AN989: OBEX OPP AND OBEX FTP PROFILES

iWRAP APPLICATION NOTE

Thursday, 06 April 2017

Version 1.4

VERSION HISTORY

Version	Comment
1.0	First version
1.1	Fixed Disconnect command syntax from figure in chapter 3.3.1
1.2	Tidied up tables, added OBEX error codes
1.3	Sending a file using FTP updated
1.4	Added Business Card Pull Feature using PULLCARD command

TABLE OF CONTENTS

1 Introduction

This application note discusses Bluetooth Object Push Profile (OPP) and Bluetooth File Transfer Profile (FTP) their advantages and how these profiles can be utilized. Also practical examples are given how the OPP and FTP are used with the iWRAP firmware.

1.1 Object push profile

OPP defines the roles of push server and push client. These roles are analogous to and must interoperate with the server and client device roles that GOEP defines.

A common scenario would be file transfer from mobile phone to PC or another mobile phone. The OPP defines two roles, that of a Push client and a Push server unit:

- Push Server – This is the device that provides an object exchange server (in other words, it is the entity that receives a file).
- Push Client – This is the device that pushes objects to the Push Server. (Supported by iWRAP starting from version 3.1)

OPP channel works on top of RFCOMM connection and the protocol used in the link is OBEX.

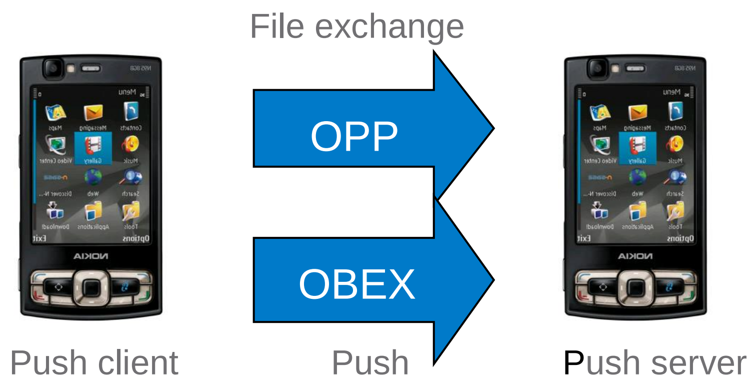


Figure 1: Typical HFP use case

1.2 File transfer profile

FTP provides the mechanisms to browse, send, receive and manipulate files on a remote device.

The FTP specification defines two roles:

- Client – Initiates connection, pushes and pulls files to and from the server. Must be also able to interpret the OBEX Folder listing Format.
- Server – Target device which needs to provide object exchange server and folder browsing capabilities using OBEX Folder Listing format. (Not supported by iWRAP)

File Transfer profile is essentially the same profile as Object Push profile while also providing possibility to fetch files from the server device and moving and browsing the server's file system.

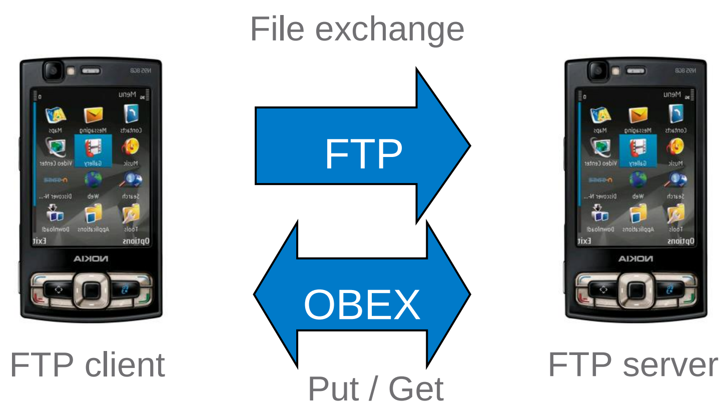


Figure 2: Typical FTP use case

2 iWRAP firmware overview

iWRAP is an embedded firmware running entirely on the RISC processor of WT12, WT12 and WT32 modules. It implements the full *Bluetooth* protocol stack and many *Bluetooth* profiles as well. All software layers, including application software, run on the internal RISC processor in a protected user software execution environment known as a Virtual Machine (VM).

