Mobile Manageability

Intel Corporation

September 30, 1997

Course Objectives

- To understand the required elements that make a managed mobile platform
- To understand what the Intel Mobile Component Instrumentation (IMCI) SDK provides to the CI Developer.
- To learn the basics of writing instrumentation using the IMCI SDK as demonstrated today.

intel®

Agenda

- Mobile Manageability Overview
- WfM for Mobile
- Platform Elements
- Software Stack
- IMCI SDK Overview
- Writing Mobile Instrumentation
- Summary
- Call to Action

intel®

Mobile Manageability Overview

- The goal of Intel's Wired for Management (WfM) initiative is to make PCs universally manageable and universally managed.
  - A consistent baseline of management capabilities and function delivered in the platform
  - A consistent target for application developers

Reduce TCO!

intel®

WfM for Mobile

- *Instrumentation*
  - Guaranteed set of management information available to management apps
    - DMI 2.0 + std groups
- *Remote New System Setup*
  - Boot from network to install std load
    - Preboot eXecution Environment
- *Wake on LAN**
  - Ability to wake platform to perform after-hours maintenance
    - WOL silicon
- *Power Management*
  - HW, BIOS interfaces to allow OS to manage platform and subsystems power policy
    - ACPI

intel®

WfM for Mobile
Instrumentation

- Same requirement as other Baseline-capable PCs with these additions:
  - DMTF Mobile Supplement to the System Standard Group
  - Dynamic instrumentation support for hot pluggable mobile devices
    - Must not require a system reboot
    - Examples: PC Cards, Hot Docking

intel®

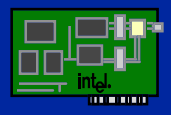
WfM for Mobile

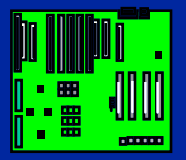
Remote New System Setup

Recommended

(a.k.a Preboot eXecution Environment)

- **Recommended with the understanding that these are the PXE Agent Implementation choices available:**
  - **Boot diskette**
  - **Adapter ROM on NIC (docking station implementation)**
  - **BIOS on the motherboard**





intel®

WfM for Mobile
Wake On LAN*

- Recommended for LAN connected notebooks only
  - Mobile usage model does not lend itself to after-hours maintenance which Wake on LAN* enables
  - Mobile Wake on LAN* silicon not available

intel®

WfM for Mobile
Power Management

- ACPI compliant platform components required
- ACPI OS recommended when it becomes available

