# *SBP-2 Printing under existing consumer PC operating systems*

1394 PWG White papers
Rev. 02b, May 13, 1998

*Reported by*
*Fumio Nagasaka (Epson Software Development Laboratory Inc.,)*

# *Table of Contents*

## *Table of Figures*

# 1. Overview

## 1.1. Scope

1394 PWG is continuing effort for standardization of IEEE 1394 Printing Protocol. One of possible solution which is available under existing consumer PC operating systems is SBP-2 printing. According to this idea, a PC shall act as SBP-2 initiator and printers shall be SBP-2 target devices.

Due to limitation of SBP-2 facilities, printing session between a host device and a printer does not provide symmetrical service. Asymmetry of service is uncomfortable for peer to peer printing (or so-called “Direct-printing”) applications. However other PC peripherals including storage devices and image scanners tend to support SBP-2 protocol for IEEE 1394 platform. Thus, several consumer PC operating system will provide SBP-2 initiator capability.

To use SBP-2 printers under current consumer PC operating system, this document describes limited functionality that is provided in current PC operating systems.

## 1.2. Goal

The SBP-2 Printing Protocol standardization through 1394 PWG shall keep upward compatibility with existing SBP-2 driver stack. If users are using SBP-2 printer on current PC OS platform, users may use same SBP-2 printer model after 1394 PWG protocol had been issued.

This document tries to describe mandatory requirements to cover SBP-2 printing now and future.

## 1.3. Purpose

Purpose of this document are listed below:

- To make it possible to print using a printer which has SBP-2 target capabilities, under existing consumer PC operating systems including Macintosh OS8.x and Windows NT5.0.
- To support Plug and Play functionality for SBP-2 target device.
- To provide multiple host connectivity
- To support multiple target devices access capability
- To keep upward compatibility between current SBP-2 printing and future SBP-2 printing

## 2. Requirements for SBP-2 printing

- Bi-directional communication between PCs and printers.
- Unordered execution of SBP-2 ORBs provided in an initiator's memory space.
- Multiple host devices support capability.
- Multiple target devices connectivity.
- Fare access both from targets and initiators.
- Status service while printing

# 3. Normative References

## 3.1. Approved references

ISO/IEC 13213:1994, Control and Status Register (CSR) Architecture for Microcomputer Buses
IEEE Std. 1394-1995, Standard for a High Performance Serial Bus

References under development
Serial Bus Protocol 2 (SBP-2) Working Draft, X3T10 Project 1155D Rev.3d, March 21, 1998
< ftp://ftp.symbios.com/pub/standards/io/t10/drafts/sbp2/>

Microsoft Plug and Play Design Specification for IEEE 1394 Ver. 1.0b, October 17, 1997
Microsoft Plug and Play Parallel Port Devices Ver. 1.0b March 15, 1996

## 3.2. WDM Specific terminology

### *3.2.1. Connection devices ID*

An unique ID given by the SBP-2 target devices on IEEE1394 high performance serial bus.
Each Logical unit of a SBP-2 target shall have at lease one textual descriptor (1284 PnP ID + GUID ) in it's IEEE 1212 Configuration ROM. The textual descriptor shall be contained in *Unit_Leaf* -> *Unit_Dependent_Leaf* -> *Textual_Descriptor*.
Also Operating System of the SBP-2 initiator device may use "node_vender_ID (24bits)" and "Chip ID (40bits)" to generate unique reference number for the SBP-2 target.

### *3.2.2. Device handle names*

The name which SBP-2 Port driver create. Any user mode application may invoke CreateFile() function with this device handle name, to open access path to the specified IEEE 1394 device.
An IEEE1394 Filter driver including SBP-2 Port driver gives device handle name for each FDO (Function Device Object). I/O Manager is responsible to allow user mode application to access FDO, utilizing device handle name.

### *3.2.3. printer port names*

The printer port name given by the port monitor application. Any user application may access to the printer utilizing printer port name. User may review printer port names using "addition WIZARD of printer" which has graphical user interface.

# 4. Windows NT5.0 as existing Operating System

## 4.1. Overview

Windows NT5.0 is proving IEEE 1394 printer through SBP-2 Port driver. This means any 1394 printer which is applicable under NT5.0 is required to perform SBP-2 target functionality. However NT5.0 does not provide full functionality for all expected future extension of SBP-2 target. This clause describes what is required for SBP-2 target devices under NT5.0.

## 4.2. Configuration ROM

Each SBP-2 target device is required to have configuration ROM according to SBP-2 specification.

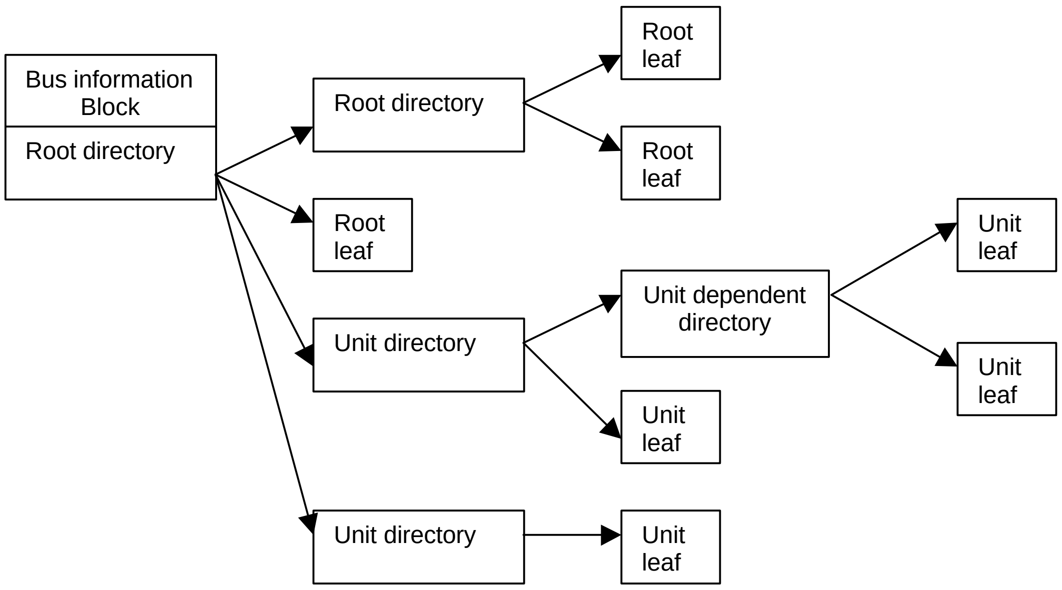


**Figure 1 Configuration ROM hierarchy**

## 4.3. Logical Unit

### *4.3.1. LUN and Unit Directory*

Windows NT5.0 is supporting only one Logical Unit Number (LUN) for one Unit Directory. Printing session from a SBP-2 initiator need to invoke “Login” to start session. And Login is associated a LUN. Normally each LUN allows an initiator to Login at a time. (see Appendix A.)

Each LUN is enumerated as stand alone device associated with its own driver stack. If any IHV provides their own Unit Directory and LUN which does not have specific device_type defined by Windows NT5.0, they need to install their proprietary driver stack, and their LUN will appear as if it is an independent hardware unit.

The following figure shows the relation ship of Unit Directories and LUNs under NT5.0.

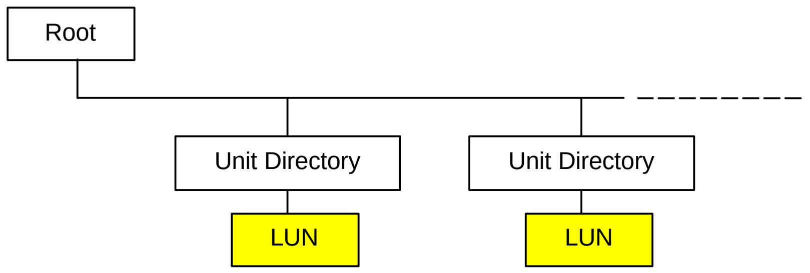


**Figure 2 LUN and Unit Directory**

### *4.3.2. Logical_Unit_Number entry*

The Logical_Unit_Number entry is an immediate entry that, when present in unit directory, specifies the peripheral device type and logical unit number of a logical unit implemented by the target.