The host system can interface to iWRAP firmware through one or more physical interfaces, which are also shown in the figure below. The most common interfacing is done through the UART interface by using the ASCII commands that iWRAP firmware supports. With these ASCII commands, the host can access *Bluetooth* functionality without paying any attention to the complexity, which lies in the *Bluetooth* protocol stack. GPIO interface can be used for event monitoring and command execution. PCM, SPDIF, I2S or analog interfaces are available for audio. The available interfaces depend on the used hardware.

The user can write application code to the host processor to control iWRAP firmware using ASCII commands or GPIO events. In this way, it is easy to develop *Bluetooth* enabled applications.

On WT32 there is an extra DSP processor available for data/audio processing.

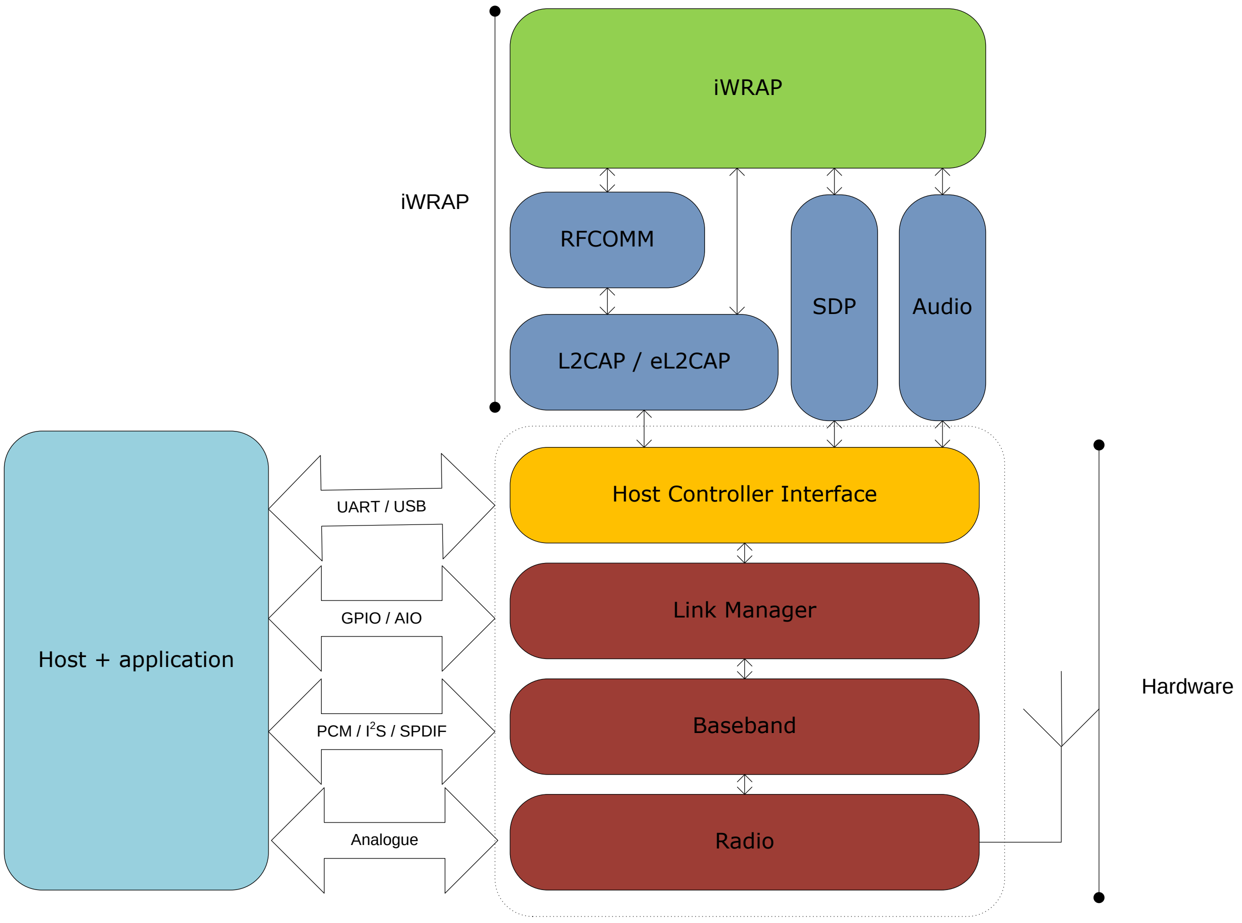


Figure 3: iWRAP Stack

In the figure above, a WRAP THOR *Bluetooth* module with iWRAP firmware could be connected to a host system for example through the UART interface. The options are:

