

*Make sure to update all fields on Title page from File->Properties Menu.*

*Once updated right click and select update field to force update*

*\*Note: This text will not print. Do not delete.*

|  |  |
| --- | --- |
|  |  |
| **Authors:** |  |
| **Project:** |  |
| **Date Saved:** |  |

|  |  |
| --- | --- |
| **Reviewed by:** |  |
| **Approved by:** |  |

CONFIDENTIAL

The information contained herein is confidential and shall not be revealed to any third party without the expressed written consent of . The document and all copies shall be returned to Icron promptly upon request by Icron.

Contents

1 Introduction 1

1.1 Purpose 1

1.2 Scope 1

1.3 Terminology References 1

1.4 Document References 1

2 Design Overview 2

2.1 Background Information 2

2.2 System Evolution Description 2

2.3 Current Process 2

2.4 User Characteristics 2

2.4.1 User Problem Statement 2

2.4.2 User Objectives 2

2.5 Proposed Process 2

2.6 Constraints 3

2.7 Design Trade-offs 3

3 Design Architecture 4

3.1 Construction and Maintenance 4

3.2 Current Status 4

3.3 Device Specific Information 4

4 External Module interface 5

4.1 Use model 5

4.2 API/Data Structures 5

5 Detailed Design 7

5.1 Construction and Maintenance 7

5.1.1 Processing 7

5.1.2 Local data structures 7

5.1.3 Inputs 7

5.1.4 Outputs 7

5.2 Current Status 7

5.2.1 Processing 7

5.2.2 Inputs 7

5.2.3 Outputs 8

5.3 Device Specific Information 8

5.3.1 Processing 8

5.3.2 Inputs 8

5.3.3 Outputs 8

6 Test Plan 9

6.1 Unit Tests 9

6.1.1 Construction and Maintenance 9

6.1.2 Current Status 9

6.1.3 Device Specific Information 9

6.2 Integration Plan 9

# Introduction

## Purpose

This document describes the role of the topology tree within the Lionsgate 1 software, how it is used, what information is captured within it and what other modules interact with it. The details of how the USB device topology is built and maintained in the software is covered as well as the status maintained for each device in the topology.

## Scope

The topology tree structure and the information and status stored at each node are covered except for that relating to the packet processing. The packet processing information can be found in the USB Descriptor Parser document.

## Terminology References

## Document References

Universal Serial Bus Specification Revision 2.0, <http://www.usb.org>

a\_lionsgate\sw\Docs\LionsGate Software High Level Design.doc

a\_lionsgate\sw\Docs\ USB Descriptor Parser.doc

# Design Overview

## Background Information

The topology is needed to identify reset and disconnected devices as well identify devices during quick enumeration and it does this by keeping track of the relative position of connected devices. The topology tree holds up to the maximum number of devices the LG1 system can support which is fifteen.

## System Evolution Description

This design replaces previous implementations where storage for a fixed number of ports on a hub was provided, limiting the size of a hub plugged into the system and consuming more memory. This design also adds valuable information about the current status of a device and about the topology itself.

## Current Process

In order to accommodate for hubs up to a maximum number of ports, storage for each potential device on a port is reserved whether a device is plugged in or not. This reserves memory which may never be needed.

## User Characteristics

### User Problem Statement

A removal action is identified through a disconnect event on a port on a hub and a reset device is identified by a port reset request to the hub and port number. The address of the removed or reset device is not given therefore it is necessary to know which device is connected to which hub and on which port. In addition, all downstream devices must be removed or reset as well.

### User Objectives

The design is going to store the topology without making any assumptions about the number of ports on a connected hub nor reserving memory for unused ports as well as saving device information and topology structure between port and root device disconnects.

## Proposed Process

The design uses a binary tree to keep track of where a device is connected in the topology. Because there isn’t any assumption about the number of ports a hub can have, the tree can accommodate a hub with any number of ports. A hub can be connected without having to allocate memory to store device information for all ports or for a predetermined maximum number of ports. And data for ports for which there is no connected device is not stored. The binary tree design also keeps track of which device is connected to which hub and on what port as well as the connections between devices such that any particular device can be found by searching the tree.

