.

15304 More Imagen-printer "Maps" than can be used at one time, are required by this document. Reduce the number of fonts in the document.

15320 The Font Device Type specified by the GPS device driver does not exist in the font database accessed by the device driver. All GPS device drivers have a default Font Device Type. A different Font Device Type may be specified at installation time. Each installed GPS device driver must have access to an installed font database that contains font data designed for that device driver.
The Font Device Type that will be used by default is the "sb" string in **DDsbFDevice** (in the Device-Specifications-Parameters module). Since this is an "sb" string, the first byte must be a count of the following characters that specify the default Font Device Type. A terminating null byte is not required, and should not be included in string's byte count.

15321 This is an internal error that indicates a problem in the software you are using. Consult Technical Support.

15322	The GPS device driver has detected a bad "font handle". It could be an internal Core error, but most likely it is caused by an error in the device-dependent code. A call by the DDP to one of the GPS font utilities included an invalid font handle as one of its parameters. Device-dependent code must not modify or attempt to generate font handles. Also, font-handle values cannot be saved from one document to the next.
15323 15324	The GPS device driver's "actual-font" information tables have overflowed. This is an internal error that indicates a problem in the software you are using. Consult Technical Support.
15325	An invalid "font handle" was present in a call to **DdFntHandles** within the GPS device driver. Most probably a call by the DDP to **DdFntHandles** included an invalid font handle as one of its parameters. Device-dependent code must not modify or attempt to generate font handles. Also, font-handle values cannot be saved from one document to the next.
15326	The recipient data structure specified in a call to **DdFntKey** within the GPS device driver was too short. The data structure pointed to by **pFontKey** must be at least 70 bytes for GPS 2.0.
15327	The alias of a document character set was requested. Only native, device character sets have aliases. An alias cannot be requested for Character-Set 0. The first Character-Set for which an alias may be requested is Character-Set 80h.
15328	An invalid "font handle" was present in a call to **DdGetAlias** within the GPS device driver. Most probably, a call by the DDP to **DdGetAlias** included an invalid font handle as one of its parameters. Device-dependent code must not modify or attempt to generate font handles. Also, font-handle values cannot be saved from one document to the next.
15329	Reserved for future use.

15330	An invalid "font handle" was present in a call to **DdGetChSet** within the GPS device driver. Most probably, a call by the DDP to **DdGetChSet** included an invalid font handle as one of its parameters. Device-dependent code must not modify or attempt to generate font handles. Also, font-handle values cannot be saved from one document to the next.
15331	The recipient data structure specified in a call to **DdGetAlias** within the GPS device driver was too short. The data structure pointed to by **prgbAlias** must be at least 41 bytes in GPS 2.0.
15332	The recipient data structure specified in a call to **DdChXlate** within the GPS device driver was too short. The data structure pointed to by **prgbXltn** must be large enough to contain the longest of the Second-Level Translations specified for this device type.
15333	This is an internal error that indicates a problem in the software you are using. Consult Technical Support.
15334	Insufficient "Font Buffer" space was allocated when installing the driver. The default installation value (4KB) is adequate for all Convergent-supplied GPS device drivers, but some custom GPS device drivers may require more. 2KB is the minimum allowed. One cause of this is a large number of very complex Second-Level Translations. This requires more memory for the GPS Core to access and process the translation data.
15335 - 15339	This is an internal error indicating a problem in the software you are using. Consult Technical Support.
15340 15341	May indicate an error in the translation escape sequences in the font database data being accessed by this driver. Verify that the rules stated in the FONT ESCAPE SEQUENCES section (page 74) are being followed in the First-Level Translations and Second-Level Translations specified for this device type.

15342 An invalid "2nd-level translation" was detected by the **WriteDdXlateByte** utility. This could be the result of several things:

1. Erroneous translation specification in the accessed font database.
2. Font handle for the wrong character set passed to **WriteDdXlateByte.**
3. Inappropriate character value passed to **WriteDdXlateByte.**

Verify that the rules stated in the FONT ESCAPE SEQUENCES section (page 74) are being followed in the First-Level Translations and Second-Level Translations specified for this device type.

15343 – 15345 This is an internal error indicating a problem in the software you are using. Consult Technical Support.

15346 An invalid "font handle" was present in a call to **DdChWidth** within the GPS device driver. Most probably, a call by the DDP to **DdChWidth** included an invalid font handle as one of its parameters. Device-dependent code must not modify or attempt to generate font handles. Also, font-handle values cannot be saved from one document to the next.

15347 An invalid "font handle" was present in a call to **DdFntKey** within the GPS device driver. Most probably, a call by the DDP to **DdFntKey** included an invalid font handle as one of its parameters. Device-dependent code must not modify or attempt to generate font handles. Also, font-handle values cannot be saved from one document to the next.

15348 – 15353 This is an internal error indicating a problem in the software you are using. Consult Technical Support.

15354 An invalid "font handle" was present in a call to **DdChXlate** within the GPS device driver. Most probably, a call by the DDP to **DdChXlate** included an invalid font handle as one of its parameters. Device-dependent code must not modify or attempt to generate font handles. Also, font-handle values cannot be saved from one document to the next.

15412 15413 For GPS 2.0, the **fLFntDev** field of the **DdDevOrient** structure in the Device-Specifications-Parameters module must be set to **FALSE** to prevent these errors at device-driver installation.

15414 An invalid "Page-Portrait String" was specified in the "Device Setup" field of the installation form. Page-Portrait strings must contain only hexadecimal digits, are limited to 20 digits, and there must be an even number of digits.

15415 An invalid "Page-Landscape String" was specified in the "Device Setup" field of the installation form. Page-Landscape strings must contain only hexadecimal digits, are limited to 20 digits, and there must be an even number of digits.

15416 An invalid "Reset String" was specified in the "Device Setup" field of the installation form. Reset strings must contain only hexadecimal digits, and there must be an even number of digits.

15417 Indicates inconsistencies in the **DdDevOrient** structure in the Device-Specifications-Parameters module. Error is caused by one or more of the following inconsistencies:

1. The **bOrients** field contains value other than 1 or 2.
2. The **bOrients** field equals 2, and the **bPtoL** field contains a value other than 1 or 2.
3. The **bOrients** field equals 2, and the **bDefault** field contains a value other than 1 or 2.
4. The **fLFntDev** field not equal to **FALSE**.
5. The **bOrients** field equals 2, and the **bDefault** field's value does not agree with the orientation implied by **DDwPgWidth** and **DDwPgLength**.

15418 The GPS 2.0 loadable requests have not been installed. The system must be rebooted after installing the GPS 2.0 loadable request file.

15419 The version of the installation parameters data and of the GPS device driver is incompatible. Verify that the GPS 2.0 Installer (GpsInstall.run) is being used. Consider deleting the existing **.state** and **.config** files for the affected device before repeating the installation process.