- If the host system has a processor, software can be used to control iWRAP by using ASCII based commands or GPIO events.
- If there is no need to control iWRAP, or the host system does not need a processor, iWRAP can be configured to be totally transparent and autonomous, in which case it only accepts connections or automatically opens them.
- GPIO lines that WRAP THOR modules offer can also be used together with iWRAP to achieve additional functionality, such as Carrier Detect or DTR signaling.
- Audio interfaces can be used to transmit audio over a *Bluetooth* link.

3 Using OPP with iWRAP

This chapter instructs the OPP usage and configuration with the iWRAP firmware.

3.1 Configuration

3.1.1 OPP client

Since OPP client establishes the connection there are no SDP records that would need to be added. Only requirement is to have the iWRAP in Multiplexing Mode (MUX). A reset is recommended after enabling the MUX mode.

Below is an example how to enable OPP client mode.

```
SET CONTROL MUX 1
Link 255: RESET
```

To start using the module in MUX mode, use of BGTerminal software is recommended.

3.1.2 OPP Server

OPP Server mode is enabled with command "**SET PROFILE OPP {*service_name*}**"

service_name	This parameter configures user friendly description of the device. Neither special characters nor white spaces are allowed. Service name **ON** enables the profile with the default name.

Finally a reset is needed to for the OPP Server profile to become active.

Below is an example how to enable OPP Server mode.

```
SET PROFILE OPP ON
RESET
```

3.2 Service discovery

Bluetooth technology enables wireless service discovery, so you can find out the capabilities the remote device supports. Wireless service discovery uses the Bluetooth Service Discovery Profile (SDP).

With iWRAP the service discovery is performed with command: "**SDP {*bd_addr*} {*uuid*}**".

bd_addr	Bluetooth device address of the remote device.
uuid	Universally unique identifier. Refers to the Bluetooth profile one wants to discover. For OPP the ***uuid*** is 1105.

Below is an example how to perform a service discovery for OPP device.

```
SDP 00:07:80:93:0c:aa 1105

SDP 00:07:80:ff:50:05 < I SERVICENAME S "OBEX Object Push" > < I PROTOCOLDESCRIPTORLIST <
< U L2CAP > < U RFCOMM I 02 > < U OBEX > > >

SDP
```

OBEX Object Push	= Service name
02	= RFCOMM channel for OPP

3.3 Connection

3.3.1 Sending a file - OPP Client

The OPP connection establishment to remote device can be done with a **CALL** command:

“**CALL {*bd_addr*} 1105 OPP**”

bd_addr	Bluetooth device address of the remote device.

Below is an example how to set up an OPP connection from iWRAP.

```
LINK 255: CALL 00:07:80:93:0c:aa 1105 OPP
LINK 255: CALL 0
LINK 255: CONNECT 0 OPP 2
LINK 255: OBEX 0 READY
```

The regular CALL and CONNECT events are received if the connection is successful. Soon after the CONNECT event also “OBEX {link_id} READY” message should be received from the iWRAP. Now it is possible to start sending files.

To send files we first need to define the file size, name and MIME type. This is done using **PUT** command:

“PUT {*file_size*} {*MIME_type*} {*filename*}”

file_size	File size in bytes
MIME_type	MIME type of the file
filename	filename for the file that you are sending

A list of different possible MIME types can be found from http://www.iana.org/assignments/media-types/

All OBEX header strings are sent in UTF-16 over the air, but to maintain compatibility with ASCII, iWRAP interprets filenames in UTF-8, since the 128 ASCII characters have the same byte representation in UTF-8 as they do in ASCII. UTF-8 is a Unicode encoding in 8 bit blocks. To send extended ASCII characters (for example), they must be encoded in UTF-8.