The binary tree connections between devices consist of three types, parent, child and sibling. Every node in the tree has a parent and it may have a child and/or sibling. Figure 1(a) shows a sample USB physical structure and its corresponding topology view in Figure 1(b). Dev1, Hub2 and Dev4 are children. Hub1, Dev2, Dev3 and Dev5 are siblings. And Root Hub, Hub1 and Hub2 are parents. The dotted lines represent parent connections and the arrows represent child or sibling connections.

|  |  |
| --- | --- |
|  |  |
| 1. Physical USB view | 1. Topology tree view |

Figure : USB topology in two views

In addition to keeping track of device connections, the topology also stores information and current status about a device. During enumeration packet parsing status is maintained for each device and port status port numbers are stored for hubs. The number of consecutive timeouts is also stored in the topology for a given device. Information consisting of whether a device is a hub, its highest endpoint, and speed is also stored. Saving the high endpoint number for a device is a compromise between storing all the endpoints and storing none. The highest endpoint number allows for code to save time when it needs to check all endpoints for certain conditions.

## Constraints

The topology covers the connections between devices. It does not contain the mapping between logical address and physical USB address assigned by the host nor endpoints for each device. When a new device is added to the topology tree the first available position under the parent is used, whether that be child or sibling of a child. And recursion cannot be used to traverse the tree due to register window constraints.

## Design Trade-offs

The main goals of the design are to minimize the amount of memory used to store the topology as well as minimize time taken to traverse the tree when searching for devices.

# Design Architecture

The design architecture can be grouped into three main areas; these are construction and maintenance, current status, and device specific information.

## Construction and Maintenance

Tasks for construction and maintenance are adding and removing, clearing and resetting a device as well as finding devices within the tree.

## Current Status

Current status of devices in the topology includes things such as current setup response, bytes parsed, timeouts, and current port status.

## Device Specific Information

Information stored for each device includes speed, highest endpoint number, and whether the device is a hub or not. Also, topology tree connections are maintained through parent, port, child, and sibling data.

# External Module interface

## Use model

This design is intended to be used by having its API’s called directly. The system is responsible for calling the adding, disconnecting, removal, and resetting functions when the conditions for these events occur.

The private data that stores the parsing status is referenced by direct calls to the API by the packet parser, and timeouts and port status data by the device manager. The hub and high endpoint information is accessed directly from the API by various modules requiring the information.

## API/Data Structures

The following API’s are exposed to external modules.

ResultT DEVICE\_Init(void);

void DEVICE\_Show(void);

ResultT DEVICE\_ShowTopology(void);

uint32 DEVICE\_ReadTopTableMemoryForLeo(uint32 offset);

// Construction and maintenance

void DEVICE\_Add(uint8 logicalAddr, uint8 parentLA, uint8 portOnParent);

void DEVICE\_Process(uint8 nodeLA, boolT includeStartNode, ActionCodesT action);

ResultT DEVICE\_ResetAll(boolT disconnectStatus);

boolT DEVICE\_GetDisconnectStatus(uint8 logicalAddr);

void DEVICE\_SetDisconnectStatus(uint8 logicalAddr, boolT status);

uint8 DEVICE\_GetDisconnectedLogicalAddress(void);

ErrorCodesT DEVICE\_FindPort(uint8 port, uint8 parentLA, uint8 \*pLogicalAddr);

// Information

void DEVICE\_ClearHighEndpoint(uint8 logicalAddr);

void DEVICE\_UpdateHighEndpoint(uint8 logicalAddr, uint8 endpoint);

void DEVICE\_GetHighEndpoint(uint8 logicalAddr, uint8 \*pMaxEndpt);

uint8 DEVICE\_IsHub( uint8 logicalAddr);

void DEVICE\_SetHub( uint8 logicalAddr);

void DEVICE\_ClearHub(uint8 logicalAddr);

void DEVICE\_SetSpeed(uint8 logicalAddr, UsbSpeedT speed);

UsbSpeedT DEVICE\_GetSpeed(uint8 logicalAddr);