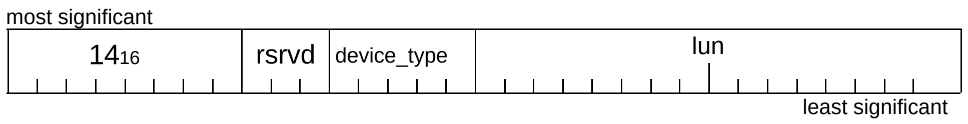


**Figure 3 Logical_Unit _Number entry format**

The value 0 to 0x1E indicate the device_type is command set dependent. The value 0x1F indicates unknown device type. The 1394 PWG supportive target shall have at least one of the value listed in the table below.

| value | Name | Description |
|---|---|---|
| 0x00 | DIRECT_ACCESS_DEVICE | disks |
| 0x01 | SEQUENTIAL_ACCESS_DEVICE | tapes |
| 0x02 | PRINTER_DEVICE | printers |
| 0x03 | PROCESSOR_DEVICE | scanners, printers, etc |
| 0x04 | WRITE_ONCE_READ_MULTIPLE_DEVICE | worms |
| 0x05 | READ_ONLY_DIRECT_ACCESS_DEVICE | cdroms |
| 0x06 | SCANNER_DEVICE | scanners |
| 0x07 | OPTICAL_DEVICE | optical disks |
| 0x08 | MEDIUM_CHANGER | jukebox |
| 0x09 | COMMUNICATION_DEVICE | networks |
| 0x1F | UNKNOWN_DEVICE | reserved for future standardization |

**Figure 4 definition for device_type field**

## 4.4. Current 1394 Driver Stack provided in Windows NT5.0

### *4.4.1. Driver stack provided by Windows NT5.0*

The figure shown below outlines device drivers stack provided in Windows NT5.0.

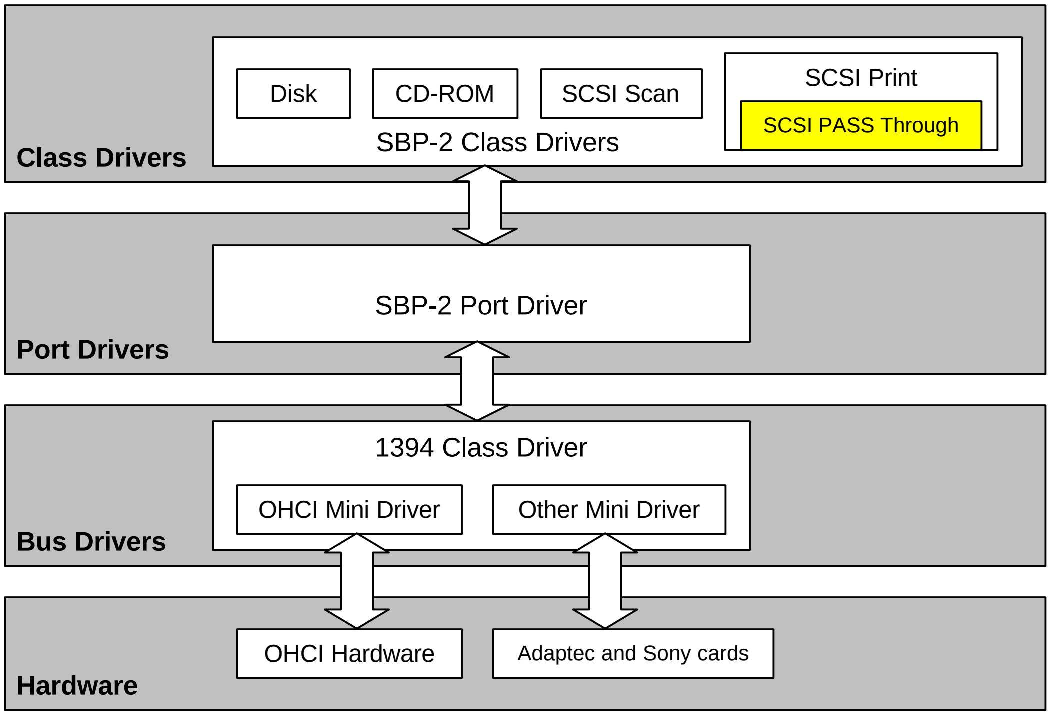


**Figure 5 Driver Stack in Windows NT5.0**

The upper side API of SBP-2 Port Driver is hidden from ISV or IHV developer. Instead of direct access to SBP-2 port, Windows NT5.0 is providing SBP-2 Class drivers including SCSI Print whose file name is “SCSIPrint.sys”. Even though the name of SCSIPrint.sys includes “SCSI”, SCSIPrint.sys does not require a target device to speak SCSI commands. SCSIPrint.sys is providing IOCTL_SCSI_PASS_THROUGH mode. Thus any vender may use it as if it acts a filter to pass data to SBP-2 Port Driver.

### *4.4.2. How does Windows NT5.0 enumerate a SBP-2 target device*

This section describes what is required in the configuration ROM of the target printer. Any NT5.0 supportive SBP-2 target device shall provide *Unit_Spec_ID* entry and *Unit_SW_Version* entry according to the SBP-2 specification. These entries are shown below.

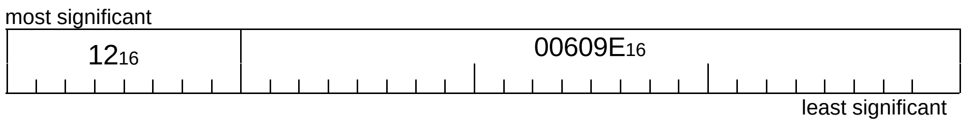


**Figure 6 Logical_Unit _Number entry format**

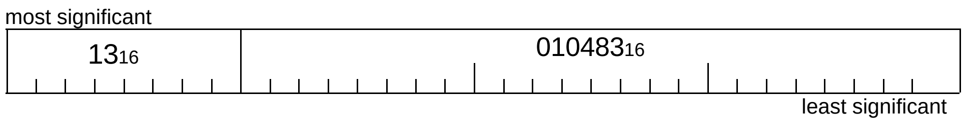


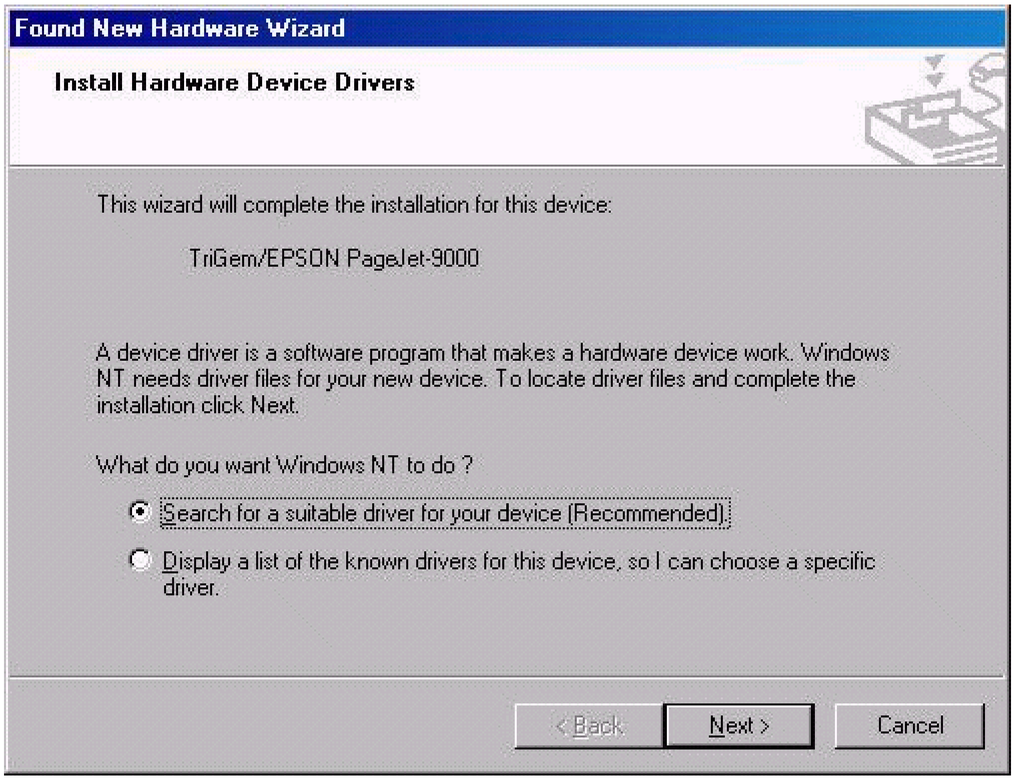
**Figure 7 Logical_SW _Version entry format**