For example, if the filename is "abcäöü", it should be sent to iWRAP as the hex sequence 0x61 0x62 0x63 0xc3 0xa4 0xc3 0xb6 0xc3 0xbc, where the first three bytes are the same as the ASCII representations of 'a', 'b' and 'c'. The other three characters are 2-byte UTF-8 representations. The extended ASCII representation of 'ä' would be 0x84 in hex, which is a UTF-8 continuation byte and an invalid UTF-8 character by itself.

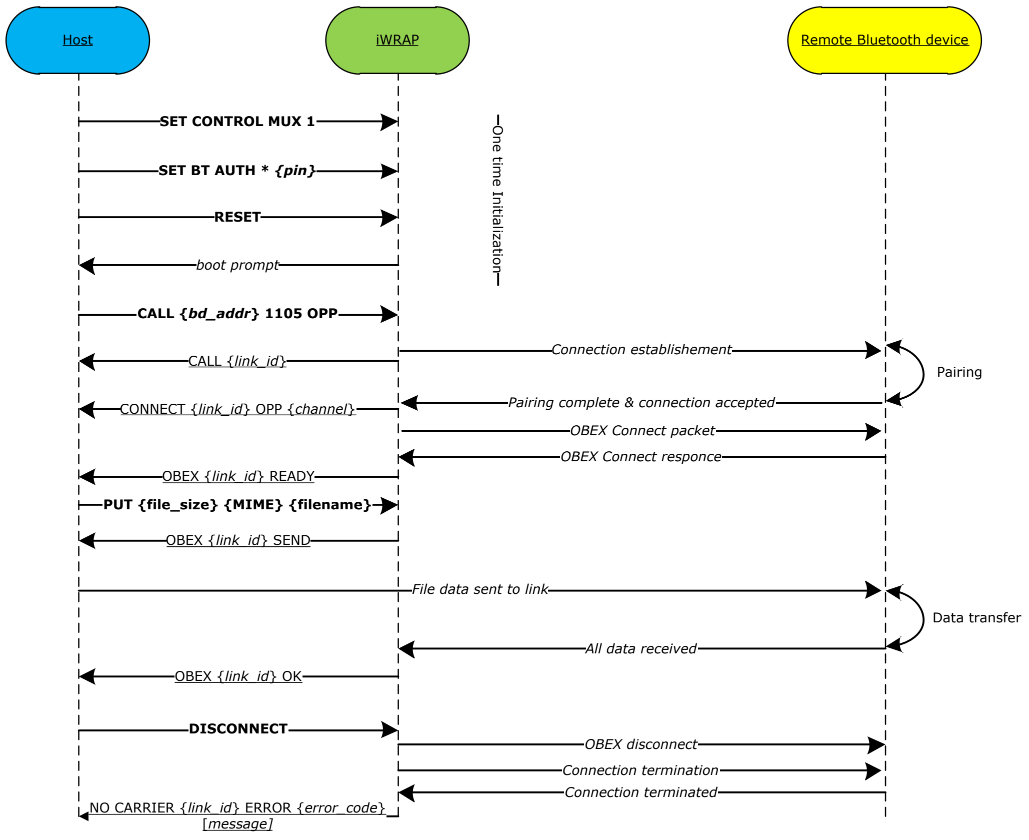
A UTF-8 character table can be found in: http://www.utf8-chartable.de/unicode-utf8-table.pl?number=512; it can be used find the UTF-8 representations for characters that cannot be represented in ASCII.

After the file parameters are defined and “OBEX {link_id} SEND” event is received we can start sending the actual data. This can be done by sending file_size bytes to the Bluetooth link whose indentifier (link_id) was received in the CALL and CONNECT events as the first parameter. In our example this identifier was 0 so the same will be also used in the following continuing example:

```
LINK 255: PUT 10 text/plain testfile.txt
LINK 255: OBEX 0 SEND
LINK    0: 0123456789
LINK 255: OBEX 0 OK
```

NOTE: If you have larger file than single MUX frame can reliably fit inside which is about 200 bytes, you need to split the data into multiple MUX frames.

If you wish to send more files you can rotate this PUT – SEND routine as many times as you like. When you want to close the connection use the DISCONNECT command.