// Current status

uint16 DEVICE\_GetbMaxPacketSize0FrameOffset(uint8 logicalAddr, uint16 bytesRead);

void DEVICE\_SetbMaxPacketSize0(uint8 logicalAddr, uint8 maxSize);

SetupResponseT \* DEVICE\_GetCurrentSetupResponse(uint8 logicalAddr);

void DEVICE\_SetBytesParsed(uint8 logicalAddr, uint16 bytes);

uint16 DEVICE\_GetBytesParsed(uint8 logicalAddr);

uint16 DEVICE\_GetRequestedLength(uint8 logicalAddr);

void DEVICE\_SetRequestedLength(uint8 logicalAddr, uint16 requestedLength);

uint8 DEVICE\_GetTimeouts(uint8 logicalAddr);

void DEVICE\_SetTimeouts(uint8 logicalAddr, uint8 timeouts);

uint8 DEVICE\_GetPortStsPort(uint8 logicalAddr);

void DEVICE\_SetPortStsPort(uint8 logicalAddr, uint8 port);

| **API** | **Usage** |
| --- | --- |
| DEVICE\_Add | When the Control Queue parser has determined that a set address request was made |
| DEVICE\_Process (action = remove) | On receiving endpoints for a device that was previously reset |
| DEVICE\_Process (action = disconnect) | When the Device Manager has received a port status response indicating a disconnected port  When the Control Queue parser has identified a clear port request |
| DEVICE\_Process (action = reset) | When the Control Queue parser has identified a port reset request |
| DEVICE\_ResetAll | On a bus reset (all devices in the topology are reset) |
| DEVICE\_FindPort | To identify which device is on a particular port for DEVICE\_Reset in all of the above three cases |
| DEVICE\_GetbMaxPacketSize0FrameOffset  DEVICE\_SetbMaxPacketSize0  DEVICE\_GetCurrentSetupResponse  DEVICE\_SetBytesParsed  DEVICE\_GetBytesParsed | Used by the packet parser for accessing the current parsing status |
| DEVICE\_GetTimeouts  DEVICE\_SetTimeouts | Used by the Response Queue handling to store the number of consecutive timeouts seen |
| DEVICE\_GetPortStsPort  DEVICE\_SetPortStsPort | Used by Device Manager to store the port number of the last port status request |

The entire topology tree is contained within the design. Each node in the topology tree consists of packet parsing information, vital device data, disconnect and port status, and timeout storage. The topology tree is an array of sixteen nodes.

typedef struct DeviceTopology {

// Setup packet parsing info

SetupResponseT currentSetupResponse; // child structure to keep track of the current setup response

uint16 requestedLength;

uint16 bytesParsed; // How many bytes have been parsed in the current response packet

uint8 bMaxPacketSize0Mask; //(8 is 0x7), (16 is 0x1F), etc.

// Port of the last port status request

uint8 currentPortSts;

// Vital device info

uint8 portOnParent; // The port number of the parent hub to which this device is connected.

uint8 parentLA; // The logical address of the parent device

uint8 childLA; // The logical address of the child device that is connected to this device

uint8 siblingLA; // The logical address of the sibling device to this device

boolT isHub; // Is this device a hub?

uint8 highEndpoint; // Highest endpoint number for the device

UsbSpeedT speed; // HS/FS/LS

// for OHCI standby-resume handling

boolT disconnectStatus; // Has the device been disconnected

// Error handling & now only for debug information

uint8 timeOuts; // How many timeouts in a row has this device received

}DeviceNodeT;

# Detailed Design

## Construction and Maintenance

### Processing

On the addition of a device DEVICE\_Add is called to setup the new node with its parent logical address and port number on the parent it is connected on and to insert the new node in the appropriate place in the topology.

Anytime a device marked as reset writes endpoints to the XSST, DEVICE\_Process is called with action set to Remove and all devices downstream from it in the topology are removed. On a port clear request and on a port status response where the port has been disconnected DEVICE\_ Process is called with action Disconnect and all devices downstream in the topology are reset and marked as disconnected. On a port reset DEVICE\_ Process is called with action set to Reset and all devices downstream in the topology are reset. When DEVICE\_Process is called it traverses the topology marking devices downstream of the passed in device to be reset, disconnected or removed respectively. Depending on the situation the passed in device may also be marked.