The IEEE 1394 Bus Class driver of Windows NT5.0 enumerates configuration ROM of every target device after bus reset had been caused. When the IEEE 1394 Bus Class driver finds a SBP-2 target, it loads the SBP-2 Port driver automatically. Then the SBP-2 Port driver reads Logical_Unit_Numer entry of unit in the SBP-2 target. If the device_type field of the target has a value 0x02, the SBP-2 Port driver loads SCSIPrint.sys.

### *4.4.3. Uni-code textual descriptor*

Also an unit directory of the SBP-2 target shall have an uni-code textual descriptor which contains 1284 PnP id string. 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 was ignored under Windows 95.

Windows NT5.0 specifies the name of a suitable universal printer driver examining the manufacture name and model name of the printer, which in this case, is a SBP-2 target device enumerated by the IEEE 1394 Bus Class driver. Normally Hardware Wizard installs application level driver automatically.



This wizard will complete the installation for this device:

TriGem/EPSON PageJet-9000

A device driver is a software program that makes a hardware device work. Windows NT needs driver files for your new device. To locate driver files and complete the installation click Next.

What do you want Windows NT to do ?

● Search for a suitable driver for your device (Recommended).

○ Display a list of the known drivers for this device, so I can choose a specific driver.

**Figure 8 Sample figure of Hardware Wizard**

### *4.4.4. Drawback of SCSIPrint.sys and universal printer driver solution*

As described in previous section, Windows NT5.0 presents universal driver solution. For the IEEE 1394 printer, NT5.0 uses SBP-2 Port driver and “SCSIPrint.sys” which is a sort of SBP-2 Class driver. Also any IHV may provide proprietary protocol driver so-called “Custom Printer Driver”. However in latter case, user needs to find IHV’s driver from FDD, CD or Web site.

| Printer Driver Type | Developer | Description |
|---|---|---|
| Universal Driver | *Microsoft* | *User does not need to insert FDD nor CD* |
| Custom Driver | *Printer Vender* | *Custom drivers are only loaded when user inserts FDD or CD utilizing Hardware Wizard utility, then INF file is referred to install new driver on to OS* |

**Figure 9 Windows NT5.0 presents two types of printer driver**

Unfortunately, due to requirements for ease to use and may be due to too generalized method for driver development, Universal Printer Drivers has several limitation. Three important drawbacks of Universal Printer Drivers are listed below;

1. Universal Printer Drivers do not use reverse directional communication.
2. Universal Printer Drivers which use “SCSIPrint.sys” do not support multiple host connectivity.

- Software implementation of a target would solve this problem.
- SCSIPrint.sys itself does not prevent reverse directional data flow

3. Universal Printer Drivers which use “SCSIPrint.sys” prevent a Status Monitor application program to access same LUN of the target device, while a Spooler application is printing.

- Software implementation of a target would solve this problem.

## 4.5. Unsolicited status

The specification of SBP-2 is defining a functionality so-called “unsolicited status”. However the current SBP-2 Port Driver does not support unsolicited status from a target device.

The 1394 PWG printing profile (Rev. 0.31, May 6,1998) is requesting that “No device-specific or service-specific information shall be carried in the unsolicited status message”. As the realistic solution, IHVs shall choose, target architecture which does not require to issue unsolicited status, even though they provide proprietary SBP-2 Class driver over Microsoft’s SBP-2 Port driver.

## 4.6. How does I/O Manager enumerate IHV’s Custom Driver

### *4.6.1. Background*

If universal printer drivers have some limitation, one possible solution to solve this problem is to develop custom driver. Naturally IHVs are required to distribute their own printer driver products and INF files using FDD, CD or Web site.

From NT5.0 operating system’s point of view, loading custom drivers has two scheme. First scheme loads device drivers and second scheme determine the name of the suitable printer driver and load it. The action taken by second step shall be described from port monitor side.

The following section outlines how each software works when a printer device is plugged in the IEEE 1394 bus.

### *4.6.2. Finding 1394 node on the bus*

Enumerating occurs in the IEEE1394 bus class driver when a 1394 printer is plugged in. The adding process of the enumerated device were executed in IEEE1394 Filter driver. One possible example of the IEEE 1394 Filter driver is SBP-2 Port driver developed by Microsoft .The access to a target device through I/O manager becomes possible after the device adding process in "IEEE1394 Filter Driver" had been taken. Furthermore, the individual printer information that was connected to IEEE1394 Bus is also stored in registry. A device name including textual descriptor from Configuration ROM of the 1394 device will be written as the key of registry.

### *4.6.3. How does a Port Monitor load the printer driver*

After device level program found a 1394 node on the bus, A “Port Monitor” user mode application may invoke SetupDiEnumDeviceInterfaces() API to enumerate the target 1394 printer.
Through enumeration process, the port monitor generates the printer port name (PrinterPnP01,PrinterPnP02....) corresponding to each printer. And the port monitor returns enumerating information with this port name to spooler. This enumerating information is displayed as an applicable printer port name with the "addition WIZARD of printer". A user application may invoke CreateFile() API specifying the parameter which contains this printer port name. Then this application may access the target device utilizing WriteFile() or ReadFile() APIs.

## 5. Bi-directional Communication Model

Windows NT5.0 and it's existing driver stack including SBP-2 Port driver provides full duplex, bi-directional communication capability between an initiator device and a target device utilizing unordered ORB processing model. Data transfer in a specific direction is accomplished in order that the ORBs are placed on the list with respect to that direction.

An initiator shall place at least one control ORB which is one special purpose input ORB to solicit the target to report buffer requirement for next reverse directional data session. Also the initiator shall place negotiation ORB which is on special purpose output ORB to inform buffer space, when the initiator only can provide smaller space than requested space, as input buffer.

### 5.1. Initiator Model

Initiators shall provide four types of normal command bock ORBs shown below;

