# Plug and Play Parallel Port Devices

**Version 1.0b**
**March 15, 1996**

## Revision History

| Issue Date | Comments |
|---|---|
| | Preliminary draft for printers only |
| February 11, 1994 | Expanded to multiple classes of parallel port devices, eliminated automatic IDs, added example of device IDs and compatible devices Ids |
| September 9, 1994 | Clarify description of how ID is generated, answer common questions asked on 1.0 version, discuss impact of compatible ID's |
| March 15, 1996 | Added CLASS key values for scanners and digital cameras. |

# Table of Contents

## Executive Summary

This document outlines the minimum hardware & software requirements for parallel port devices to obtain "Plug & Play" support under Chicago. "Plug & Play" (in this document) is defined as the ability to attach an I/O device to a host, and enable the host to determine that the device has been added, identify it, and either automatically install the necessary device drivers or prompt the user for a diskette containing them. This document is not intended to provide a complete, detailed description of how an operating system will internally accomplish this, but rather to define the necessary hardware/software requirements.

## Host Requirements

Plug & play support for parallel devices requires a minimum of a parallel port capable of supporting nibble mode as defined in IEEE-P1284.

It is suggested that hosts support ECP mode as defined by the IEEE 1284 specification in order to enjoy the improved performance benefits ECP mode provides. Additional information is available from Microsoft on implementing ECP host support.

## Device Requirements

Requirements for attached I/O devices can be divided into 3 areas, hardware, software, and device identification. Each of those areas is discussed in the following sections.

### Hardware Requirements

Plug & Play support for parallel port devices will require that the device provide a parallel interface, which conforms to the IEEE 1284 specification. This interface may be 1284-I compliant (using 1284-A or 1284-B connectors), or 1284-II compliant (using 1284-C connectors).

It is also required that these devices minimally support nibble mode as described in the IEEE 1284 specification.

It is suggested that parallel port devices support ECP mode as defined by the IEEE 1284 specification in order to enjoy the improved performance benefits ECP mode provides.

Note that in order for a peripheral to receive "Plug & Play" certification from Microsoft, nibble mode support & reporting the minimal device ID key values, (as described in IEEE 1284), is required.

### Software Requirements

Plug & play support for parallel port devices will require that the device return a device ID string in nibble mode, as defined by the IEEE 1284 specification.

The details of how the device ID should be returned are described below, minimal Plug & Play support is available to devices returning the required device ID fields as documented by the IEEE 1284 specification (described below), however Microsoft has defined 3 additional keywords which provide more optimal Plug & Play support. Vendors are strongly encouraged to support these 3 additional fields.

Note that Plug & Play support for devices only reporting the 3 required key values as described in IEEE 1284 is possible, supporting the 3 additional key values will provide the following:

- Allows devices to specify their class and activate class-specific installation procedures
- Allows the device itself to return a "user-friendly" name describing the device, which will be used during installation UI if no entry is found for the device in an INF file..
- Allows the device to report other devices that it is compatible with. Devices which only support the minimal 1284 device ID key values will not be able to have a compatible driver selected and installed automatically.

## Device Identification

### General Requirements

The following requirements apply to both the "Minimum Requirements" and "Optimal Requirements" sections below.

- All strings must be composed of ASCII values between 32-127 (20h-7Fh), inclusive.
- All MANUFACTURER, MODEL, CLASS, and COMPATIBLE ID key values must remain static for any device. For a specific hardware configuration of a device, the values of these fields should not change regardless of how NVRAM settings have been configured. However, if additional hardware is installed which requires different device drivers, the device ID keys can differ from what has been originally returned by the device. See the section below for a discussion of multiple “personality” devices.
- All MANUFACTURER, and MODEL key values must **remain unique from each manufacturer**.

### Minimum Requirements

The minimum requirements for reporting device ID's are as described in Section 6.6 of the IEEE 1284 specification. Specifically, these are (case sensitive): **MANUFACTURER**, **COMMAND SET**, and **MODEL**. (The abbreviations: **MFG**, **CMD**, and **MDL** are acceptable, as described in the IEEE 1284 specification). The COMMAND SET field will be ignored under Chicago.

A unique Plug & Play ID will be generated for the device using the following algorithm: The MANUFACTURER & MODEL fields will be concatenated, and then a 4 digit checksum will be generated on the resulting string. After the check sum is computed the string will be truncated to 20 characters if its length exceeds 20 characters. The checksum will then be converted to a 4 character string, and appended to the above truncated string, resulting in a string of 24 characters or less.

A code fragment that illustrates how the checksum & device ID values are generated is listed at the end of this document.