DEVICE\_FindPort is used to identify the device that matches the parent and port passed into the function. Every child and sibling under the parent is looked at for the matching port number. When there is no matching port number the device does not exist in the topology.

### Local data structures

DEVICE\_Process uses a list to keep track of which nodes to work on as well as the number of nodes in the list.

### Inputs

DEVICE\_Add is passed the logical address of the new device, parent logical address, port number on parent and speed of the device. DEVICE\_Process takes in the logical address of the node to start from as well as which action to perform (Remove, Disconnect or Reset) and whether to include the starting device node.

typedef enum

{

eRemove = 0, // Remove nodes from the topology

eDisconnect, // Disconnect nodes (which is essentially doing a reset and marking the LA for reuse)

eReset, // Reset a device

eActionMax

}ActionCodesT;

DEVICE\_FindPort takes in the parent logical address and the port number on the parent to search for.

### Outputs

DEVICE\_FindPort passes back the logical address of node that was found or zero when no device was found and an error message of port-not-found is returned.

## Current Status

The APIs related to packet parsing are discussed in the USB Descriptor Parser document.

### Processing

DEVICE\_GetPortStsPort and DEVICE\_SetPortStsPort are used to track the port number of the last port status request received on the Control Queue for a particular parent hub. Correspondingly, DEVICE\_GetTimeouts and DEVICE\_SetTimeouts are used to track the number of consecutive timeouts seen for a given device in the Response Queue.

### Inputs

DEVICE\_GetPortStsPort takes in the parent logical address to retrieve the port number from and DEVICE\_SetPortStsPort takes in the parent logical address as well as the port number to store.

DEVICE\_GetTimeouts takes in the logical address of the device to retrieve the timeout count from and DEVICE\_SetTimeouts takes in the logical address as well as the timeout count to store.

### Outputs

DEVICE\_GetPortStsPort passes back the port number of the last port status request on that parent hub. DEVICE\_GetTimeouts passes back the current timeout count.

## Device Specific Information

### Processing

DEVICE\_ClearHighEndpoint, DEVICE\_UpdateHighEndpoint, and DEVICE\_GetHighEndpoint are used to access the highest endpoint number for a particular device in the topology tree. DEVICE\_UpdateHighEndpoint is called when an endpoint is being written to the XSST and if the new endpoint number is higher than the currently stored one, the high endpoint number is updated.

DEVICE\_IsHub, DEVICE\_SetHub, and DEVICE\_ClearHub are used to manage whether a device is hub or not.

DEVICE\_SetSpeed stores the speed of a device and DEVICE\_GetSpeed returns the speed of a device.

### Inputs

DEVICE\_ClearHighEndpoint and DEVICE\_GetHighEndpoint take in the logical address of the device to access for the high endpoint number. DEVICE\_UpdateHighEndpoint also takes in the endpoint number.

DEVICE\_IsHub, DEVICE\_SetHub, and DEVICE\_ClearHub take in the logical address of the device as input.

DEVICE\_SetSpeed takes the logical address and speed of the device and DEVICE\_GetSpeed takes the logical address of the device being queried.

### Outputs

DEVICE\_GetHighEndpoint passes back the stored high endpoint number for the given device. DEVICE\_GetSpeed returns the speed of the requested device.

# Test Plan

## Unit Tests

### Construction and Maintenance

Adding, removing, disconnecting and resetting could be tested individually by creating a test device topology and calling the respective API with the information for the device and then verifying the resulting topology. Searching for devices could also be performed on the test topology by simply calling the API with appropriate parameters and then ensuring the correct device or no device was found.

Important corner cases such as first or last device added, as well as trying to add more devices than allowed could easily be tested by setting up an initial test topology tree.

### Current Status

Packet parsing current status testing would be done in conjunction with the testing for the packet parser. See the USB Descriptor Parser document.

### Device Specific Information

Again the test topology could be used to confirm collection and setting of the correct device information into the topology by calling the various API and verifying the expected data.

## Integration Plan

The topology as described in this document has already been implemented and tested as part of the system.