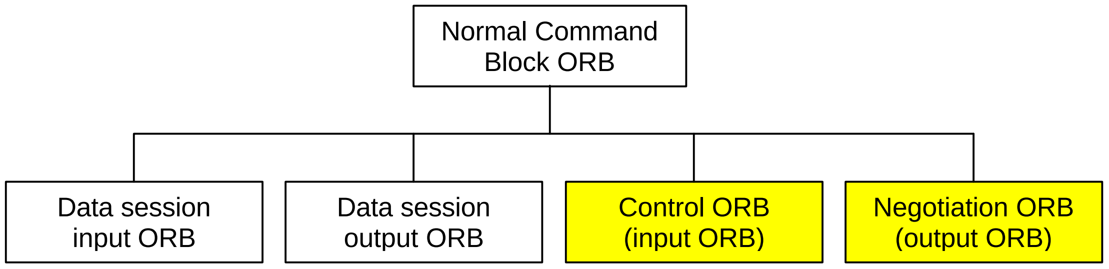


**Figure 10 Normal command block ORB types for bi-d**

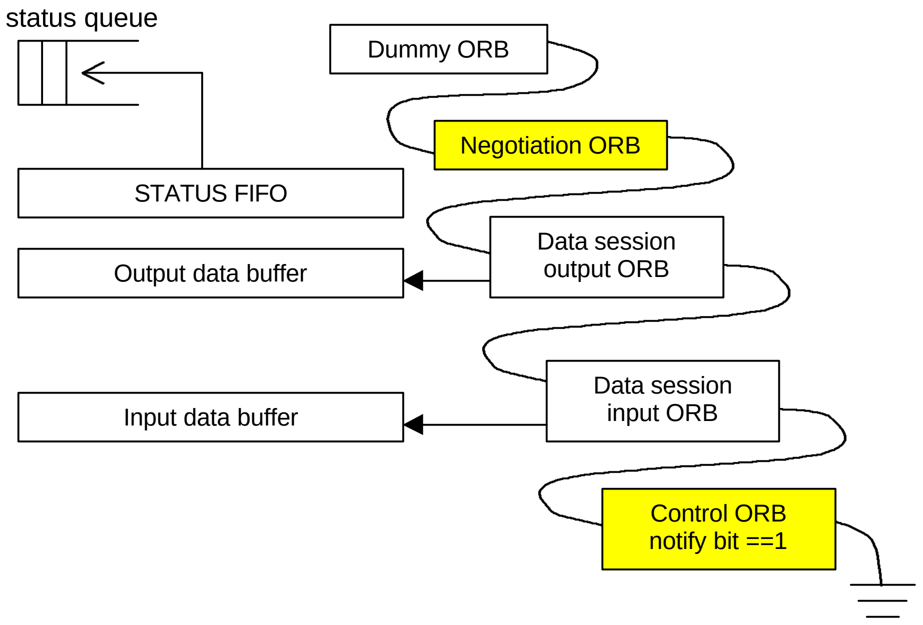


**Figure 11 ORBs list in an initiator device**

An initiator device shall place at least a Negotiation ORB, a Control ORB and some data session ORBs in the linked list of ORBs. Data session out put ORB shall have a pointer points an output buffer.

Negotiation ORB is an output ORB which shall be responsible to **grant credit** for the target device in forward data session, and be responsible to **issue credit request** to the target device in reverse data session.

Control ORB is an input ORB which shall be responsible to **receive credit granted for** the initiator device from the target device, in forward data session, and be responsible to **receive credit request from** the initiator device in reverse data session.

The size of output data and the length of the output buffer is determined according to the input credit from the target device. And the size of the input buffer is limited by the credit issued from the initiator to the target. The size of received data will be informed when the target device consumed Control ORB and returned STATUS BLOCK to the initiator.

| Direction of data | Negotiation ORB | Control ORB |
|---|---|---|
| forward data session<br>Initiator → Target | Issue credit request | Receive granted credit |
| reverse data session<br>Target → Initiator | Grant credit | Receive requested credit |

**Figure 12 Responsibility of Negotiation ORB and Control ORB**

## 5.2. Target Model

A target device may skip execution of requested ORB, when it is an input ORB and write queue does not have any request to write to the initiator. The target shall consume Control ORB at the end of one data session, and shall be responsible to report buffer space kept in current state.

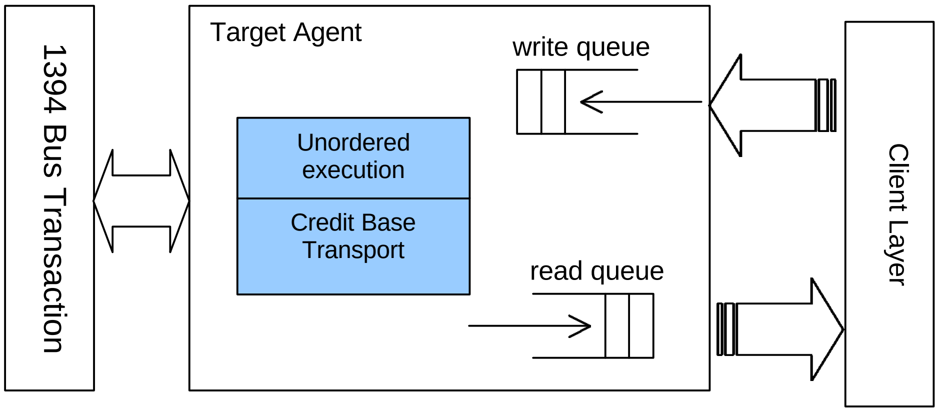


**Figure 13 Target Model**

## 5.3. Status Block returned in unordered execution

In unordered execution model, an initiator easily cause critical section if an initiator removes outstanding ORBs from the list. Thus, a target device is responsible to return status block with suitable information in it's *src* field.

When the *src* filed has value zero, the target has knowledge for a subsequent ORB to be fetched, and the ORB will never be fetched by the target. Thus the initiator may reuse or de-allocate it.

When the *src* field has value one, the target does not know the address of a subsequent ORB, until the initiator pings the door bell. And the ORB will be fetched again after the initiator pings the door bell. Because the target needs to read next_ORB field from this ORB. Thus the initiator shall not reuse or de-allocate it, until the target returns completion status for subsequent ORBs.

# 6. Multiple Host and/or Multiple Device

## 6.1. Multiple Host service with single LUN model

This section describes, how SCISPrint.sys (or PnPprint.sys) and SBP-2 Port driver under Windows NT5.0 can provide multiple host connectivity within only one LUN in a target device.

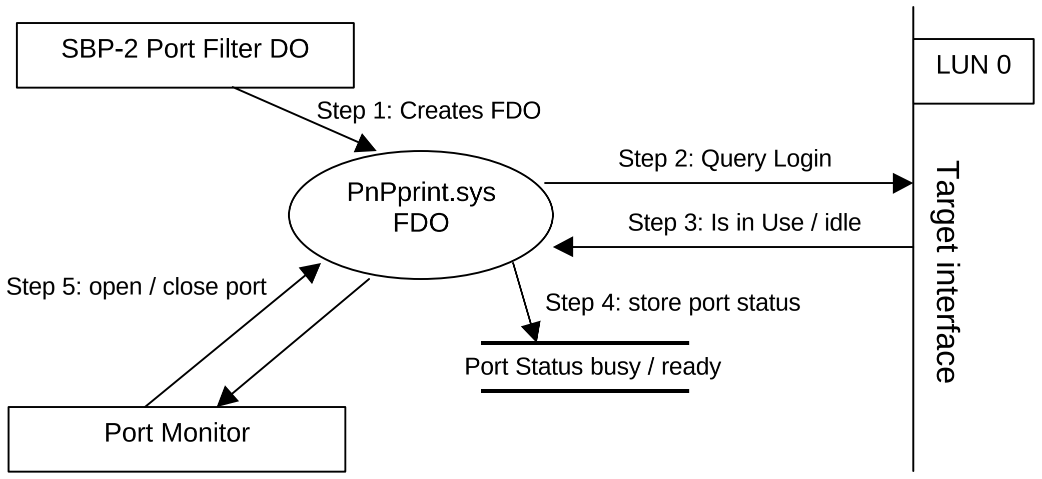


**Figure 14 Port open / close process**

Under this environment, an user mode application called Port Monitor shall be responsible to open and close connection to the target device LUN 0, at each printing session. The “SBP-2 Port” filter device object (Filter DO) creates PnPprint.sys function device object (FDO), then this FDO invokes “Query Login” to the specific target. The result is kept in the registry, and the FDO returns this result, when the Port Monitor opened the port. If the Port Monitor got result which say “port is busy”, the Port Monitor is required to close port as soon as possible, and retry to open same port several seconds later, until it receives successful result.

This mechanism avoid conflict of duplicated access to one LUN in the specific target device.

## 6.2. Multiple Host service with dual LUN model

One of simple and effective idea to service multiple host connectivity is to provide two Unit Directories and two LUNs which is associated to each Unit Directory.

