x1815 - Hosts Info

Version 2

Overview

This feature allows managing host information on devices with multihost capabilities. Every device channel is linked to a different host, one active at a time, on eQuad/Unifying, USB, BT or BLE buses.

This feature supports reading host bus information, saving and reading host os version, moving host connection across channels, and deleting host connections.

It might happen, for technical reasons, that some device does not implement all functions, therefore function [0]getFeatureInfo() returns a capability mask detailing which functionalities are implemented. This approach has been preferred to splitting every sub-functionality into separate mini features.

Bus naming convention

- **BT** : Bluetooth BR/EDR "classic".
- **BLE**: Bluetooth LE "Low Energy".
- **eQuad**: Any eQuad/Unifying version.
- **USB** : Any USB bus version. **Reserved for futur use.**

Notice: Definitions for USB buses are reserved for future use.

Functions and Events

[0] getFeatureInfo() → capabilityMask, numHosts, currentHost

Get the number of hosts in the registry and the currently active host index (logically the caller). Also returns a capability_mask for the device.

Parameters

none

Return

[2bytes] capabilityMask

Sub-features implemented by the device. A bit is 1 if associated capability is available, else it has to be 0.

- **Get Name**: getHostFriendlyName() command is supported.
- **Set Name**: setHostFriendlyName() command is supported.
- **Move Host**: moveHost() command is supported.
- **Delete Host**: deleteHost() command is supported.
- **OS Version**: setHostOsVersion() and getHostOsVersion() commands are supported.
- **eQuad HD**: getHostDescriptor() can return an eQuad descriptor.
- **USB HD**: getHostDescriptor() can return an USB descriptor. Reserved for futur use.
- **BT HD**: getHostDescriptor() can return a BT descriptor.
- **BLE HD**: getHostDescriptor() can return a BLE descriptor.

[1byte] numHosts

The number of hosts / channels, including paired and unpaired.

[1byte] currentHost

The current host channel index [0..numHosts - 1].

Table 1. getFeatureInfo() response packet format.

byte \ bit	7	6	5	4	3	2	1	0
0	-	-	-	Set Os Version	Delete Host	Move Host	Set Name	Get Name
1	-	-	-	-	BLE HD	BT HD	USB HD	eQuad HD
2	numHosts							
3	currentHost							
4..15	reserved							

Errors

none

[1] getHostInfo(hostIndex) → hostIndex, status, busType, numPages, nameLen, nameMaxLen

Get a particular host basic information.

Parameters

[1byte] hostIndex

Channel / host index [0..numHosts-1]. 0xFF = Current Host.

Table 2. getHostInfo() request packet format.

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1..15	reserved							

Return

[1byte] hostIndex

host index.

[1byte] status

Pairing status (0=empty slot, 1=paired)

[1byte] busType

The host bus type can be:

- 0 - Undefined: the Host entry is unused.
- 1 - eQuad.
- 2 - USB.
- 3 - BT.
- 4 - BLE.
- 5 - BLE Pro (Bolt)

NOTE BLE Pro uses the same characteristics as BLE, only the busType value change.

[1byte] numPages

Number of Host Descriptor pages available for the host, depending on busType (may be less, including 0, if the corresponding information could not be collected):

- Undefined: always 0.
- eQuad: always 0.
- USB: Reserved for futur use.
- BT: 2 pages with data collected from Device ID Service Record.

- BLE: 1 or more pages with data collected from GAP and DIS services.

[1byte] nameLen

The byte length of the host friendly name (without null terminator).

[1byte] nameMaxLen

The maximum byte length of the host friendly name (without null terminator).

Table 3. getHostInfo() response packet format.

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1	status							
2	busType							
3	numPages							
4	nameLen							
5	nameMaxLen							
6..15	reserved							

Errors

- **(2) InvalidArgument:** Invalid host index.

[2] getHostDescriptor(hostIndex, pageIndex) → hostIndex, busType, pageIndex, hostDescriptor

Get a host descriptor page. The host descriptor is the collection of data describing the host profile (VendorID, Version), collected from its transport channel and (BT, USB..) and depending on it.

Parameters

[1byte] hostIndex

Channel / host index. 0xFF = Current Host.

[1byte] pageIndex

Index of the host descriptor page to query (0..numPages-1 returned by getHostInfo()).

Table 4. getHostDescriptor() request packet format.

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1	pageIndex							
2..15	reserved							

Return

Returns a **HostDescriptor** packet structure. The header describes the transport protocol, the rest of the structure depends on it. The total structure shall fill the 16 bytes packet, all unknown or unused bits set to 0. An unused (unpaired) host entry should have all bytes 0, including transportId.

busType

- 0 - undefined: the Host entry is unused.
- 1 - eQuad.
- 2 - USB. Reserved for futur use.
- 3 - BT.
- 4 - BLE.
- 5 - BLE Pro (Bolt).