intel®

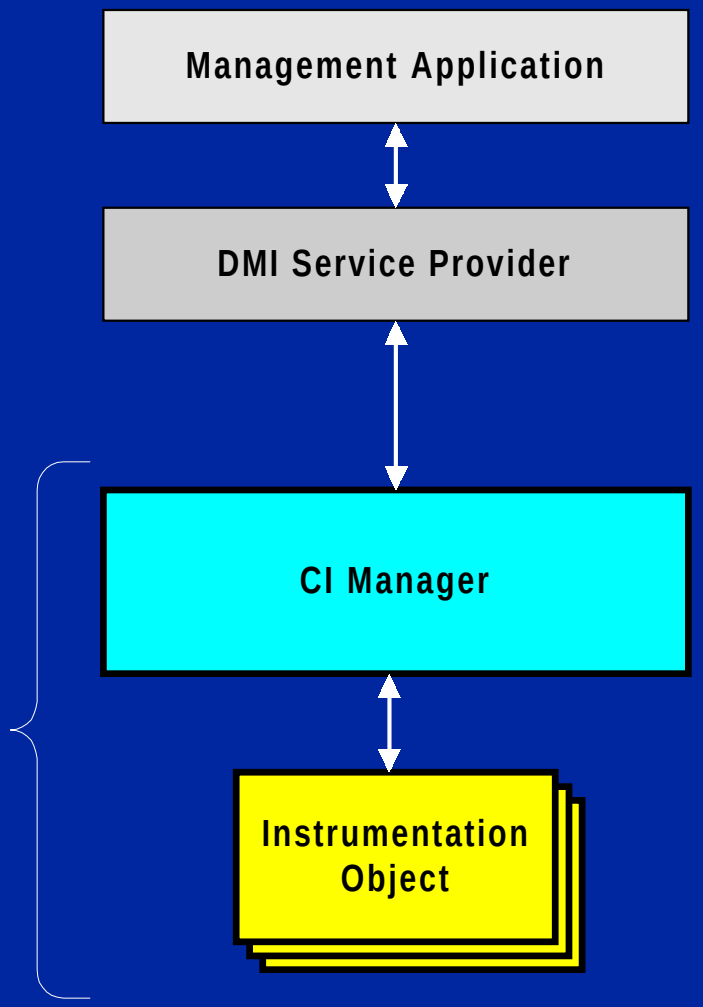
Platform Elements
Hardware/Firmware

- Thermal sensor(s) to detect and report over temperature conditions
  - LM75* integral to Pentium(R) Processor Mobile Module
  - Add other sensors as necessary (Motherboard, PC Card slots, Battery)
  - Choose sensors appropriate for Mobile (low voltage, low current)
- SMBIOS v2.0 or higher
  - Provides platform information to management driver software stack

intel®

Software Stack

- DMI Stack
  - DMI 2.0 Service Provider
  - DMI manageability Instrumentation that conforms to Wired for Management Baseline v1.1



intel®

Intel Mobile Component Instrumentation (IMCI) SDK

- Contents
  - CI Manager
  - Instrumentation Object Framework
  - Debug Viewer
  - WfM 1.1 Compliant Sample Instrumentation Objects
  - Reference Manual

intel®

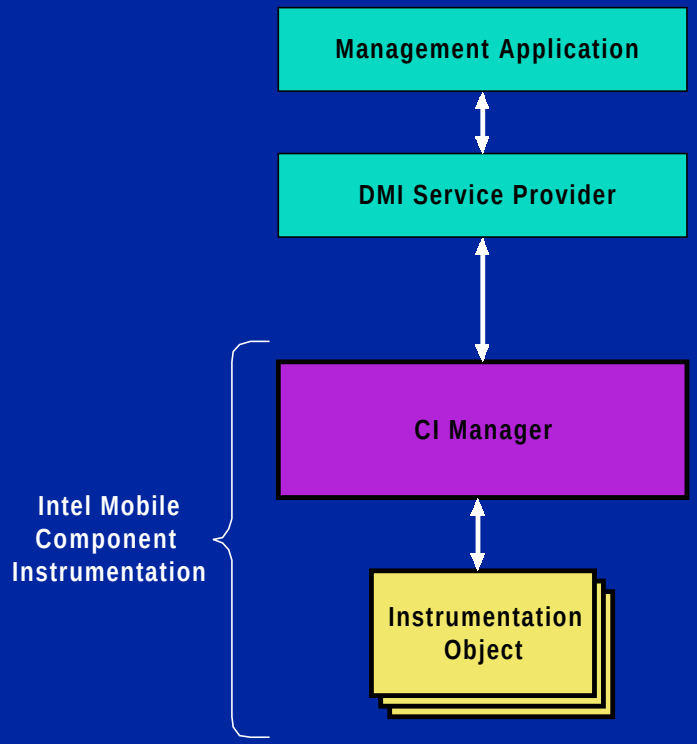
IMCI SDK Overview

- Features
  - Simple to write instrumentation
    - Shields developers from row management and service provider registration
    - Simple, flexible way to handle events
  - Fully supports hot docking and hot pluggable devices
  - Resource-Smart
    - Memory usage controlled by instrumentation object caching algorithm
    - Power-friendly (verified by Intel Power Monitor)