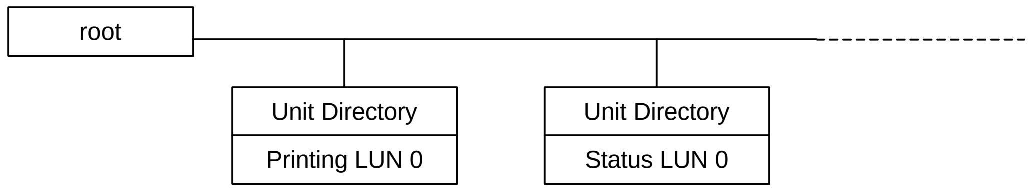


**Figure 15 Status Unit Directory and LUN model**

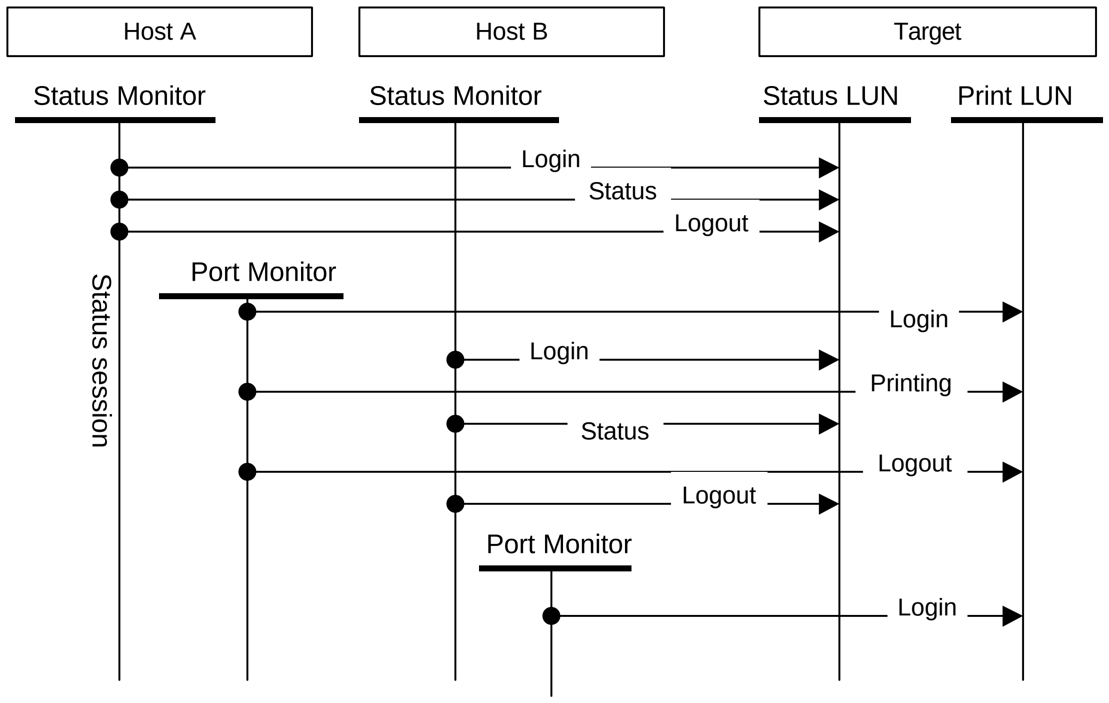


**Figure 16  Two LUN model for Multiple Host service**

The Status LUN allows short status session for initiator devices. An initiator may reserve Login to the printing session utilizing the Status LUN. The Status LUN returns allowance to login to the Printing LUN, if the Printing LUN is in idle state.

Otherwise, if the Printing LUN is occupied by other initiator device, then the Status LUN remember a time stump which notes when this initiator requested first contact to the printing session. The Status LUN does not provide “login solicitation” capability. Instead of this capability, the Status LUN returns the tome constant for this initiator to retry login to Status LUN.

If several initiator is trying to connect to the Printing LUN of a specific target, the first initiator receives most shortest retry time constant in seconds. And the initiator which had requested next to the first one receives shorter time constant than other initiators. If the Printing LUN become to be free, the STATUS LUN returns allowance to login to the Printing LUN.

# 7. 1394 PWG aware Printer Driver

## 7.1. PWG LUN support in Windows NT5.0 second generation

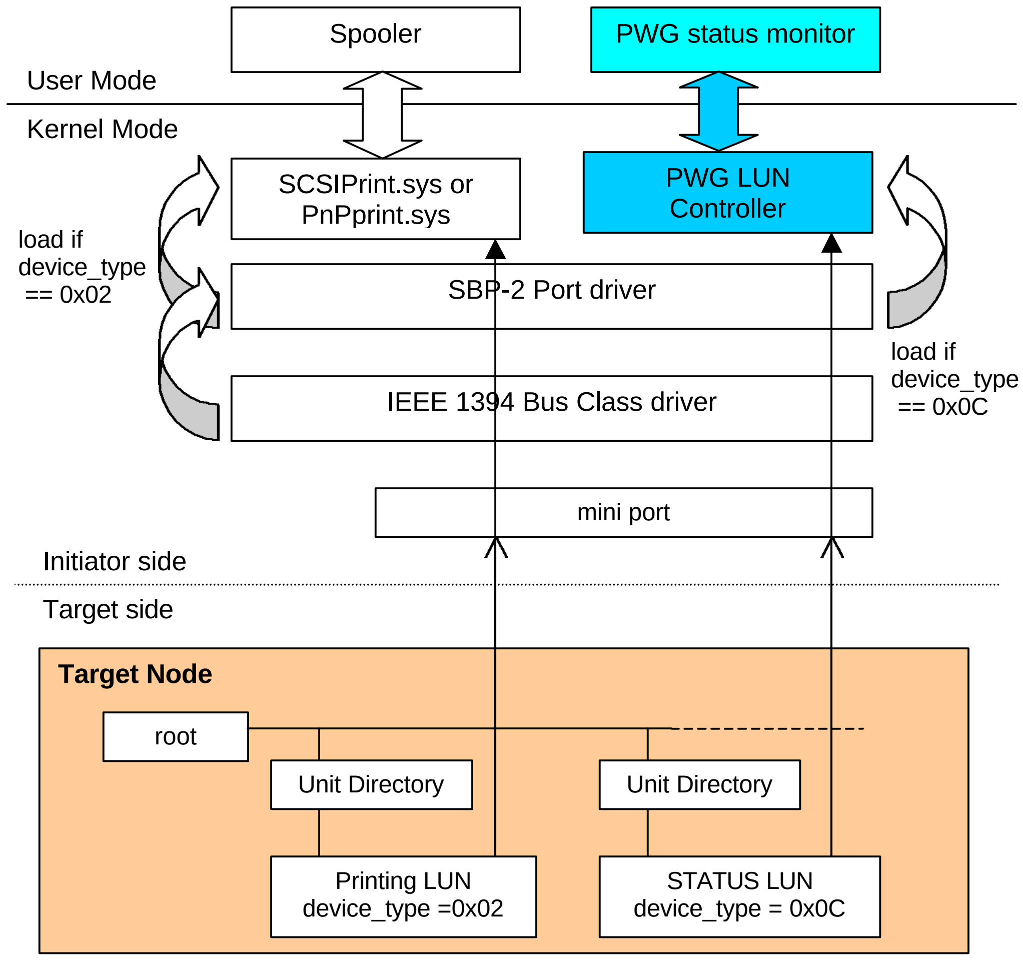

Spooler
PWG status monitor
User Mode
Kernel Mode
SCSIPrint.sys or PnPprint.sys
PWG LUN Controller
load if device_type == 0x02
SBP-2 Port driver
load if device_type == 0x0C
IEEE 1394 Bus Class driver
mini port
Initiator side
Target side
Target Node
root
Unit Directory
Unit Directory
Printing LUN device_type =0x02
STATUS LUN device_type = 0x0C

**Figure 17 Additional driver stack for 1394 PWG aware drivers**

The figure illustrated above shows additional driver stack which expected to be available after Windows NT5.0 Service Release. The SBP-2 Port driver shall be responsible when Logical_Unit_Number entry of Configuration ROM has device_type field as value 0x0C.

To support this architecture, a printer vender shall provide new unit directory which is associated with “STATUS LUN”, in their 1394 target device. And Microsoft shall choose one of these solution listed below;
1) develop “PWG LUN Controller” protocol driver as one of the “SBP-2 Class”.
2) open API sets of “SBP-2 Port driver” to encourage IHVs to develop their proprietary 1394 PWG aware “LUN Controller” SBP-2 Class driver.

