# Standard for Device and Service Discovery on an IEEE Std. 1394 Topology

**Proposal Draft Ver.0.4a**
**July 10,1997**

## Canon Inc.

| date | version | revision notes |
|---|---|---|
| 97/4/2 | V0.2 | original release |
| 97/6/6 | V0.3 | section 6 renewed, DDP document ver.0.1 |
| 97/6/15 | V0.4 | renamed and rewritten based on PWG-C 6/11,12 meeting results |
| 97/7/11 | V0.4a | modified based on PWG 6/23,24 meeting results |

# TABLE OF CONTENTS

# 1. INTRODUCTION

## 1.1 SCOPE

This document will describe the 1394 Device Discovery and status retrieval (DDsr(Function Discovery;FDP)) Protocol Specification which should apply to all devices that connect to the IEEE1394 High Performance Serial Bus. The specification will provide a flexible method for any device to comply to.

## 1.2 PURPOSE

This purpose of this protocol specification is to define a universal, method for simple discovery of functional units and low-level service discovery within a IEEE1394 device. What will be described is;

- A method for identifying a DDsr(Function Discovery;FDP) compliant node.
- A method for identifying a functional unit (or multiple functional units) within a DDsr(Function Discovery;FDP) compliant node.
- A method for retrieving information on each of the functional units.
- definition of the DDsr(FDP) protocol to support the above functions.

## 1.3 BACKGROUND

Even though the current IEEE1394-1995 specifies the configuration ROM format following rules of the IEEE 1212 -1994 standards, thus allowing IEEE1394 node discovery, it does not define a common method for discovering the function units within a particular node.

Currently, there are communication protocols focused on IEEE1394 communication such as the Function Command Protocol (FCP) used with the AV command set, and the Serial Bus Protocol-2 (SBP-2) which acts as a lower-layer to command sets. Some of these communication protocol (stacks) each provide a method for discovering the function units of the node, but under the condition that the discovery will be accomplished within that particular protocol definition. There may be other protocols that do not support function discovery at all.

In any case, definition of a universal method for discovery of functional units and low-level service discovery is necessary and will be useful for any 1394 device.

## 1.4 REFERENCES

- IEEE Std. 1394-1995, Standard for a High Performance Serial Bus
- IEEE Std. 1284-1994, Standard for Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers
- ANSI/IEEE Std. 1212 ISO/IEC 13213 Control and Status Registers (CSR) Architecture for microcomputer buses
- 1394-based Digital Camera Specification Version 1.04 August 9,1996
- AV/C Digital Interface Command Set Version 1.0 September 13,1996

# 2. DEFINITIONS

## 2.1 TERMINOLOGY

**Datalink**
In this document, a “datalink” will be defined as the (or part of the) lowest layer above the transaction layer defined in the IEEE1394-1995 specification. It will be the lowest layer of the printer protocol stack in the communication protocol stack.

**Transport**
In this document, a “transport” will be defined as the set of communication protocol stacks as a whole. The set will consist of protocol layers ranging from the data-link layer to the application layer.

**Node Discovery**
In this document, a “node discovery” will be defined as discovering a IEEE1394 node and information which will include DDsr(FDP) protocol compliance. Other node information discovery will include manufacturer and ,model number, global unique ID of the nodes which are defined in the IEEE1394-1995 specification.

**Unit (function) Discovery**
In this document, a “unit (function ) discovery” will be defined as discovering a functional unit (or multiple units) within a IEEE1394 node. A unit or unit class will be categorized by the functionality. Examples of functions would be image output units such as printing functions, image source units such as scanning functions.

**Low-level service Discovery**
In this document, “Low-level service discovery” is defined as discovering the availability of the datalinks (=lowest layer above 1394 transaction layer), and the entry points of the datalinks.