For example, the following device ID string will be generated for the HP LaserJet 4P printer:

**“Hewlett-PackardHP_La7EE2”**

since the MFG field is Hewlett-Packard, and the MDL field is HP LaserJet 4P.

The parallel port enumerator will then prepend "LPTENUM\" (minus quotes) to the string generated above, and the resulting string will be the Plug and Play ID value. The Plug & Play ID will be used to search all known device information files (INF in Windows) available on the host for a driver which will support the device. Note the ordering of device ID values listed in the INF files will be significant in determining the device driver which is the "best match" for a given device. See the section below on "Ordering Device ID values" for a more detailed description.

## Optimal Requirements

The recommended requirements for reporting device ID's are as described in the previous section, with 3 additional keywords defined: **CLASS, DESCRIPTION** and **COMPATIBLE ID** (which may be abbreviated as **CLS, DES,** and **CID** respectively). Note these values are not case sensitive.

- **CLASS** should have one of the following values: **PRINTER, MODEM, NET, HDC**[1]**, PCMCIA, MEDIA**[2]**, FDC**[3]**, PORTS, SCANNER, DIGCAM**

- **DESCRIPTION** key is an ASCII string that provides the description of the device the manufacturer would like presented to the user if an INF file is not found. The **DESCRIPTION** key should be no longer than 128 characters.

- **COMPATIBLE ID** key may have any value that exactly matches an ID value listed in an INF file, as described in the previous section. Additionally there may be Plug & Play IDs assigned to legacy parallel port devices which have driver support in Chicago. These IDs may also be specified in the CID field. If the device is not compatible with another device, or the manufacturer does not wish to have a compatible device driver used, there is no need to support this keyword.

## Additional Information on Compatible ID values

The values returned for COMPATIBLE ID should be carefully chosen. If a device returns a compatible ID which is an **exact** match for a device ID for another vendors device, sufficient care should be taken to ensure that the manufacturer of the device which is returning the compatible ID has license rights to install the device driver from the vendor providing the device driver. Microsoft will supply values for COMPATIBLE ID values for many of the printer drivers shipped with Chicago, and vendors are encouraged to freely use these values for COMPATIBLE ID keywords. Additional information on the exact syntax for COMPATIBLE ID values for specific printer models will be provided in the Chicago Device Driver Kit (DDK).

The syntax of the string returned from the device as the COMPATIBLE ID value will be slightly different depending on exactly what a given device is reporting itself as being compatible with. More specifically, if the device is reporting a compatible ID for the exact device ID of another physical device, it will need to include the enumerator name as part of the Compatible ID field. Otherwise, one of the compatible ID values documented in the Chicago Device Driver Kit may be used, which will not require the enumerator name. An example will illustrate this best. Suppose the INF file contains the following entry:

"Sample Printer"
=X.DRV,LPTENUM\Sample_Printer_CompaAAAA

[1]HDC - Hard Disk Controller
[2]Media refers to any multimedia device
[3]FDC - Floppy Disk Controller

The example above is a listing a printer driver which displays the printer name “Sample Printer”, which is supported by X.DRV, and the Plug and Play ID generated from the P1284 keywords (as described later in this paper) is “Sample_Printer_CompaAAAA”. As described earlier, the parallel port enumerator will prepend “LPTENUM\” to this, and the resulting value is what is listed in the INF file.

If another device wished to report itself as compatible with this device, it would need to report its CID keyword as (minus quotes) “LPTENUM\Sample_Printer_CompaAAAA”. Alternatively, it could report its CID keyword as (minus quotes) “Sample_Printer_CompaAAAA” only if the INF file contained the line below, duplicating the original ID without the LPTENUM reference:

“Sample Printer”
=X.DRV,LPTENUM\Sample_Printer_CompaAAAA, Sample_Printer_CompaAAAA

Note that the very first time Chicago is started on a machine, the way in which compatible ID values are processed is handled slightly differently than subsequent sessions. See the section below on “Ordering Device ID values” for more information on compatible ID’s.

A printer which is compatible with the HP LaserJet 4L, and has the necessary rights to use a driver provided by Hewlett-Packard, will return the following string in its compatible ID field:

**“CID:LPTENUM\Hewlett-PackardLaserC029”**

A printer which is compatible with the HP LaserJet 4L, and wishes to report the compatible ID for the driver provided by Microsoft will return the following string in its compatible ID field:

**“CID:HP_LaserJet_4L”**

## INF file Requirements

The information described above covers what devices should be reporting back as device identification values. Additional information on the INF file format for Plug & Play devices and specific classes is included in the Chicago Device Driver Kit (DDK)

## Multiple Emulation (“Personality”) Device Requirements

It is becoming increasingly common for printers to support multiple Page Description Languages, (such as HP-PCL & PostScript) which require separate printer drivers under Windows, or to even for a given device to be multi-function (printer+fax, printer+SCSI drive, etc.). Plug and Play supports the notion of mapping a single device ID for a given device to a single device driver, but not the notion of mapping a ***single*** device ID to multiple device drivers. However, Plug and Play does provide solutions to address this issue, on either the enumerator or class installer level.