3.3.2 Business Card Pull Feature - OPP Client

Devices that support the business card pull service shall store the owner’s business card in the OBEX Default Get Object. Some devices (e.g. public devices) might hold information in the owner's business card that is relevant to the device rather than to the owner of the device. The Default Get Object does not have a name; instead it is identified by its type. To provide application level interoperability, both the OPP Client and the OPP Server shall support the vCard 2.1 content format.

To perform a Business Card Pull from an OPP Server the OPP connection establishment to remote device should be done with a **CALL** command:

“**CALL {*bd_addr*} 1105 OPP**”

bd_addr	Bluetooth device address of the remote device.

Next **PULLCARD** command should be used. If OPP Server supports Business Card Pull Feature, OPP Client receives file content in vCard 2.1 format. The OBEX transfer is interpreted as shown in the table below.

Field explanation	PUT	Length	Filedata field ID	Filedata length	Filedata
Not last packet Last packet	0x90 0xa0	***x*** ***x***	0x48 0x49	***z*** ***z***	
Field size in bytes	1	2	1	2	***z***-3
Mandatory/Optional	M		M		
Packet length	***x*** bytes				

For more information please see IrDA Object Exchange Protocol specification [2].

3.3.3 Receiving a file - OPP Server

When another device initiates OPP connection to the module you will receive RING event from the iWRAP interface.

```
RING 0 00:18:42:f1:a5:4d 2 OPP
```

Where first parameter indicates the link_id given for the connection and third parameter indicated the local service channel where the OPP service is running.

After receiving the RING event you should see the actual OBEX packets arriving. The OBEX transfer is interpreted as shown in the table below. All field IDs that you don’t understand you can ignore. Essential part is in field which starts with byte 0x48 or 0x49 if the whole file fits into a single field.

In OBEX, all multi-byte values are sent most significant byte first, e.g. 0x1234 is transmitted as 0x12, 0x34.

Field explanation	PUT	Length	Filename field ID	Filename length	Filename	Filedata field ID	Filedata length	Filedata
Not last packet	0x02	***x***	0x01	***y***		0x48	***z***	
Last packet	0x82	***x***	0x01	***y***		0x49	***z***	
Field size in bytes	1	2	1	2	***y***	1	2	***z***-3
Mandatory/Optional	M		O			O/M		
Packet length	***x*** bytes							

For more information please see IrDA Object Exchange Protocol specification [2].

4 Using FTP with iWRAP

This chapter describes using FTP with iWRAP.

4.1 Configuration

4.1.1 FTP client

iWRAP supports only the FTP client mode. Since the FTP client establishes the connection no SDP record is necessary to present and therefore no SET PROFILE setting is needed. The only requirement is to have the iWRAP in Multiplexing Mode (MUX). A reset is recommended after enabling the MUX mode.

Below is an example how to enable FTP client mode.

```
SET CONTROL MUX 1

Link 255: RESET
```

To start using the module in MUX mode, use of BGTerminal software is recommended.

4.2 Service discovery

Bluetooth technology enables wireless service discovery, so you can find out the capabilities the remote device supports. Wireless service discovery uses the Bluetooth Service Discovery Profile (SDP).

With iWRAP the service discovery is performed with the command: "**SDP {*bd_addr*} {*uuid*}**".

bd_addr	Bluetooth device address of the remote device.
uuid	Universally unique identifier. Refers to the Bluetooth profile one wants to discover. For FTP the ***uuid*** is 1106.

Below is an example how to perform a service discovery for an FTP device.

```
SDP 00:07:80:93:0c:aa 1106

SDP 00:07:80:ff:50:05 < I SERVICENAME S "OBEX File Transfer" > < I PROTOCOLDESCRIPTORLIST
< < U L2CAP > < U RFCOMM I 02 > < U OBEX > > >

SDP
```

OBEX File Transfer	= Service name
02	= RFCOMM channel for FTP

4.3 Connection

4.3.1 Opening FTP connection

To establish an FTP connection to a remote device, issue:

"**CALL {*bd_addr*} 1106 FTP**"

bd_addr	Bluetooth device address of the remote device.

Below is an example on setting up an FTP connection from iWRAP.

```
LINK 255: CALL 00:07:80:93:0c:aa 1106 FTP
LINK 255: CALL 0
LINK 255: CONNECT 0 FTP 2
LINK 255: OBEX 0 READY
```

After receiving the CONNECT event, you must wait for the OBEX {link_id} READY event before using the connection.