**Device Discovery**
In this document, “device discovery” will consist of discovery of the following;

- Node Discovery
- Unit Discovery
- Low-level Service discovery

# 3. OVERVIEW/FUNCTIONAL CHARACTERSTICS

## 3.1 Overview

**This proposal for the DDsr(FDP) Protocol will not define nor specify a transport ,or datalink** , but will define a block in the IEEE1212 configuration ROM for discovering the function units within a node, and then their transports ,or datalinks supported by each function. Examples of function units would be printers, scanners, video cam-corders etc. Examples of datalinks will include XXX over SBP-2 and the AV/C protocol as well as vendor-specific protocols. The DDsr(FDP) Protocol will provide information on unique Ids of each function units, as well as status of each units.

**“DDsr(FDP) Protocol” Capable Nodes**

**Nodes that are capable of the DDsr(FDP) protocol** connecting to a DDsr(FDP) compliant target node will execute the DDsr(FDP) protocol. As a result of this protocol execution, the initiator device can discover the function units within the target node, and datalink capability of each unit, and it’s entry point. If the initiator discovers the preferred function and supported datalink, it will enable it and make logical connection. From this point on until the image source disconnects, the printer can be controlled using the transport enabled.

**Non-“DDsr(FDP) Protocol” Capable Nodes**

**Nodes that are not capable of the DDsr(FDP) protocol** will follow the procedures defined in the communication protocols it supports. This may differ from node to node, so nodes may not be able to establish device discovery or low-level service discovery.

## 3.2 Operational Model

The following section will describe procedures of the DDsr(FDP) IEEE1394 device.

- The operational model
- Interfaces

A IEEE1394 node will support the DDsr(FDP) Protocol and one or more

communication protocols. The DDsr(FDP) Protocol is defined in this document, and examples of communication protocols may be FCP+AV/C and protocols using SBP-2 as the base layer. It may also be a device specific protocol. The DDsr(FDP) Protocol will be used to discover a (DDsr(FDP) Protocol compliant) device node, the first discover the function unit (s) of the node, and secondly find out the communication protocols each unit supports.
A basic discovery scheme using DDsr(FDP) will take the following sequence.

1. A initiator device will look into (read) the defined address in the configuration ROM of the target node to discover a DDsr(FDP) compliant device node.
2. The initiator device will read out information for the address of the Function_unit directory block, which will store information on the function units available in the node.
3. The initiator will read out the Function_unit directory block and retrieve supported function units of the target node, and pointers for leafs blocks of each unit.
4. If further information is needed on a given function unit, the initiator will read out the leaf block of the function unit which pointer was given in the directory block. The function unit leaf block stores a list of datalink(s) supported by that particular unit, and the pointer to the entry of that datalink. It will also store information on unit-uniuqe Ids(example: Plug and play string)
5. Initiator device will enable the datalink that matches the capabilities of both the initiator and target.

## 3.3 Interface

Printers will at least support asynchronous bi-directional interfaces that will be used for the DDsr(FDP) Protocol.

# 4. DDsr(FDP) PROTOCOL DEFINITION

This section will describe the details of the DDsr(FDP) protocol.
The DDsr(FDP) protocol will define and provide 4 main functions:

1. Function 1: Function Unit Discovery
2. Function 2: Low level service discovery of each unit
3. Function 3: Unique ID information retrieval of each unit
4. Function 4: Status retrieval of each unit

In other words, the DDsr(FDP) protocol will allow retrieving information on the functional units within a node, and information on each of the functional units such as the communication protocols each one supports, and some unit Ids unique to each unit.
The DDsr(FDP) protocol will follow the CSR architecture described in ISO/IEC 13213, ANSI/IEEE Std 1212.