Table 5. getHostDescriptor() response header packet format

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1	busType				pageIndex			
2..15	reserved							

BT and BLE Descriptor page 0

The descriptor contains the host Bluetooth 6-bytes address (starting from left-most byte when displayed: 01:02:03:04:05:06).

Table 6. getHostDescriptor() response BT and BLE section page 0 packet format

byte \ bit	7	6	5	4	3	2	1	0
2	-	-	-	-	-	-	-	Valid Address
3	Bluetooth Address 1							
4	Bluetooth Address 2							
5	Bluetooth Address 3							
6	Bluetooth Address 4							
7	Bluetooth Address 5							
8	Bluetooth Address 6							
9..15	reserved							

BT Descriptor page 1

The BT descriptor page 1 contains fields from the host’s BT transport:

- **validFields** - 1 byte. A bit-field indicating which of the following fields are valid.
- **SpecificationID** - 2 bytes, field ID 0x0200 as of BT specification.

- **VendorID** - 2 bytes, field ID 0x0201 as of BT specification.
- **ProductID** - 2 bytes, field ID 0x0202 as of BT specification.
- **Version** - 2 bytes, field ID 0x0203 as of BT specification.
- **PrimaryRecord** - 1 bit, field ID 0x0204 as of BT specification.
- **VendorIDSource** - 2 bytes, field ID 0x0205 as of BT specification.

Table 7. getHostDescriptor() response BT classic section page 1 packet format

byte \ bit	7	6	5	4	3	2	1	0
2	-	-	Valid VendorID Source	Valid PrimaryRecord	Valid Version	Valid ProductID	Valid VendorID	Valid SpecificationID
3	-	-	-	-	-	-	-	PrimaryRecord
4	SpecificationID MSB							
5	SpecificationID LSB							
6	VendorID MSB							
7	VendorID LSB							
8	ProductID MSB							
9	ProductID LSB							
10	Version MSB							
11	Version LSB							
12	VendorIDSource MSB							
13	VendorIDSource LSB							
14..15	reserved							

BLE Descriptor pages 1 and subsequent

The BLE descriptor starting at page 1 are chunks of a unique continuous buffer containing a varying list of UUID data. Thus pages 1 to numPages-1 shall be collected and assembled into one contiguous buffer before decoding.

The contiguous buffer contains a varying list of UUID descriptors, including zero (16 bits key plus data, for most UTF-8 text). The buffer starts with the number of UUIDs and the total byte size of UUID descriptors, followed by the sequence of UUID decriptors. A UUID descriptor contains the UUID 16bits value, the data byte len and the data itself.

```
struct page1_buffer {
    uint8_t  uuidsNum;   /* number of UUIDs */
    uint16_t uuidsSize;  /* following UUIDs data byte size */
    .
    struct uuid_desc_s {
      uint16_t uuid;           /* UUID */
      uint8_t  dataSize;       /* following data byte size */
      uint8_t  data[dataSize]; /* data[size], can span on page2, 3... */
    } uuid_desc[uuidsNum];
}
```

Table 8. getHostDescriptor() response BLE section page 1 packet format

byte \ bit	7	6	5	4	3	2	1	0
2	Number of uuids							
3	uuids total byte size MSB							
4	uuids total byte size LSB							
5	uuid[0] MSB							
6	uuid[0] LSB							
7	size[0]							
8	data[0] byte[0]							
..	..							
9+size[0]	data[0] byte[size[0]-1]							
10+size[0]	uuid[1] MSB							
11+size[0]	uuid[1] LSB							
12+size[0]	size[1]							
13+size[0]	data[1] byte[0]							
..	..							
13+size[0]+size[1]	data[1] byte[size[1]-1]							
..	..							

Errors

- **(2) InvalidArgument:** Invalid host index or page index.
- **(4) HardwareError:** Data could not be retrieved.

[3] getHostFriendlyName(hostIndex, byteIndex) → hostIndex, byteIndex, nameChunk

Get a Host Friendly Name chunk. Can be null if the host channel is not paired.

The Host Friendly Name is the name provided by the host when establishing the link with the device. This name is usually meant to identify the host for the user.

The full ASCII name byte length is given by getHostInfo().

Parameters

[1byte] hostIndex

Channel / host index. 0xFF = Current Host.

[1byte] byteIndex

Index of the first host name byte to copy (0..nameLen-1 returned by getHostInfo()).

Table 9. getHostFriendlyName() request packet format

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1	byteIndex							
2..15	reserved							

Return

[1byte] hostIndex

Channel / host index.

[1byte] byteIndex

Same as parameter.

[14bytes] nameChunk

The host name chunk, copied from full host name byteIndex’th byte, padded with null bytes '\0' if the copied string is shorter than the payload size (HPPLong: 16 bytes).

Table 10. getHostFriendlyName() response packet format.

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1	byteIndex							
2	nameChunk[0]							
..	..							
15	nameChunk[13]							

Errors

- **(2) InvalidArgument:** Invalid host index.
- **(5) NotAllowed:** Operation not permitted/not implemented (see "capabilityMask").