4.3.2 Closing FTP connection

The FTP connection should be closed with a **DISCONNECT** command:

"**DISCONNECT**"

Disconnect command closes the OBEX connection cleanly, and then disconnects the underlying RFCOMM link.

```
LINK 255: DISCONNECT
LINK 255: NO CARRIER 0 ERROR 0
```

4.3.3 FTP commands

There are several commands that you can use for moving in directory tree, modifying, deleting and creating files and directories. Basically FTP provides the PUT function as OPP client but on top of that you can use all other functions listed below.

Command	Function	Return
CD {*directory*}	Change to ***directory.***	"OBEX OK" or "OBEX NOT FOUND"
CDMK {*directory*}	Make ***directory*** and CD to it.	"OBEX OK" or "OBEX NOT FOUND"
CDUP	Go one directory level up.	"OBEX OK" or "OBEX NOT FOUND"
DEL {*object*}	Removes on ***object.*** (file or directory)	"OBEX OK" or "OBEX NOT FOUND"
DIR	Directory	Directory listing (in XML)
GET {*file_name*}	Download file with name ***file_name.***	File data in MUX frames
LS	Alias to DIR	Directory listing (in XML)
PUT {size} {mime} {filename}	Start OBEX transmission for file with name ***file_name.***	"OBEX OK" after {size} bytes is successfully transferred
RD {*object*}	Alias to **RMDIR**	"OBEX OK" or "OBEX NOT FOUND"
RM {*object*}	Alias to **DEL**	"OBEX OK" or "OBEX NOT FOUND"
RMDIR {*object*}	Removes on ***object.*** (file or directory)	"OBEX OK" or "OBEX NOT FOUND"

4.3.4 Sending a file using FTP

To upload a file to remote device you must use the PUT command. Please refer to 3.3.1 Sending a file - OPP Client.

Note:

If a PUT operation is ongoing (until **OBEX *{link_id}* OK** response is received) other FTP commands will not work. All file data must be completely sent to allow other FTP operations.

4.3.5 Receiving a file using FTP

To download a file using FTP you need to use the GET command. The file is received in OBEX format. The OBEX transfer is interpreted as shown in the table below. All field IDs that you don't understand you can ignore.

In OBEX, all multi-byte values are sent most significant byte first, e.g. 0x1234 is transmitted as 0x12, 0x34.

Field explanation	GET	Length	Filename field ID	Filename length	Filename	Filedata field ID	Filedata length	Filedata
Not last packet	0x90	***x***	0x01	***y***		0x48	***z***	
Last packet	0xa0	***x***	0x01	***y***		0x49	***z***	
Field size in bytes	1	2	1	2	***y***	1	2	***z***-3
Mandatory/Optional	M		O			O/M		
Packet length	***x*** bytes							

For more information please see IrDA Object Exchange Protocol specification [2].