Fig 4.1, Fig 4.2 shows the basic architecture and hierarchy of discovery in the DDsr(FDP) protocol

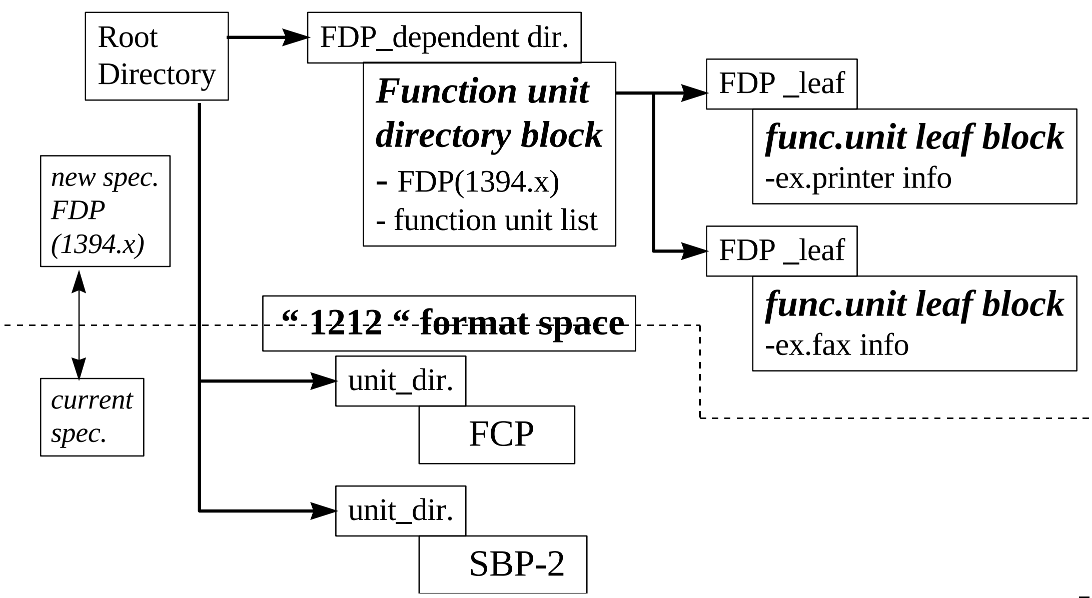



fig 4.1

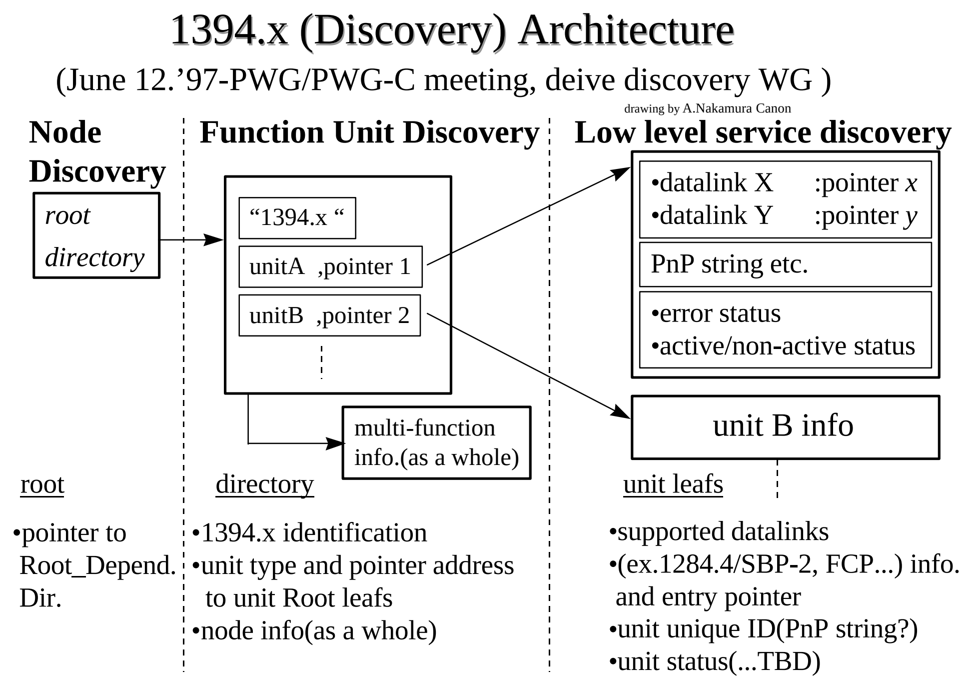

1394.x (Discovery) Architecture
(June 12.'97-PWG/PWG-C meeting, deive discovery WG )
drawing by A.Nakamura Canon
Node Discovery
Function Unit Discovery
Low level service discovery
root directory
"1394.x "
unitA ,pointer 1
unitB ,pointer 2
multi-function info.(as a whole)
•datalink X :pointer x
•datalink Y :pointer y
PnP string etc.
•error status
•active/non-active status
unit B info
root
directory
unit leafs
•pointer to Root_Depend. Dir.
•1394.x identification
•unit type and pointer address to unit Root leafs
•node info(as a whole)
•supported datalinks
•(ex.1284.4/SBP-2, FCP...) info. and entry pointer
•unit unique ID(PnP string?)
•unit status(...TBD)

fig 4.2

In this revision, section4.1 will describe 2 different methods of placement of the DDsr blocks,
1:A dedicated offset entry and directory for DDsr.(similar to the pwr mngment entry..see FYI at bottom of page.)
2:Usage of the Node_dependent_info_offset......ALTERNATIVE METHOD
**This is for study purposes, and 1 method will be defined as the standard in the final specicfication.**

## 4.1 Root Directory - Configuration ROM

The identification of the DDsr(FDP) Protocol compliant device will be defined by a entry in the Root directory in the configuration ROM with a dedicated IEEE1212 key_value.

The DDsr(FDP)_dependent _Directory offset of the Root Directory, of the configuration ROM will be defined by the DDsr(FDP) Protocol.
