OBIS Internet Enabled Device Discovery Process

# SUMMARY

OBIS Connection (herein *ObiCon*) needs a way to discover when new internet enabled devices become available.  This needs to work without the software having to continually poll for devices.

With USB, Windows delivers explicit messages to the App when USB devices are added or removed from the system.  No such standard protocol exists for the Internet, so we have to provide one.

Fortunately a simple solution is as follows.  It’s easily implemented by a few dozen lines of code running in a separate thread in each internet enabled product.

# INTRODUCTION

The *Discovery* protocol is asymmetrical.  The world is divided into *Devices* (like Multi-Controller) and *Clients* (like ObiCon).  Devices provide services via the internet, while Clients utilize those services.  Clients strive to keep track of available Devices, while Devices merely respond to direct requests.

Devices and Clients may be connected or disconnected from the LAN at any time (or more simply turned on or off).  Clients need to be made aware of new devices when they become available and (optionally) when they become unavailable.

Clients make use of *Services* provided by Devices, when available.  Services are generally handled by a TCP/IP connection, independent of the Discovery process, and otherwise are beyond the scope of this specification.

# DEVICE PROTOCOL

When Devices come online they announce their presence with a **Howdy** message.

Subsequently they monitor the net for **Hoosier** messages*[Note 1]*.  Upon receipt they re-issue a **Howdy** message.

Just before normal termination they send a **Goodbye** message.

# CLIENT PROTOCOL

Clients continuously listen for **Howdy** and **Goodbye** messages, and thereby maintain a list of known active devices.  This is essentially how ObiCon presently keeps track of USB devices. *[Note 2]*

Clients when they startup issue a **Hoosier** message.

The existing *Re-Scan Device List* UI function will be wired to also send a **Hoosier** message, in addition to re-scanning for USB and RS-232 devices.

# EXCLUSION LIST

By default Clients (ObiCon) will listen for and list as available all Coherent devices it discovers on the LAN.  This is the desired out-of-the-box behavior.

However, for larger installations, customers will likely want to have multiple Clients, each controlling different groups of Devices.  Thus each Client will maintain a Device Exclusion list of devices to ignore.  ObiCon already implements this for USB and RS-232 devices.

From the main window, it will be sufficient to click on a device sub-panel, right click, and select the “Exclude This Device” from the context menu.  Then that device will be excluded from subsequent Client activity.

Elsewhere [TBD] there will be a button to bring up the Exclusion List.  This is to review and edit the exclusion list.  It’ll show all known and previously excluded Devices, with a check box to exclude or include in future activity.

The Exclusion List is stored on a per-user basis.

To facilitate setup in larger facilities, we’ll add commands to Save and Load *Configurations* to and from a file.  Configurations will include the contents of the Exclusion List and possibly other things.  Files then may be shared among multiple users.  Once a configuration file is loaded, those settings are remembered for that user until subsequently changed.  The contents of a Configuration file beyond the Exclusion List are TBD.

# MORE FINE PRINT

The protocol is LAN based.  It’s not possible to discover devices across LAN boundaries.

Devices have types.  To determine device types we will use the same company/product IDs defined for USB.

Howdy messages include the following:

* Device IP address (unique to the LAN)
* Device Name (user-assigned, not necessarily unique)
* Device Company and Product ID
* Local temperature and humidity

Hoosier messages may optionally request responses from

* All devices recognizing the protocol
* Devices with a specific Company ID
* Devices with a specific Company ID and Product ID

Discovery is UDP based, which is efficient, stateless and connectionless.  The Port numbers are 100151 for Client to Device messages and 100251 for Devices to Clients.

When replying to a Hoosier message, Devices will generate a random number between one and, say, 500; then they’ll wait that many milliseconds before sending their reply.  This is to avoid swamping Clients with too many messages at once.

Services are TCP/IP based connections.  Devices and Clients should make every effort to cleanly close any open Service connections before terminating.

Devices and Clients may infer a Goodbye event from certain pathological behavior observed on an open Service connection.

***Note 1:*** Hoosier is the nickname for Indiana state residents.  By the time settlers got to Indiana a new type of door latch had been invented.  It automatically locked when the door was closed.  When knocking on doors further east, people were accustomed to hearing “come in”.  But when they got to Indiana, they instead heard “who’s there?”, as Hoosiers otherwise had to get up and open the door to let someone in and didn’t want to get off their asses for just anybody.

***Note 2:*** Discovery on RS-232 ports entails periodically interrogating each port for a plausible response.  No way around it but not how we want to work with USB or LAN.