## 7.2. 1394 full feature LUN support in Windows NT6.0

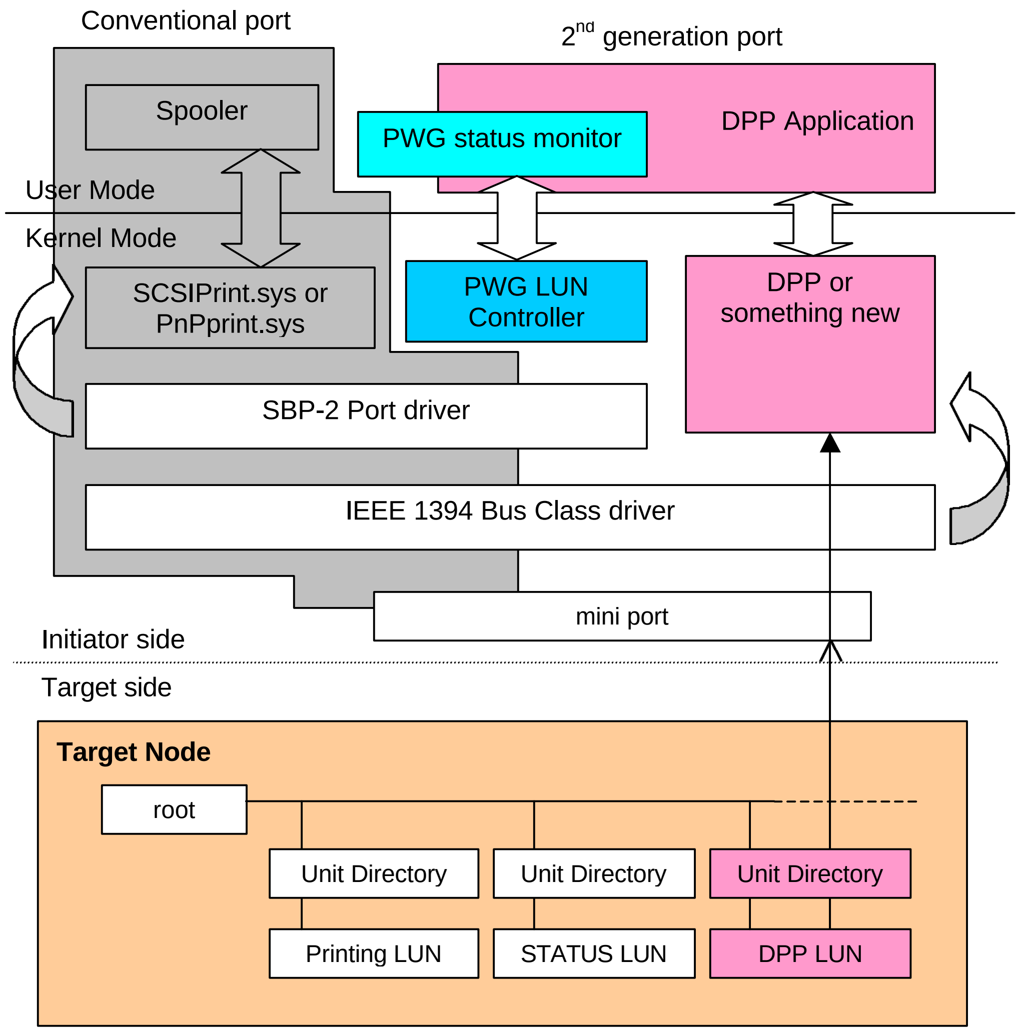


**Figure 18 Protocol stack applicable new protocol drivers**

PWG-C or DSIWG/1394 TA is going to standardize new symmetrical 1394 printing protocol. To make this protocol driver available under next Windows NT (may be 6.0), PWG-C or DSIWG is responsible to determine *Unit_Spec_ID* and *Unit_SW_Version*. These values are required for “1394 Bus Class Driver” to find correct target device and to load suitable device drivers.

Also, any IHV who is going to provide these protocol drivers, is responsible to ship target device which has an Unit Directory associated with DPP LUN.

# 8. How to load IHV's proprietary drivers

## 8.1. Background

An IHV will be required to provide mechanism to install their own driver stack, if they are going choose other IEEE 1394 protocol like DPP or something new, instead of SBP-2 Port driver and SCSIPrint.sys which is available in Windows NT5.0. Or any IHV may use both SCSIPrint.sys and their proprietary driver simultaneously. As an assumption, the SCSIPrint.sys would be good enough to perform simple printing session, and IHV's protocol driver shall be responsible to report status of the target device and be responsible connection management to support multiple host connectivity.

The following sections describe how NT5.0 I/O manager and drivers know what shall be loaded while enumeration process has been begun.

## 8.2. Primary Enumeration

The I/O manager controls "1394 Bus Class Driver" to enumerate 1394 bus device. This enumeration shall be completed before an IEEE 1394 Filter Driver starts PnP process. One of well known example of the "IEEE 1394 Filter Driver" is "SBP-2 Port Driver" which was mentioned in previous section.

The following shows job steps taken in primary enumeration phase;

1. 1394 Bus Reset occurs.
2. MS's 1394 Bus Class Driver (1394 BUS) enumerates the buses
   Reads every node's "Configuration ROM".
   Configures all logical unit's "PnP ID" from "Configuration ROM" information,
3. Search "1394 PnP ID" in "HKEY_LOCAL_MACHINE\Enum\1394"
   if "PnP ID" is found , goto step 6
   if "PnP ID" is not found, goto step 4
4. Creates driver database from ".inf" files in "Windows\inf"
5. Search "PnP ID" from the ID information of the driver database
   if found, create a key whose name is same as "PnP ID" in HKEY_LOCAL_MACHINE\Enum\1394.
   if not found, prompt to user "insert floppy disk".
6. The I/O manager writes the driver information key name in KEY_LOCAL_MACHINE\Enum\1394\[PnP_id]\Class.
7. The I/O manager writes the driver name(eg.PrntFilt.sys) that was loaded by this driver in KEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Class\[driver information key name]\NTMPDriver contains.

## 8.3. Secondary Enumeration

I/O manager calls "PnPAddDevice" API of "IEEE1394 Filter Driver" when a 1394 target printer was detected on the bus. The following sections outline job steps of "IEEE1394 Filter Driver" associated "PnPAddDevice" API call.

### *8.3.1. Generation of the device handle name and FDO (Function Device Object)*

Generates the device handle name required by a user mode application to access the device. The unique device name including GUID is generated by Bus Class Driver which is Physical Device Object (PDO). Or Bus Class Driver may create a device object without specific name. Then **IoCreateDevice**() API is used to generate FDO.

### *8.3.2. The FDO connection to the device layer*

The generated FDO is connected to a device layer. **IoAttchDeviceToDeviceStack**() API binds generated FDO to lower level PDO. These steps are shown below;

```
NTSTATUS AddDevice(IN PDRIVER_OBJECT DriverObject,
                   IN PDRIVER_OBJECT pdo)
{
PDEVICE_OBJECT     fdo;    // function device object
NTSTATUS           status;
status = IoCreateDevice(DriverObject,
                   sizeof(DEVICE_EXTENSION), NULL,
                   FILE_DEVICE_UNKNOWN, 0, FALSE,
                   &fdo);
if (!NT_SUCCESS(status))
      return status;
PDEVICE_EXTENSION pdx = (PDEVICE_EXTENSION)fdo->DeviceExtension;
status = IoRegisterDeviceInterface(      pdo,
                                         &GUID_PWG,
                                         NULL,
                                         &pdx->ifname);
if (!NT_SUCCESS(status)) {
      IoDeleteDevice(fdo);
      return status;
      }

      IoSetDeviceInterfaceState(&pdx->ifname, TRUE);
pdx->LowerDeviceObject = IoAttchDeviceToDeviceStack(fdo, pdo);
fdo->Flags &= ~DO_DEVICE_INITIALIZING;
return STATUS_SUCCESS;
}
```

### *8.3.3. Getting connected device ID*

Textual Descriptor of the devices which was found on 1394 bus is extracted from Configuration ROM. The Textual Descriptor shall contain unique device identifier that can distinguish a node from other nodes, also shall contain some information to specify applicable higher level drivers. From this point of view, a "1284 PnP ID" and GUID shall be included in this Textual Descriptor.