Address Locations are with respect to the Root Directory which has a base address of:
FFFF F000 0000h

**DDsr(FDP)_directory_entry ...ROOT DIRECTORY**
This field will identify that this device will support the DDsr(FDP) protocol ,and will identify the offset address of the DDsr(FDP) Function_unit directory entry.

| Offset | 0-7 | 8-15 | 16-23 | 24-31 |
|---|---|---|---|---|
| 0438h | D7h or F1h | DDsr_directory offset | | |

**FYI: If the DDsr Protocol shall be part f the IEEE1212 specification, it shall have it's dedicated root directory offset entry using the key_value ranging from 17h-2Fh which is reserved for definition by the CSR Architecture......17h for DDsr(FDP)?**

**FYI: The IEEE1394 Specification for Power Management has it's own (special) root directory offset entry using the key_value of F0h(concationation of 3h and 30h) which 30h-37h is reserved for definition by the bus standard identified in BUS_INFO_BLOCK. (Which is IEEE1394 in this case.)......31h for DDsr(FDP)?**

**(17h concatenated with the key_type value of 3h results in D7h, as noted above.)**
**(31h concatenated with the key_type value of 3h results in F1h, as noted above.)**

## 4.2 DDsr(FDP) Function Unit Directory and Leafs

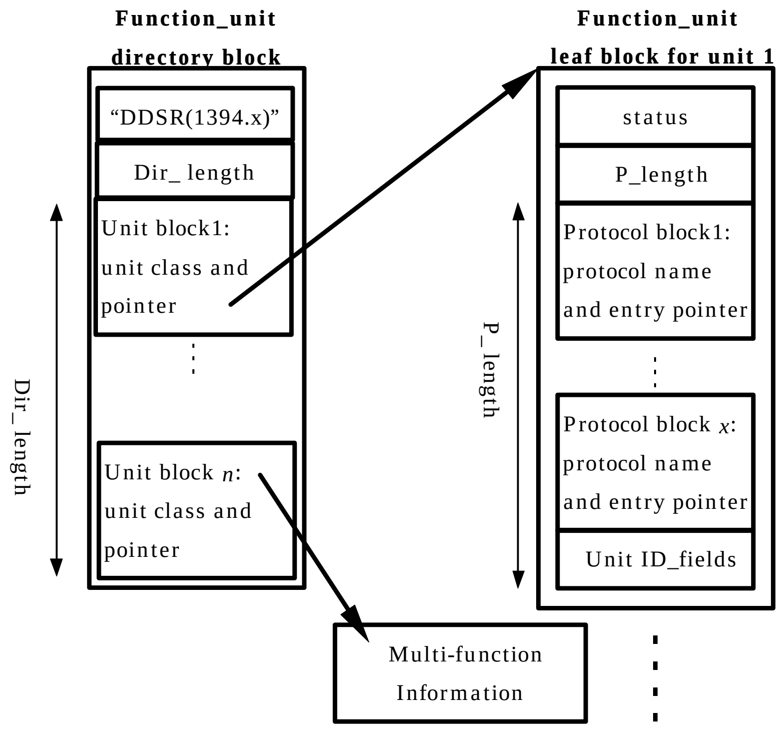
The DDsr(FDP) Register map of a node consists of 2 main parts; 1 Function_unit directory block and 1 or more Function_unit blocks. Readouts of the DDsr(FDP) register map will be done using the read transactions defined in the IEEE1394-1995 specification..