intel®

IMCI SDK Overview

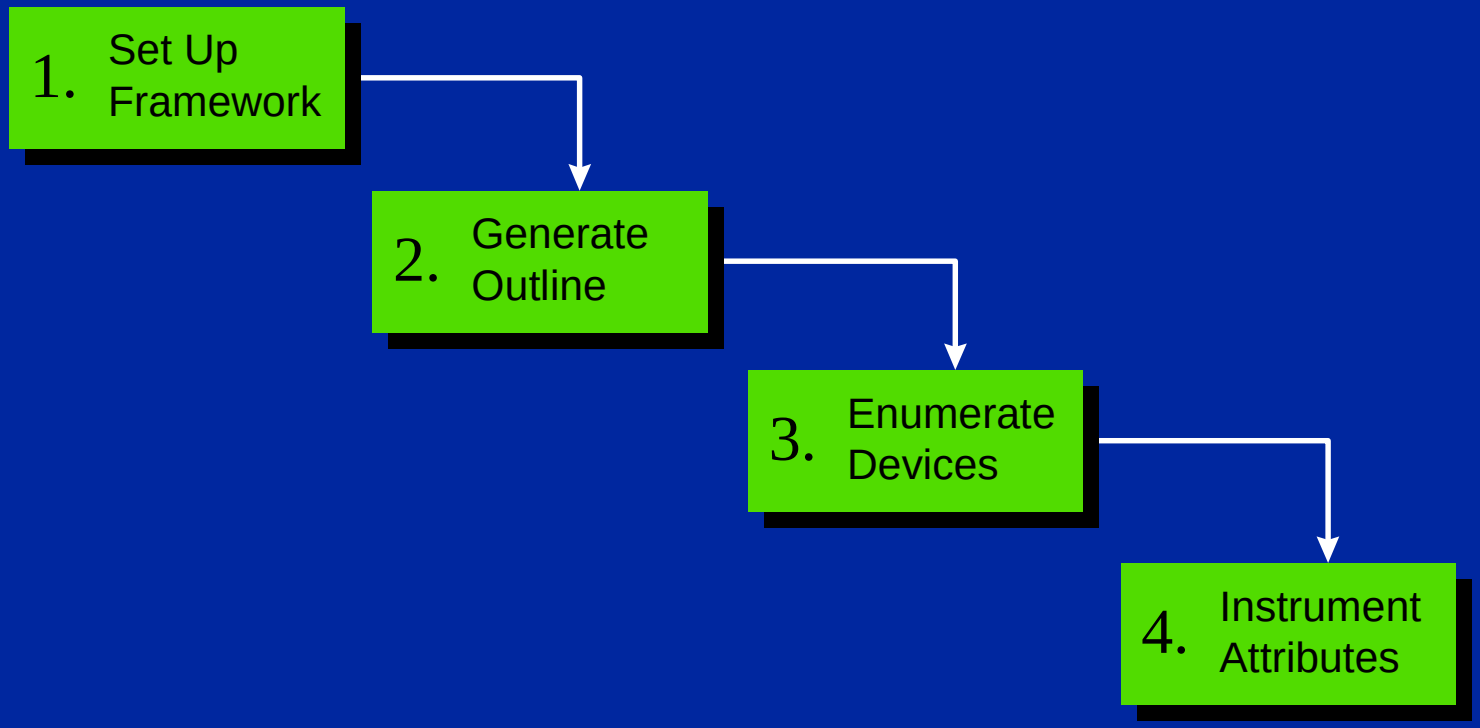
- **WE WROTE 10,000 LINES OF CODE SO YOU DON'T HAVE TO**



intel®

Writing Instrumentation

- What was once complicated, is now just four simple design steps



intel®

Writing Instrumentation

- Let's write an object that:
  - Responds to serial port requests
  - Supports multiple serial ports
  - Supports dynamically changing serial ports
- We'll do this by adding code to the `ServiceRequest()` and `EnumerateDevices()` methods
- Let's follow the design steps and beginning with Step 1

intel®

Writing Instrumentation

1. Set Up Framework

Writing Instrumentation

1. Set Up Framework

- **Process**
  - **Copy framework files**
  - **Setup identifying information in code**
  - **Build and register instrumentation object**
  - **Create and import registry file**

intel®

Writing Instrumentation

2. Generate Outline

Writing Instrumentation

2. Generate Outline

- Process
  - Add if & switch statements
  - Return the index attribute
  - Return a string attribute
- Demo
- Test for Understanding

intel®

Writing Instrumentation

2. Generate Outline

- Add if & switch statements
  - Check group class name
  - Switch on attribute number

```
if (0 == lstrcmpi("DMTF|Serial Ports|003",
          (const char*)szGroupClassName))
{
  switch (attributeNumber)
  {
     :
   }
}
```

intel®

Writing Instrumentation

2. Generate Outline

- Return the index attribute
  - Return the `ROWID` reference macro for attribute 1 using `MakeReference()`

```
case 1:
   hErr = MakeReference(ROWID, pDataStruct);
   break;
```

intel®

Writing Instrumentation

2. Generate Outline

- Return a string attribute
  - Return “Hello Ports!” for attribute 4 using `MakeDisplayString()`
  -

```
case 4:
   hErr = MakeDisplayString(“Hello Ports!”,
                            pDataStruct);
   break;
```

intel®

Writing Instrumentation

2. Generate Outline

- **Demo**
  - **Use a DMI browser to make sure we're talking to the CI Manager and DMI SP**