**Multiple device drivers installed via a bus enumerator**
Certain bus enumerators, such as serial ports, provide a mechanism for the device to report multiple device ID values, each of which maps to a specific device driver. The parallel port bus enumerator does not support the ability to enumerate multiple device ID values. Please refer to the “Plug and Play COM Device” (currently Draft) Specification for additional information on enumerating ID values for a multi-functional device over a serial port.

**Multiple device drivers installed via a class installer**

Certain class installers, such as the printer class installer under Chicago, provide a mechanism for device vendors to supply their own DLLs which may install device driver support via one of two possible methods. The first method allows the printer class installer provided with Chicago to carry out installation of 1 printer driver (the one which is mapped to a particular printer driver via the Plug and Play ID in an INF file), and then call into a vendor supplied 32 bit DLL at the end of the printer installation process which may then choose to either install additional printer driver(s), or call other class installers to install additional device drivers as needed. The second method allows a vendor supplied DLL to perform all of the functionality required to install one or more device driver by allowing the printer class installed supplied with Chicago to copy files from an installation diskette & then the printer class installer will call directly into the vendor supplied DLL to perform the rest of the installation process. In both cases, information on the DLL & the entry point to call is provided by the INF file. Additional information on this will be provided in the Chicago Device Driver Kit (DDK).

## Ordering of Device ID & Compatible Device ID values

As described earlier, the compatible device ID provides the IDs of other devices with which this device is compatible. The operating system uses this information to load compatible device drivers if necessary. There can be several compatible device identifiers for each logical device. The order in which these device IDs are listed in an INF file may be used by the operating system as a criteria for determining which driver should be searched for and loaded first. The information below discussed the impact of ordering device ID values under Chicago.

The ordering scheme assigns a numeric “rank” value to each ID value, and the lower the rank order the better match a given device driver is considered to be for a given device. A rank order of 0 is determined to be an exact match and when a rank order match of 0 is found the device driver will automatically be installed without prompting the user. If a rank order match of 0 is not found, but a higher order rank match is, the user will be informed that a compatible device driver is available, and the user may either provide a diskette containing the device driver (ideally a rank 0 matching driver) for the device, or choose the most compatible driver as desired. If the user selects the compatible driver, the “best” match (lowest rank order matching) driver will be installed.

Rank ordering is determined as follows:

1) Taking the list of all ID values returned by the device, and assign a rank value to them. The device ID build from the MFG and MDL keywords is assigned rank 0, the first CID listed (if present) is assigned rank 1, the 2nd (if present) CID is assigned a rank of 2, etc.

2) Search the contents of the INF files on the host system for matches with any of the ID values returned by the device.

3) In cases where an exact match is found, create an “INF rank value”, (which is determined by the order that the ID is found in the list after a particular driver, the 1st ID listed in the INF after a particular driver is assigned INF rank of 0, the ID following the 1st (for the same driver) ID is assigned an INF rank of 1, the next (3rd) ID after that is assigned INF rank of 2 etc.), and then add the INF rank value to the rank value of the ID from the device.

4) Once all INF files have been searched, and all rank values have be determined, if a rank 0 match is found, automatically install the driver without prompting the user. If no rank 0 matches exist, prompt the user if they wish to supply a diskette with a driver for the device, or the user can choose to install the default, compatible, device driver.

Two examples of how this would work are provided below:

**Example #1:**

INF file contains:

"Sample Printer 1" =X1.DRV,LPTENUM\Sample_Printer_CompaAAAA, Sample_Printer_CompaBBBB
"Sample Printer 2" =X2.DRV,LPTENUM\Sample_Printer_CompaCCCC, LPTENUM\Sample_Printer_CompaDDDD, Sample_Printer_CompaEEEE

Device returns the following:

> Device ID (built as described elsewhere in this paper)="LPTENUM\Sample_Printer_CompaCCCC"
> Compatible ID="CID:LPTENUM\ Sample_Printer_CompaAAAA, Sample_Printer_CompaBBBB"

In this example, the device returns a device ID which is found to have a rank 0 match for "Sample Printer 2" (X2.DRV), and that driver is installed automatically without prompting the user (except to supply diskettes that contain required files). The other ID values are essentially ignored since a rank 0 match was found.