The Function_unit directory block will store information on the function_units within a node. In detail, it will give information on ;

- the function_class of each unit in the node
- pointer to the Function_unit leaf block for each unit

The Function_unit leaf block will store information of each of the function_units within a node. In detail, it will give information on ;

- the protocols supported by the unit
- entry pointer of each of the protocols
- status information of the units




Nodes with multiple function units will have multiple unit blocks, and units supporting multiple datalinks will have multiple protocol blocks.

Information of the multi-function units as a whole (multi-function information) can also be stored in the Functional unit leaf block as well.

### 4.2.1 Function_ unit directory block (unit discovery)

Address offset locations noted in this section are with respect to a base address noted in the Node_Dependent_Info Directory offset in the configuration ROM.

| Offset | R/W | 0-7 | 8-15 | 16-23 | 24-31 |
|---|---|---|---|---|---|
| **0000h** | **R** | **directory length** | | **CRC16** | |
| 0004h | R | **00xxxxxxb** | spec_identifier | | |
| 0008h | R | **00xxxxxxb** | configuration_change_identifier | | |
| 000Ch | R | **00xxxxxxb** | dir_length | | |
| 0010h | R | **01xxxxxxb** | pointer | | |
| 0014h | R | **00xxxxxxb** | fuct._unit_class | | |
| 0018h | | | | | |
| | R | **01xxxxxxb** | pointer | | |
| | R | **00xxxxxxb** | fuct._unit_class | | |

Unit block (0010h–0014h)

*Key_type* *Key_values*

#### 4.2.1.1 Spec_identifier - Key Value : XX

The Spec_identifier field is used to identify compliance with the DDsr(FDP) protocol.

| Field | Bit | Description |
|---|---|---|
| spec_identifier | [8..31] | “DDsr(FDP)” |

#### 4.2.1.2 Configuration_change_identifier- Key Value : XX

The Configuration change identifier field shows a value of a state-change “random” counter. A state-change counter **must** change it’s value when there is a configuration change in the function unit directory or the function unit leaf of any of the functional units.

| Field | Bit | Description |
|---|---|---|
| Configuration change identifier | [8..31] | value of state-change random counter |

#### 4.2.1.3 Dir_length- Key Value : XX

The Dir_length field is used to inform the remaining length of the Function_unit directory block. The value of this field will represent the remaining length of the block in number of quadlets. (refer to above register map)

| Field | Bit | Description |
|---|---|---|
| Dir_length | [8..31] | remaining length of block in quadlets |

#### 4.2.1.4 Unit block (Pointer / Fuct._unit_class) - Key Value : XX

The Pointer field and Fuct._unit_class field are the pair of fields that make up a unit block for each function_unit in the Function_unit directory block. The value of the Fuct._unit_class field will represent the functional class of the unit, and the value of the pointer field will represent the pointer address of the Function_unit leaf block of the unit it represents.

| Field | Bit | Description |
|---|---|---|
| Pointer | [8..31] | pointer address of the Function_unit leaf block |

| Field | Bit | Description |
|---|---|---|
| Fuct._unit_class | [8..31] | functional class of the unit<br>0 : others<br>1 : propriatory node function<br>2 : propriatory function<br>3 : printing function<br>⋮ |