intel®

Writing Instrumentation

2. Generate Outline

- **Test for Understanding**
  - How could this instrumentation object handle requests for additional groups?
  - (Add extra “if…else” statements to compare the group class names provided by the CI Manager)
  - If the CI Manager manages rows, how are row index attributes returned?
  - (Call the MakeReference() helper function with the `ROWID` macro)
  - Where does the CI Manager look to determine what instrumentation objects exist?
  - (The Registry)

intel®

STRETCH BREAK

Writing Instrumentation

3. Enumerate Devices

intel®

Writing Instrumentation

3. Enumerate Devices

- Process
  1. Get the devices from the Windows* Configuration Manager
  2. Parse the list of devices
  3. Return devices to the CI Manager
- Why is this so cool?
- Demo
- Test for Understanding

intel®

Writing Instrumentation

3. Enumerate Devices

- Get the devnodes from the Windows* Configuration Manager
  - Use `GetDevnodeListForClassNames()` to get a linked list of devnodes for the PnP Class Name “Ports”

```
typedef struct DEVNODEINFO
 {     ULONG ulDevnodeID;
       char szHardwareKey[REG_KEY_SIZE];
       char szClassName[CLASSNAME_SIZE];
       char szDescription[DEV_DESC_SIZE];
       char szService[CLASSNAME_SIZE];
       struct DEVNODEINFO* pNext;
 } DEVNODE_INFO, *PDEVNODE_INFO;
```

intel®

Writing Instrumentation

3. Enumerate Devices

- Parse the list of devnodes (continued)
  - Inside this loop….

```
pTmp = pHead;
while (NULL != pTmp && SUCCEEDED(hErr))
{
        //
        // Devices are compared against and
        // returned (showed on next slide)
        //
        pTmp = pTmp->pNext;
}
```

intel®

Writing Instrumentation

3. Enumerate Devices

- Parse the list of devnodes
  - While looping through the list, compare each devnode to the kind of device for which we want to return data
  - Look for the “COM” string (as in COM1)

```
char* szReturn = strstr(pTmp->szDescription, "COM"));
 if (NULL != szReturn)
 {
     // Found a COM port so return a device to the
     // CI Manager (shown on next slide)
 }
```

intel®

Writing Instrumentation

3. Enumerate Devices

- **Return devices to the CI Manager**

```
lstrcpy(m_szDeviceType, "Serial");
m_DeviceID.ulDeviceID = pTmp->ulDevnodeID;
hErr = AddDeviceToList();
```

- **Why is this so cool?!?**

intel®

Writing Instrumentation

3. Enumerate Devices

- `EnumerateDevices()` gets called
  - First time object is registered
  - Whenever a PnP event for “Ports” occurs
- The CI Manager keeps track of which devices were added/removed
- One registration method that works all the time…Cool!

intel®

Writing Instrumentation

3. Enumerate Devices

- Demo
  - Recompile and test
  - Notice the `RowID`, configured earlier, changes with each row

intel®

Writing Instrumentation

3. Enumerate Devices

- **Test for Understanding**
  - **What does `GetDevnodeListForClassNames()` return?**
    (A list of devnodes that match the specified PnP class name)
  - **Who handles the details of tracking which devices are new and which ones are gone?**
    (The CI Manager)
  - **What does the CI Manager send to an instrumentation object to identify a device during a service request?**
    (The DeviceID structure)

intel®

Writing Instrumentation

4. Instrument Attributes

intel®

Writing Instrumentation

4. Instrument Attributes

- Returning real data for a device
  - Now that we have a devnode, let's get the real information
    - IRQ from Configuration Manager
    - IO Address from Configuration Manager
  - Let's look at some source in `ServiceRequest()`...

intel®

Writing Instrumentation

4. Instrument Attributes

- Returning real data for a device
  - How to get the IRQ and IO addresses

```
hErrFromGetHWInfo = GetHWInfo(
        pCIOComponent->m_OsType,
        pCIOComponent->m_DeviceID.ulDevnodeID,
        NULL,
        &hwStruct);
 if (SUCCEEDED(hErr))
        hErr =
MakeInteger(hwStruct.wIRQNumber[0],
                        pDataStruct);
```

- Recompile...