**Example #2:**

INF file contains:

"Sample Printer 1" =X1.DRV,LPTENUM\Sample_Printer_CompaAAAA, Sample_Printer_CompaBBBB
"Sample Printer 2" =X2.DRV,LPTENUM\Sample_Printer_CompaCCCC, LPTENUM\Sample_Printer_CompaDDDD, Sample_Printer_CompaEEEE
"Sample Printer 3" =X3.DRV,LPTENUM\Sample_Printer_CompaFFFF, LPTENUM\Sample_Printer_CompaGGGG, Sample_Printer_CompaHHHH

Device returns the following:

> Device ID (built as described elsewhere in this paper)="LPTENUM\Sample_Printer_CompaDDDD"
> Compatible ID="CID:LPTENUM\ Sample_Printer_CompaHHHH, Sample_Printer_CompaBBBB"

In this example, the device returns a device ID which is not found to have a rank 0 match with any ID values in INF files on the host system. After calculating all of the rank assignments, "Sample Printer 2" (X2.DRV) is determined to have a rank value of 1 (and hence the best match), "Sample Printer 3" (X3.DRV) is determined to have a rank value of 3, and "Sample Printer 1" (X1.DRV) is also determined to have a rank value of 3. The user will then be prompted that an exact match has not been found, but that compatible drivers are available, and offer the opportunity to provide a diskette supplied by the vendor, or for the user to accept the default driver. If the user choose to accept the default driver, the lowest rank order matching driver will be used.

The only variation on the behavior described above is the very first time Chicago is started on a system. In this one scenario, the best rank matched compatible device driver will be installed automatically without prompting the user.

## Algorithm for building Plug and Play device ID values for parallel devices

The following code fragment illustrates how parallel devices will have device ID values generated based on the MFG and MDL keywords under Chicago.

```
#define MAX_DEVNODE_NAME_ROOT        20
#define _16_BIT_CHECKSUM                    5

WORD    wCRC16a[16]={
    0000000,    0140301,    0140601,    0000500,
    0141401,    0001700,    0001200,    0141101,
    0143001,    0003300,    0003600,    0143501,
    0002400,    0142701,    0142201,    0002100,
};

WORD    wCRC16b[16]={
    0000000,    0146001,    0154001,    0012000,
    0170001,    0036000,    0024000,    0162001,
    0120001,    0066000,    0074000,    0132001,
    0050000,    0116001,    0104001,    0043000,
};

void BuildP1284PnPID
(
    void
)
{
    WORD    wCheckSum;
    char    szCheckSum[_16_BIT_CHECKSUM];
    WORD    wTempSize;

    LPSTR   lpMFG = NULL;
    LPSTR   lpMDL = NULL;

    // The following are used to generate checksum values
    PBYTE   pb;
    BYTE    bTmp;
    ULONG   ulSize, ulSeed;
    PULONG  pulSeed;

    // following fills in the lpMDL & lpMFG values as returned from device
    FindP1284Keys(&lpMFG, &lpMDL);

    // N:  Concatenate the MFG and MDL fields
    wTempSize = (WORD) lstrlen(lpMFG) + (WORD) lstrlen(lpMDL);

    // Concatenate the provided MFG and MDL P1284 fields
    lstrcpy(szTemp, lpMFG);
    lstrcat(szTemp, lpMDL);
    ulSize=lstrlen(szTemp);

    // Checksum the string
    // start a seed at 0, and create a pointer to the seed.  After the
    // checksum, convert the checksum to a word.
    ulSeed = 0;
    pulSeed = &ulSeed;

    for (pb=(BYTE *)szTemp; ulSize; ulSize--, pb++)
    {
        bTmp=(BYTE)(((WORD)*pb)^((WORD)*pulSeed));      // Xor CRC with new char
        *pulSeed=((*pulSeed)>>8) ^ wCRC16a[bTmp&0x0F] ^ wCRC16b[bTmp>>4];
    }
    wCheckSum=(WORD)(*pulSeed);
    ConvertHexWORDToString(szCheckSum, wCheckSum);

    // Form the Dev Node ID.
    if (wTempSize > MAX_DEVNODE_NAME_ROOT)
        szTemp[MAX_DEVNODE_NAME_ROOT] = '\0';

    // Got to remove spaces from lpTemp, Just replace them with Underscores
    StringSubst ( szTemp, ' ', '_',
                  wTempSize > MAX_DEVNODE_NAME_ROOT ? MAX_DEVNODE_NAME_ROOT : wTempSize);
```

```
    lstrcpy(szDevNodeName, LPTENUM);
    lstrcat(szDevNodeName, szTemp);
    lstrcat(szDevNodeName, szCheckSum);
}
```