***FYI : From IEEE std 1212 document:***

***Node_Dependent Info***

*Used to provide additional information about the node.*
*The leaf or directory Node_Dependent _Info provodes vendor-dependent information. The format and meaning of this information is dependent on the 48-bit value produced by prepending the 24-bit Node_Vendor_ID value to the 24-bit Node_HW_Version number*

**Key definitions**

*.........The remaining* ***key_value*** *values are reserved as follows;*
*.........$38_{16}$ to $3F_{16}$ are allocated for definition by vendors. Vendor-dependent key_value values may be position - and context -dependent. Within a vendor-dependent directory, the meaning of all key-value parameters is also vendor dependent.*

### 4.2.2 Function_ unit leaf block (low-level service discovery)

Address offset locations noted in this section are with respect to the pointer address of each functional_unit noted in the Pointer field of the Function_unit Directory block. (section4.2.1)

| Offset | R/W | 0-7 | 8-15 | 16-23 | 24-31 |
|---|---|---|---|---|---|
| **0000h** | **R** | **leaf_length** | | **CRC** | |
| 0004h | R | functional unit_status | | | |
| 0008h | R | p_length | | | |
| 000Ch | R | protocol_name | | | |
| 0010h | | | | | |
| 0014h | R | | | | |
| 0018h | R | | | | |
| 001Ch | R | 0h | protocol ID1 | | |
| 0020h | R | 0h | protocol ID2 | | |
| 0024h | R | *Key_type* | entry address | | |
| ⋮ | | ⋮ | | | |
| | R | | | | |
| | R | entry address | | | |
| | R | ID_length | | | |
| | R | unit_info_ID | | | |

Offsets 000Ch–0024h: **Protocol Block**; rows 000Ch through the last entry address span **p_length**; unit_info_ID spans **ID_length**.

4.2.2.1 Functional unit_Status......RAM implementation only **TBD**

The Status field is used to inform the basic (primitive) functional status of the unit.

| Field | Bit | Description |
|---|---|---|
| Status/Active | [30] | 0 : unit is in non-active state<br>1 : unit is in active state |
| Status/Error | [31] | 0 : no error<br>1 : error |
| | [16..29] | reserved |

#### 4.2.2.2 P_length

The P_length field is used to inform the total length of the protocol blocks in this Function_unit leaf block. The value of this field will represent the field length in number of quadlets. (refer to above register map)
If a Protocol block is not used, the value of the P_length will be 0.

| Field | Bit | Description |
|---|---|---|
| P_length | [0..31] | length of protocol blocks in quadlets |

#### 4.2.2.3 Protocol_block(Protocol_name,protocol_ID1,2,entry_address)

In this revision, section4.2.2.3 will describe 3 parts, or 3 ways to specify a protocol in a protocol block.
1: Protocol description in ASCII
2: Protocol description using Unit_Spec_Id, Unit_Sw_Ver.
3: Entry offset related to protocol
**Of the 3 ways, 3:Entry offset will be a minimum requirement. The other necessary fields (1 or 2, or both) to describe the protocols are TBD.**

The Protocol block is comprised of 3 parts;
1) the Protocol_name field
2) the Protocol ID fields, and
3) the entry_address field with a key_type(used to specify the characteristics of the entry_address field).

The **Protocol_name field** is a 4 quadlet field used to specify the supported datalinks of the function_unit in ASCII.

| Field | Bit | Description |
|---|---|---|
| Protocol_name | [0..31] | The name of the datalink supported by the unit in ASCII. |

The **protocol ID fields** consist of 2 fields that when prepended, will specify the datalink ;

1: The **Protocol_ID 1 field** will have the same value used in the **unit_spec_id, node_spec_id, or the module_spec_id** field of the configuration ROM that best describes the datalink. (Usually a unit_spec_id in the unit_directory defined in each datalink specification.)