intel®

Writing Instrumentation

4. Instrument Attributes

- Demo
  - Note that IRQ and IO addresses are properly returned

intel®

Writing Instrumentation

4. Instrument Attributes

- But wait! There's more!
  - Dynamic Devices
  - CI Manager handles the code for this step!
  - Let's change our configuration and watch what happens to our IRQ and IO addresses for different rows

intel®

Writing Instrumentation

4. Instrument Attributes

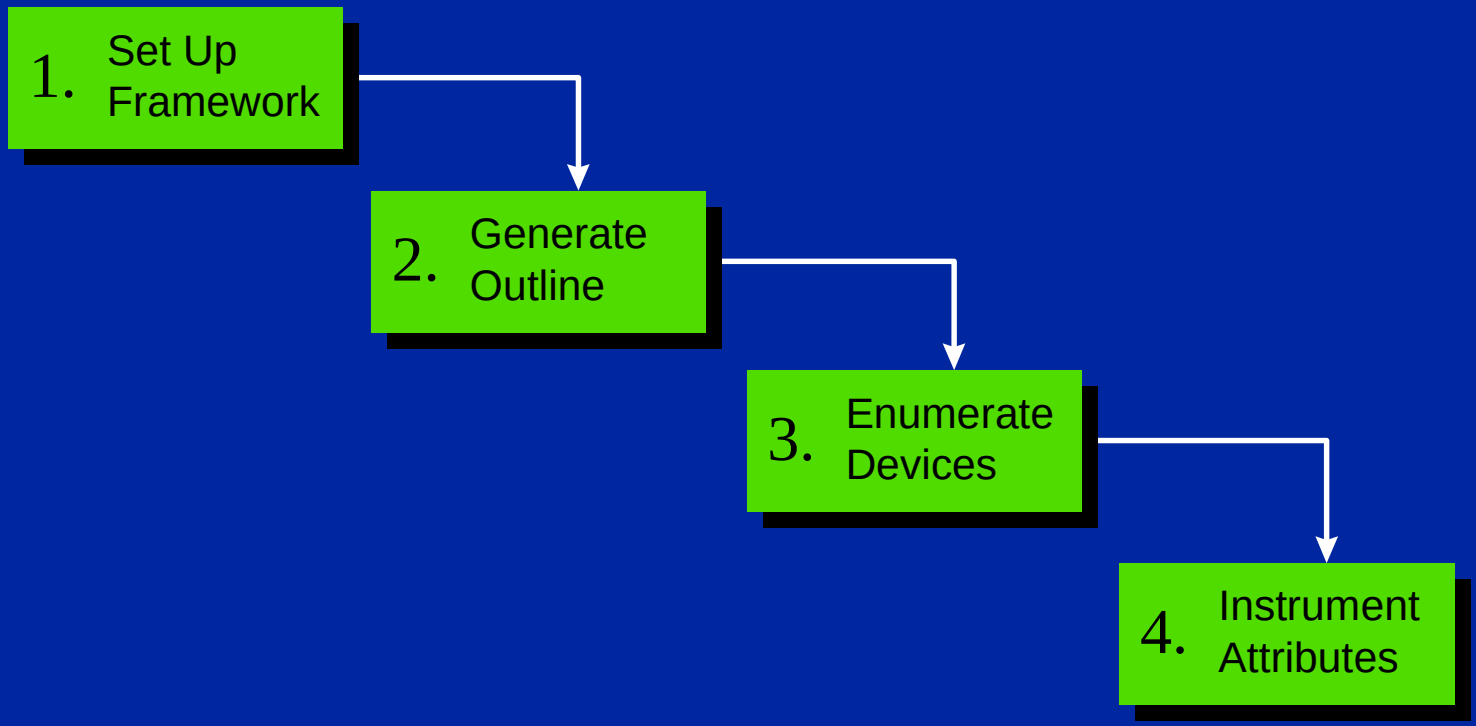
- **Test for Understanding**
  - **Where can more information be obtained about the Framework Helper API's?**
  - **(The Reference Manual on the CD)**
  - **How much extra work is required to make the device dynamic?**
  - **(None!)**
  - **Okay, now what should you do?**

    (Read the next slides, endure the marketing pitch)

intel®

Summary

- WE WROTE 10,000 LINES OF CODE SO YOU DON'T HAVE TO
- YOU'VE SEEN HOW EASY IT IS



intel®

Call to Action

- Install the SDK
- Develop WfM Compliant instrumentation for your mobile PCs
- Watch the web for updates to the SDK (http://www.intel.com/managedpc)
- We'll be here for questions

intel®

Collateral

- On the conference CD
  - Intel Mobile Component Instrumentation SDK
    - Source for Ports sample (IOPorts)
    - Reference Manual
  - WfM Design Guide
    - Intel Mobile Component Instrumentation SDK

intel®