### *8.3.4. Registry update*

Registry shall keep information involved with relationship between devices ID and device handle name. This information will be changed when the IEEE1394 bus enumerating has been processed. The port monitor reads data from the registry listed below. The port monitor binds devices ID with device handle name.

| | |
|---|---|
| ***position in registry*** | HKLM\SOFTWARE\Epson\IEEE1394\Driver\ |
| ***registry key and value*** | Key = device handle’s name (including GUID ) Value = Textual Descriptor (1284 PnP ID + GUID + Serial number ) Or Configuration ROM 8byte(64bit) ID (Vender ID 3byte ,chip ID 5byte) |

## 8.4. Application level Plug and Play

### *8.4.1. Port Enumeration*

A Port Monitor enumerates the printer ports on IEEE 1394 bus, when the spooler starts up. Then it reports results to the spooler. For example, the standard local port monitor enumerates LPT1, LPT2, COM1, COM2, FILE as the local ports. The spooler keeps information returned from the Port Monitor and the printer ports are displayed as the available printer ports on "Add Printer Wizard".

### *8.4.2. Step 1; read keys from the registry*

Get the data structure which describes relationship between the device handle's names and the device IDs from the specific position of the registry listed below. This data structure is enumerated by "IEEE 1394 Filter Driver". The port monitor can search the device handle's name using the known device ID.

| | |
|---|---|
| **position in registry** | HKLM\SOFTWARE\Epson\IEEE1394\Driver\ |
| **registry key and value** | Key = device handle's name, prtntXX. Value= Configuration ROM 8byte(64bit) ID or the unique value based on it |

**Figure 19 The registry information recorded by the filter driver**

### *8.4.3. Step 2; Store Port Monitor’s property*

The Port Monitor generates the registry key to store binding information for printer port names and device IDs. The port monitor always shall assign the same printer as the same port name, utilizing this information.

| | |
|---|---|
| **position in registry** | HKLM\SOFTWARE\Epson\IEEE1394\Monitor\ |
| **registry key and value** | Entry name = printer port name. For example, ep1394prnXX Device ID key = Configuration ROM 8byte(64bit) ID |

**Figure 20 The registry information recorded by SBP-2 port monitor**

The key of the device handle name is involved with the device handle name that could be changed at every time. Also the status key shall be involved with the printer port which could be changed at every time.

For example,
HKLM\SOFTWARE\Epson\IEEE1394\Monitor\ep1394prn01\
HKLM\SOFTWARE\Epson\IEEE1394\Monitor\ep1394prn01\ID 1234

### *8.4.4. Step 3; Bind the printer port names with the device handle names.*

Now the printer port names is associated with the device handle names, through the device ID as intermediary name. Thus the Port Monitor can open the driver and print to the IEEE 1394 printer associated with the printer port name.

### *8.4.5. Step 4; Make the new printer port for the new printer*

When detected printer was completely new, the name would not be resolved within step 3, then the printer port for it shall be created from scratch.

The printer port name is created whenever the new printer is connected, for example ep1394prn01, ep1394prn02, ep1394prn03. After the port name was created, the data structure which describes relationship between the printer port names and the device IDs is recorded in the registry. Thus, the same port name is always assigned for the same printer.

| | |
|---|---|
| **position in registry** | HKLM\SOFTWARE\Epson\IEEE1394\Monitor\ |
| **registry key and value** | Entry name = printer port name. For example, ep1394prnXX<br>Device ID key = Configuration ROM 8byte(64bit) ID |

**Figure 21 The registry information recorded by SBP-2 port monitor**

### *8.4.6. Step 5; Confirm the connection.*

Confirm whether or not the printer is connected actually. To do that, the port monitor calls CreateFile() to all printer ports that it knows. The results will be stored in the registry.

### *8.4.7. Step 6; Create the description for the port.*

The port monitor creates the port description for every printer port. The Port Monitor has the data structure listed below, which contains the pointer to the next description. So the port descriptions are kept as the linked list. The created description will be returned as a port handle to the spooler, when the Port Monitor invokes OpenPort() API to open connection.

```
// The definition of the port description
//   For SBP2 Port Monitor
typedef struct _INIPORT {
```

```
// basic information

DWORD   signature;                // signature
                                  // value to check the validity of the handle.
DWORD   cb;                       // the size for port description including the string size.

// link information

struct  _INIPORT *pNext;                  // the pointer to the next port description

// port information

LPWSTR  pPortName;                // port name
DWORD   Status;                           // port status
LPWSTR  pDeviceHandleName;        // device handle name
LPWSTR  pDeviceID;                // device ID
DWORD   JobId;                    // print job ID

// port control to Write and Read

HANDLE pIventHandle;              // This is used when the Callback for WritePort() waits.
                                  // This is set by WritePort().
DWORD dwWriteTimeout;
DWORD dwReadTimeout;

// The control information for SBP2

SBP2SESSION sbp2;                 // SBP2 session specific information
                          // This is set in StartDocPort().

} INIPORT, *PINIPORT;
```

### 8.4.8. *Step 7; Notice the enumeration information to the spooler.*

The port monitor stores the enumerated data into the port-information-structure; PORT_INFO, then returns the pointer to the arrangement of the port-information-structure to the spooler. The spooler displays the list of the ports on "Add Printer Wizard" window, using this data structure.

```
/**********************
 *
 * function:   LocalmonEnumPorts
 *
 * purpose:    Enumerate printer ports
 *
 * arguments:      LPTSTR   pName = not used
 *             DWORD   Level  = structure level
 *             LPBYTE  pPorts = spooler data structure
 *             DWORD   cbBuf  = size of spooler structure passed in
 *             LPDWORD pcbNeeded = size of spooler structure required
 *             LPDWORD pcReturned = number of entries returned
 *
 *
 * returns:    pPorts, pcbNeeded, bpcReturned
 *
 */

BOOL WINAPI LocalmonEnumPorts(
    LPTSTR   pName,
    DWORD   Level,
    LPBYTE  pPorts,
    DWORD   cbBuf,
    LPDWORD pcbNeeded,
    LPDWORD pcReturned
)
{
              … IHV Programming …
}
```