2: The **Protocol_ID 2 field** will have the same value used in the **unit_sw_version, node_sw_version, or the module_sw_version** field of the configuration ROM that best describes the datalink. (Usually a unit_sw_version in the unit_directory defined in each datalink specification.)

**In case ***_spec_id or ***_sw_version are not provided in a particular node**, the values of Protocol_ID 1 will be the assumed value for unit_spec_id, and Protocol_ID 2 field will be the assumed value for unit_sw_version . Assumed values for unit_spec_id and unit_sw_version are defined in the ISO/IEC 13213, ANSI/IEEE Std 1212 document.

| Field | Bit | Description |
|---|---|---|
| Protocol_ID1 | [8..31] | The unit_spec_id value of the datalink supported by the unit. |
| Protocol_ID2 | [8..31] | The unit_sw_version value of the datalink supported by the unit. |

The **entry_address field** with the **key_type** will inform the node_ID and the entry offset address of the datalink noted above.
The entry_address field for a datalink having a corresponding unit directory shall point to that unit_directory ,in which case the value of key_type shall be 3.

| Field | Bit | Description |
|---|---|---|
| key_type | [0....1] | 0 : immediate value<br>1 : intial-register space offset for a immediate value<br>2 : indirect-space offset for a leaf<br>3 : indirect-space offset for a directory |
| entry_address | [2..31] | address for datalink entry. |

#### 4.2.2.4 ID_length

The ID_length field is used to inform the total length of the Unit_info_ID field for this Function_unit leaf block. The value of this field will represent the field length in number of quadlets. (refer to above register map)

| Field | Bit | Description |
|---|---|---|
| ID_length | [0..31] | length of Unit_info_ID field in quadlets |

#### 4.2.2.5 Unit_info ID

The multi-quadlet Unit_info ID field is used to inform an unique ID of the function unit. The contents of the Unit_info ID field will at least include a function unit ID field that will follow the format of the Device ID field defined in section 7.6 of the IEEE std 1284-1994.

| Field | Bit | Description |
|---|---|---|
| Unit_info ID | [0..31] | unit ID string |

***FYI : From IEEE std 1212 document:***
***from section 8.1.3......Driver and diagnostic identifiers***

*.........The arows in figure 53 illustrate the default values fir various company_id values. For example, when Node_Spec_Id is not provided, its assumed value shall be equal to Node_Vendor Id. Similary when Node_Vendor_id is not provided, its assumed value shall be equal to Module_Vendor_Id.*

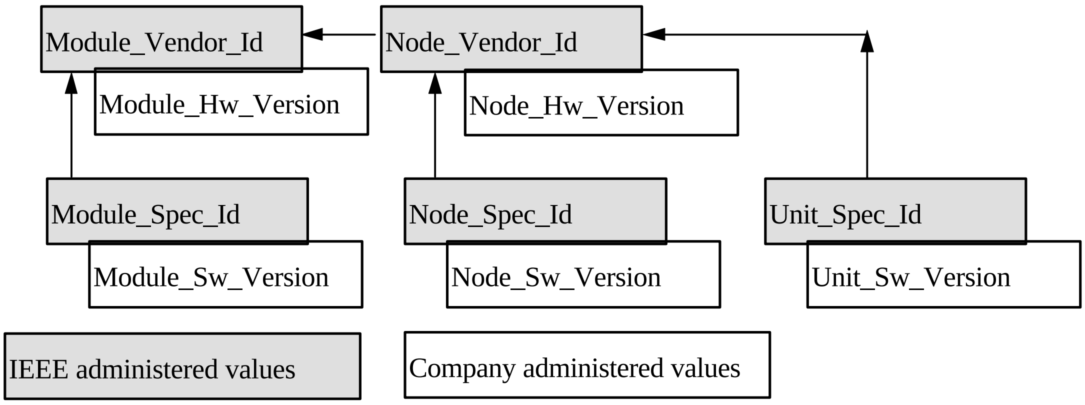



*Fig.53*

*..........The Module_Sw_Version, Node_Sw_Version, and the Unit_Sw_Version values (when concatinated with their respective specifier identifier,such as Module_Spec_Id) are expected to uniquely identify the appropriate I/O driver software for the module, node, unit, respectivel y.*