[4] setHostFriendlyName(hostIndex, byteIndex, nameChunk) → hostIndex, nameLen

Write a host name chunk, starting at byteIndex. Existing string is overwritten, extended or shorten by the chunk, considering that resulting Host Friendly Name new byte length is byteIndex + strlen(chunk), truncated at maxLen maximum allowed length.

Host Friendly name is ASCII encoded, which doesn’t contain null byte.

Parameters

[1byte] hostIndex

Channel / host index. 0xFF = Current Host.

[1byte] byteIndex

Index of the first host name byte to write (0..strlen-1).

[14bytes] nameChunk

The host name chunk to write, padded with null bytes '\0' if it is shorter than the payload size (HPPLong: 16 bytes).

Table 11. setHostFriendlyName() request packet format.

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1	byteIndex							
2	nameChunk[0]							
..	..							
15	nameChunk[13]							

Return

[1byte] hostIndex

Channel / host index.

[1byte] nameLen

Resulting name len.

Table 12. setHostFriendlyName() response packet format.

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1	nameLen							
2..15	reserved							

Errors

- **(2) InvalidArgument:** Invalid host index.
- **(4) HardwareError:** Data could not be written.
- **(5) NotAllowed:** Operation not permitted/not implemented (see "capabilityMask").

[5] moveHost(hostIndex, newIndex) → void

Change a given host index by associating it to a new index. Other hosts indexes might be shifted up or down.

Parameters

[1byte] hostIndex

Channel / host index to move.

[1byte] newIndex

New index of the channel.

Table 13. moveHost() request packet format.

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1	newIndex							
2..15	reserved							

Return

none

Errors

- **(2) InvalidArgument:** Invalid old or new host index.
- **(4) HardwareError:** Data could not be retrieved or written.
- **(5) NotAllowed:** Operation not permitted/not implemented (see "capabilityMask").

[6] deleteHost(hostIndex) → void

Clear the hostIndex Host entry (undo pairing, clear host info), ready for a new pairing. If the host is the current host, connection is lost after disconnection.

Parameters

[1byte] hostIndex

Channel / host index to delete.

Table 14. deleteHost() request packet format.

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1..15	reserved							

Errors

- **(2) InvalidArgument:** Invalid host index.
- **(4) HardwareError:** Data could not be deleted.
- **(5) NotAllowed:** Operation not permitted/not implemented (see "capabilityMask").

[7] getHostOsVersion(hostIndex) → hostIndex, OsType, OsVersion, OsRevision, OsBuild

Implemented if GetOsVersion bit set in capability bits.
Read Host OS Type and Version, saved previously by SW. Can be null.

Parameters

[1byte] hostIndex

Channel / host index to query.

Table 15. getHostOsVersion() request packet format.

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1..15	reserved							

Return

[1byte] hostIndex

Channel / host index.

[1byte] OsType

Enumerated values (defined in the same order as x4531 Platform Descriptor OS bit field):

- 0: Unknown
- 1: Windows
- 2: WinEmb
- 3: Linux
- 4: Chrome
- 5: Android
- 6: MacOS
- 7: IOS

[1byte] OsVersion

Os 1st Version number.

[2bytes] OsRevision

Os 2nd Version number [Big Endian].

[2bytes] OsBuild

Os 3rd Version number [Big Endian].

Table 16. getHostOsVersion() response packet format.

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1	osType							
2	osVersion							
3	osRevision MSB							
4	osRevision LSB							
5	osBuild MSB							
6	osBuild LSB							
7..15	reserved							

Errors

- **(2) InvalidArgument:** Invalid host index.
- **(4) HardwareError:** Data could not be retrieved.
- **(5) NotAllowed:** Operation not permitted/not implemented (see "capabilityMask").

[8] setHostOsVersion(hostIndex, OsType, OsVersion, OsRevision, OsBuild) → void

Implemented if SetOsVersion bit set in capability bits.
Write Host OS Type and Version. Can be null.

Note: FW does only store this information, but does neither use nor analyze it. Usage of this data is fully upon SW responsibility.

Parameters

[1byte] hostIndex

Channel / host index to write.

[1byte] OsType

Refer to getHostOsVersion() [OsType].

[1byte] OsVersion

Os 1st Version number.

[2bytes] OsRevision

Os 2nd Version number [Big Endian].

[2bytes] OsBuild

Os 3rd Version number [Big Endian].

Table 17. setHostOsVersion() request packet format

byte \ bit	7	6	5	4	3	2	1	0
0	hostIndex							
1	osType							
2	osVersion							
3	osRevision MSB							
4	osRevision LSB							
5	osBuild MSB							
6	osBuild LSB							
7..15	reserved							

Return

none

Errors

- **(2) InvalidArgument:** Invalid host index.

- **(4) HardwareError:** Data could not be written.
- **(5) NotAllowed:** Operation not permitted/not implemented (see "capabilityMask").

ChangeLog

- Version 2: Add BLE Pro busType.
- Version 1: Add BLE descriptor pages.
- Version 0: Initial version.