5 OBEX error codes

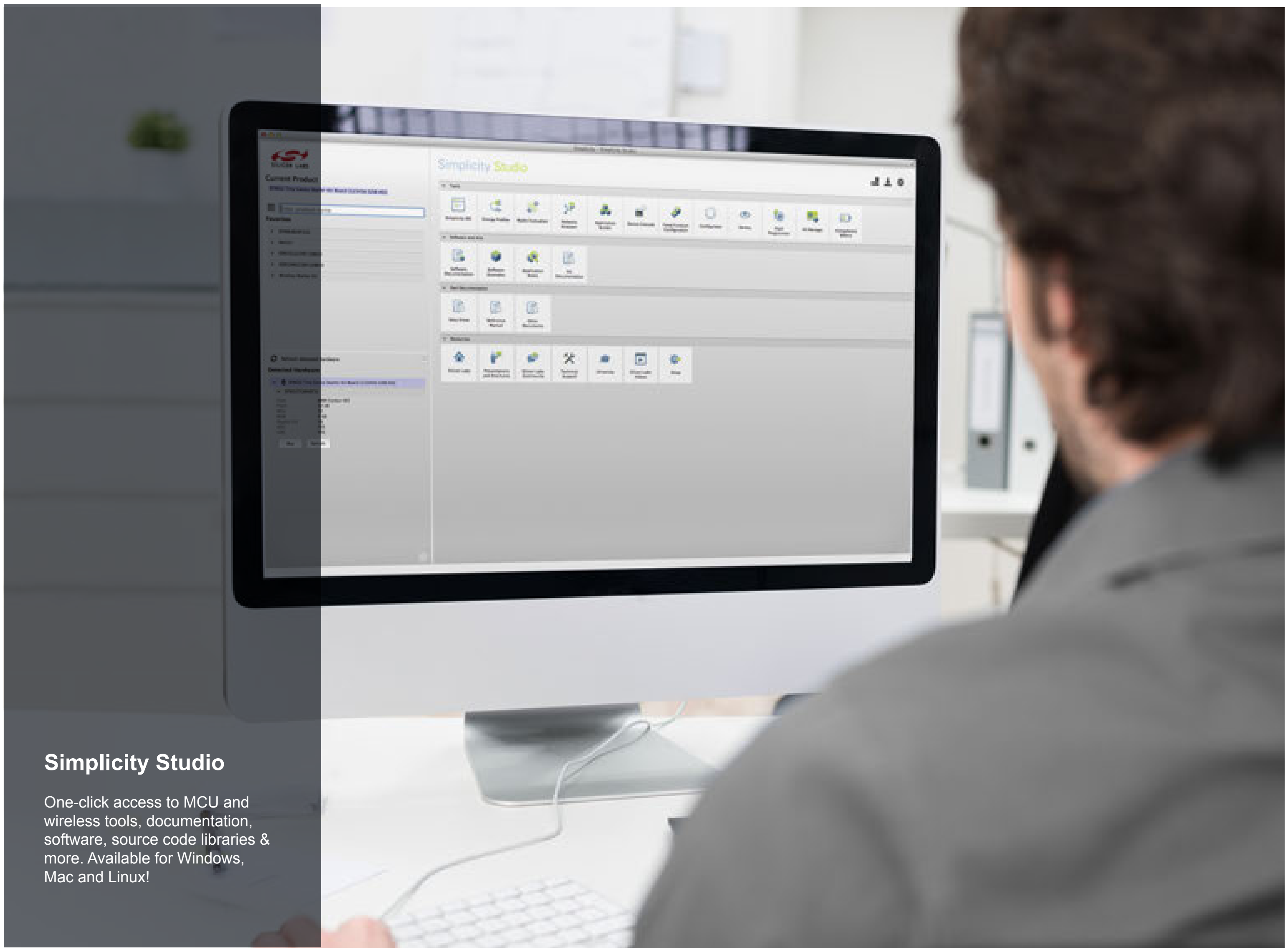
The following table lists the raw values, ASCII strings and their corresponding explanations for possible OBEX error codes.

Error code	iWRAP string	Explanation
0x40	BAD REQUEST	The request had bad syntax or was inherently impossible to be satisfied.
0x41	UNAUTHORIZED	The request cannot be completed without proper authorization.
0x43	FORBIDDEN	The request is for something that is forbidden by the server; authorization will not bypass this.
0x44	NOT FOUND	File or directory not found.
0x45	ERROR:45	The requested method is not allowed for this object.
0x49	ERROR:49	The request could not be completed due to a conflict with the current state of the resource.
0x50	ERROR:50	Internal server error; the remote device may be busy with other requests or for some other reason cannot process the request.
0x51	NOT IMPLEMENTED	The requested command is not implemented on the remote device.
0x53	SERVICE UNAVAILABLE	The server is currently unable to handle the request due to a temporary overloading or maintenance of the server.

References

[1] MIME Media Types http://www.iana.org/assignments/media-types/

[2] IrDA OBEX Specification http://www.irda.org/



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

Disclaimer
Silicon Laboratories intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Laboratories products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Laboratories reserves the right to make changes without further notice and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Laboratories shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any Life Support System without the specific written consent of Silicon Laboratories. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Laboratories products are not designed or authorized for military applications. Silicon Laboratories products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons.

Trademark Information
Silicon Laboratories Inc.® , Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, Clockbuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world’s most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, ISOmodem®, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress® and others are trademarks or registered trademarks of Silicon Laboratories Inc. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. All other products or brand names mentioned herein are trademarks of their respective holders.

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com