## 4.3 Bus Reset - Reconnection

There are no DDsr(FDP) protocol reconnection functions for re-establishing connection after a 1394 bus reset. In other words, the transport in progress when bus-reset occurs will be responsible for reconnection after bus reset is cleared.
Naturally ,
transport candidates for the DDsr(FDP) protocol require support for reconnection functions in the case of a 1394 bus reset.

**Bus Reset - Reconnection Requirements**
Values of the fields of the DDsr(FDP) Protocol that will dynamically change;

1. shall not change during bus reset
2. shall be updated by the device upon reconnection with a time limit of 1sec after bus-reset is cleared. Methods of updating are beyond the scope of the proposal.

Devices connecting to DDsr(FDP) compliant nodes shall keep track of the **values of the Configuration-state counter field** for any changes in configuration change of the node.

# 5. IEEE 1394 SPECIFIC ADDRESS SPACE

The IEEE1394 printer compliant with this specification should be compliant with IEEE1394 and IEEE1212 standards. This section will describe the CSR and Configuration ROM locations that the printer will implement. All locations are intended to comply with the IEEE1394 standard.
Address Locations noted in this section are with respect to a base address of:

FFFF F000 0000h

## 5.1 CSR

The printer will implement the following CSR's, as required by the IEEE 1394 standard. :

CORE CSRs

| **offset** | **0-7** | **8-15** | **16-23** | **24-31** |
|---|---|---|---|---|
| 0000h | STATE_CLEAR | | | |
| 0004h | STATE_SET | | | |
| 0008h | NODE_IDS | | | |
| 000Ch | RESET_START | | | |
| 0010h | | | | |
| 0014h | | | | |
| 0018h | SPLIT_TIMEOUT_HI | | | |
| 001Ch | SPLIT_TIMEOUT_LO | | | |

SERIAL BUS DEPENDENT CSRs

| **offset** | **0-7** | **8-15** | **16-23** | **24-31** |
|---|---|---|---|---|
| 0200h | CYCLE_TIME | | | |
| 0204h | | | | |
| 0208h | | | | |
| 020Ch | | | | |
| 0210h | BUSY_TIMEOUT | | | |

## 5.2 CONFIGURATION ROM

The printer will implement the following CONFIGURATION ROM as well as the DDsr(FDP) defined Blocks

**METHOD 1**

BUS INFORMATION BLOCK

| offset | 0-7 | | 8-15 | 16-23 | | 24-31 |
|---|---|---|---|---|---|---|
| 0400h | 04h | | crc_length | rom_crc_value | | |
| 0404h | 31h | | 33h | 39h | | 34h |
| 0408h | **** | rsv | FFh | **** | rsv | |
| 040Ch | node_vendor_id | | | | | chip_id_hi |
| 0410h | chip_id_lo | | | | | |

ROOT DIRECTORY

| offset | 0-7 | 8-15 | 16-23 | 24-31 |
|---|---|---|---|---|
| 0414h | 0009h | | CRC | |
| 0418h | 03h | module_vendor_id | | |
| 041Ch | 17h | module_dependent_info leaf offset | | |
| 0420h | 0Ch | node_capabilities | | |
| 0424h | 08h | node_vendor_id | | |
| 0428h | 09h | node_hw_version | | |
| 0430h | 8Dh | node_unique_id_leaf_offset | | |
| 0434h | **F1h** | **DDsr_dependent__dir_offset** | | |
| 0438h | D1h | unit_directory_offset(s) | | |

NODE UNIQUE ID LEAF

| offset | 0-7 | 8-15 | 16-23 | 24-31 |
|---|---|---|---|---|
| 0000h | 0002h | | CRC | |
| 0004h | node_vendor_id | | | chip_id_hi |
| 0008h | chip_id_lo | | | |