# Appendix A. Example of Configuration ROM of Windows NT5.0 applicable printer

The following shows an example Configuration ROM which is including NT5.0 supportive textual descriptor with 1284 PnP ID.

```
/*          value           no add       comment */
/* Bus information block */
            0x045ecd53,     /*  1 400           length of configurationROM & all CRC */
            0x31333934,     /*  2 404           ASCII strings “1394”. */
            0x50327000,     /*  3 408           bit 31 : 0 = capable of isocronous
resource manager.
bit 30 : 1 = capable of cycle master.
bit 29 : 0 = isochronous supported.
bit 28 : 1 = bus manager supported.
bit 27-24 : reserved
bit 23-16 : 32h = cycle
clock accuracy
(must be FF with no cycle master support)
bit 15-12 : 128 = max record size.
bit 11-0  : reserved. */
0x000048FF,     /*  4 40c           Vendor ID value 000048 and Chip ID value
FF00000001 */
0x00000001,     /*  5 410           */
/* Root directory. */
            0x000434a7,     /*  6 414           length of Root directory & CRC */
            0x03000048,     /*  7 418           Module vendor ID. Value is 000048. */
            0x0c008380,     /*  8 41c           bit 24-16 : reserved.
                                                                bit 15 : 1=SPLIT_TIMEOUT
register is implemented.
bit 14 :
1=messages-passing registers are implemented.
bit 13 :
1=INTERRUPT_TARGET and INTERRUPT_MASK registers are implemented.
bit 12 : 1=ARGUMENT registers are implemented.
bit 11 : 1=Node implements TEST_START&TEST_STATUS registers and testing state.
bit 10 : 1=Node implementes the private space.
bit  9 : 1=Node uses
64-bit addressing
bit  8 : 1=Node uses the fixed addressing scheme.
bit  7 :
1=STATE_BITS.lost bit is implemented.
bit  6 :
1=STATE_BITS.dreq bit is implemented.
bit  5 : 1=Reserved. = 0.
bit  4 :
1=STATE_BITS.elog bit and ERROR_LOG registers are implemented.
bit  3 :
1=STATE_BITS.atn bit is implemented.
bit  2 :
1=STATE_BITS.off bit is implemented.
bit  1 : 1=Node supports the dead state.
bit  0 : 1=Node supports
the initializing state. */
        0x8d000002,    /*  9 420         Node unique ID leaf offset. */
        0xd1000004,    /* 10 424         Unit directory No.1 offset. (used to
transmit data to device medium) */
/* Node unique ID leaf. */
        0x00022ddc,    /* 11 428         length of leaf & CRC */
        0x000048FF,    /* 12 42c         */
        0x00000001,    /* 13 430         serial number */
                                                 /*        Vendor ID and Chip ID.
*/
/* Unit directory. (No.1) */
        0x0009aa02,    /* 14 434         length of Unit directory & CRC */
        0x1200609e,    /* 15 438         Unit spec ID ?? */
        0x13010483,    /* 16 43c         unit_sw_version (target conforms to
SBP-2). */
        0x38000000,    /* 17 440         command_set_spec_ID. */
        0x39000000,    /* 18 444         command_set. */
        0x3b000000,    /* 19 448         command_set_version. */
        0x54004000,    /* 20 44c         csr_offset (MANAGEMENT_AGENT register is
at FFFF F001 0000). */
        0x3a400a08,    /* 21 450         logical_unit_characteristics.
                                                         bit 23    : queuing bit
(task management model is SBP-2).
bit 22    : ordered bit (executes all tasks in order).
bit 21    : isochronous bit (does not support)
bit 20-16 : reserved.  bit 15-8  :
login_timeout (500x10 milliseconds).
bit 7-0   : ORB_size (8
quadlets). */
        0x14020000,    /* 22 454         logical unit number. */
        0xd4000001,    /* 23 458         Unit dependent directory offset. */
/* Unit dependent directory */
        0x00049bae, /* 24 45c   Unit dependent directory length & CRC */
        0x81000004,    /* 25 460         vender key(Unicode) offset */
        0x81000009,    /* 26 464         vender key(ascii) offset */
        0x8200000d,    /* 27 468         model key(Unicode) offset */
        0x82000030,    /* 28 46c         model key(ascii) offset */
/* Vendor Leaf(Unicode) */
        0x00058cce,    /* 29 470         vendor Leaf(Unicode) length & CRC */
        0x80000000,    /* 30 474         vendor spec ID */
        0x00000409,    /* 31 478         vendor language ID */
        0x45005000,    /* 32 47c         vendor text "EPSON " */
        0x53004f00,    /* 33 480         */
        0x4e000000,    /* 34 484         */
/* Vendor Leaf(ascii) */
        0x000452da,    /* 35 488         vendor Leaf(Ascii) length & CRC */
        0x00000000,    /* 36 48c         vendor spec ID */
        0x00000000,    /* 37 490         vendor language ID */
        0x4550534f,    /* 38 494         vendor text "EPSON   " */
        0x4e000000,    /* 39 498         */
/* Model Leaf(Unicode) */
```

```
        0x00235621,     /* 40 49c       model Leaf(Unicode) length & CRC */
        0x80000000,     /* 41 4a0       model spec ID */
        0x00000409,     /* 42 4a4       model language ID */
        0x4d004600,     /* 43 4a8       model text
“MFG:EPSON;CMD:ESCPL2E,PRPXL,BDC;MDL:Stylus COLOR 800;CLS:PRINTER;”. */
        0x47003a00,     /* 44 4ac       */
        0x45005000,     /* 45 4b0       */
        0x53004f00,     /* 46 4b4       */
        0x4e003b00,     /* 47 4b8       */
        0x43004d00,     /* 48 4bc       */
        0x44003a00,     /* 49 4c0       */
        0x45005300,     /* 50 4c4       */
        0x43005000,     /* 51 4c8       */
        0x4c003200,     /* 52 4cc       */
        0x45002c00,     /* 53 4d0       */
        0x50005200,     /* 54 4d4       */
        0x50005800,     /* 55 4d8       */
        0x4c002c00,     /* 56 4dc       */
        0x42004400,     /* 57 4e0       */
        0x43003b00,     /* 58 4e4       */
        0x4d004400,     /* 59 4e8       */
        0x4c003a00,     /* 60 4ec       */
        0x53007400,     /* 61 4f0       */
        0x79006c00,     /* 62 4f4       */
        0x75007300,     /* 63 4f8       */
        0x20004300,     /* 64 4fc       */
        0x4f004C00,     /* 65 500       */
        0x4f005200,     /* 66 504       */
        0x20003800,     /* 67 508       */
        0x30003000,     /* 68 50c       */
        0x3b004300,     /* 69 510       */
        0x4c005300,     /* 70 514       */
        0x3a005000,     /* 71 518       */
        0x52004900,     /* 72 51c       */
        0x4e005400,     /* 73 520       */
        0x45005200,     /* 74 524       */
        0x3b000000,     /* 75 528       */
/* Model Leaf(ascii) */
        0x0013c813,     /* 76 52c       model Leaf(Ascii) length & CRC */
        0x00000000,     /* 77 530       model spec ID */
        0x00000000,     /* 78 534       model language ID */
        0x4d46473a,     /* 79 538       model text
“MFG:EPSON;CMD:ESCPL2E,PRPXL,BDC;MDL:Stylus COLOR 800;CLS:PRINTER;”. */
        0x4550534f,     /* 80 53c       */
        0x4e3b434d,     /* 81 540       */
        0x443a4553,     /* 82 544       */
        0x43504c32,     /* 83 548       */
        0x452c5052,     /* 84 54c       */
        0x50584c2c,     /* 85 550       */
        0x4244433b,     /* 86 554       */
        0x4d444c3a,     /* 87 558       */
        0x5374796c,     /* 88 55c       */
```

```
0x75732043,    /* 89 560       */
0x4f4C4f52,    /* 90 564       */
0x20383030,    /* 91 568       */
0x3b434c53,    /* 92 56c       */
0x3a505249,    /* 93 570       */
0x4e544552,    /* 94 574       */
0x3b000000,    /* 95 578       */
```

# Appendix B. SBP-2 LUN

SBP-2 Spec Rev.3d

*p27*

The exclusive bit (abbreviated as x in the figure above) shall specify target behavior with respect to concurrent login to a logical unit. When exclusive is zero, the target, subject to its own implementation capabilities, may permit more than one initiator to login to a logical unit. If exclusive is one the target shall permit only one login to a logical unit at a time; see 8.2 for a description of target behavior.

*p56*

The target shall perform the following steps (in any order) to validate a login request:

- The target shall read the initiator's unique ID, EUI-64, from the bus information block by means of two quadlet read transactions. The source_ID from the write transaction used to signal the login ORB to the target's MANAGEMENT_AGENT register shall be used as the destination_ID in the quadlet read transactions;
- The target shall determine whether or not the initiator already owns a login by comparing the EUI-64 just obtained against the login_owner_EUI_64 for all login_descriptors. If the initiator is currently logged-in to the same logical unit, the login request shall be rejected with an sbp_status of access denied;
- If the exclusive bit is set in the login ORB and there are any active login_descriptors for the logical unit, the target shall reject the login request with an sbp_status of access denied;
- If an active login_descriptor with the exclusive attribute exists for the lun specified in the login ORB, the target shall reject the login request with an sbp_status of access denied;
- Else the target shall determine if a free login_descriptor is available and, if none are available, reject the login request with an sbp_status of resources unavailable.

end of file