



Windows® Media Center

Remote Control and Receiver- Transceiver Specifications and Requirements for Windows Media Center in Windows Operating Systems

Applies to: Windows® Media Center is included in Windows® 7 Home Premium, Windows® 7 Professional, and Windows® 7 Ultimate. Windows Media Center is also included in Windows® 7 Enterprise, which is not an OEM-licensed product.

Abstract: This document is intended for independent hardware vendors (IHVs) and PC OEM partners who want to create remote control and receiver device combinations that decode input from the remote control for Windows Media Center. This document provides details, requirements, and options for designing and building remote controls and receivers for Windows Media Center Technologies on Windows® operating systems.

(c)2011 Microsoft Corporation. All rights reserved. This document is provided "as-is." Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

Microsoft, Windows 7, Win32, Windows, Windows Vista and Windows XP are either registered trademarks or trademarks of Microsoft Corporation in the United States or other countries or regions. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Contents

Introduction	1
What's New	1
Remote Control Specifications	2
Remote Control Requirements.....	2
Range Requirements	2
IR Protocol	2
Button Code Set Definition	2
Windows Logo Program	2
OEM Requirements	2
Remote Control Button Requirements.....	6
Button Functionality and Windows Mapping.....	11
Feedback LED	35
Backlight	36
Additional Navigation Methods	36
Additional Audio/Video Control Methods	36
Multifunction Remote Controls - Highly Recommended.....	36
Remote Control for OEM Bundled PC and Consumer Electronic Equipment	36
Windows Media Center Universal Remote Controls	37
TV standards and Windows Media Center Remote Controls	38
Windows Media Center Two-Way Remote Control with Compatible Auxiliary Display for Windows SideShow	39
Extender for Windows Media Center Remote Controls.....	39
Remote Address	39
Labels and Icons for the Remote Control	39
Windows Media Center Keyboards	39
Green Start Button Requirements.....	41
Key Logo License Agreement and the Green Start Button	41
Green Start Button Requirements	41
Green Start Button Subassembly	42
Device Housing and Orientation	44
Sample Implementation and Design Variations.....	51
Resources.....	56
Receiver/Transceiver Specifications	59
Overview of IR Receiver Options	59
Concepts.....	59
The Remote Control Functionality Needed	62
How Should You Build Your Device	63
More Complicated Receiver Examples.....	66
Connecting Your Receiver to the PC.....	68
Things to Remember When Building Your Device	70
IR Receiver/Transceiver Hardware Requirements	73
Components of an IR Transceiver	74
IR Transceiver Requirements	74

Emulation Requirements	76
Device Design Considerations.....	77
Microsoft Compatible Device Descriptor.....	82
Commands and Responses	85
Commands That Set Device State	86
Commands That Query Device State	92
Responses to Commands: Non-Error Cases	99
Responses to Commands: Error Cases	111
Illegal Command Handling.....	112
Bootloader Implementation.....	114
Bootloader Commands	121
Bootloader Responses	126
Format for Transmitting and Receiving IR.....	129
Suggested Firmware Memory Organization	132
Port Driver Requirements.....	133
Basic CIR Architecture.....	134
Introduction to the CIRClass Framework.....	135
CIR Version 1 DDI and Version 2 DDI.....	137
CIRClass and CIR Port Interface Details.....	139
Example CIR Port Driver – Hardware Design Requirements and Considerations.....	147
IOCTL Definitions.....	151
IR_ENTER_PRIORITY_RECEIVE_PARAMS	160
IR_DEV_CAPS	160
IR_DEV_CAPS_V1	161
IR_DEV_CAPS_V2 (Version 2 Only).....	161
IR_PRIORITY_RECEIVE_PARAMS	162
IR_RECEIVE_PARAMS	163
IR_SET_WAKE_PATTERN_PARAMS (Version 2 Only)	164
IR_TRANSMIT_PARAMS.....	164
IR_TRANSMIT_CHUNK	165
HID Device Requirements.....	165
HID Remote Control Receiver Requirements.....	166
HID Usage Codes	167
Reserved Button Codes.....	173

Introduction

This document provides the information needed to design a remote control and receiver that will work with those versions of Windows 7 that include Windows Media Center. This document describes the requirements for a remote control, the remote control functions that must be supported, the hardware specifications, and the infrared (IR) protocol requirements. This document also describes in depth the different options that are available for building a receiver that converts the commands from the remote control into actions for Windows Media Center.

What's New

General

- Combined the Remote Control Specifications and Receiver/Transceiver Specifications documents into one document for convenience.
- Changed the Power button label to Sleep throughout the document to accurately reflect the functionality of this remote control button.
- Removed the option for Green Button remote controls from Windows 7 Home Basic.

Remote Control Updates

- Added an option to support the Microsoft Quatro Pulse remote control IR protocol.
- Added support for the optional buttons for ISDB markets: 10, 11, 12, Audio Select, Logical Channel Input, and Network Select.
- Added the option to use the "moon" icon for the Sleep button (formally the Power button).
- Updated the label for the "*" button to include a "." ("*/.")
- Updated the icon for the More Information button to add a circle around the "I".
- Added clarification for the ordering of the color buttons for ISDB markets.
- Updated the "TV standards and Windows Media Center Remote Controls" section, which describes the types of buttons that can be distributed in different regions.
- Updated to have the option to remove the text labels above the number pad buttons.

Receiver/Transceiver Updates

- Added an "Overview of IR Receiver Options" section to help OEM partners understand the options for building Windows Media Center remote control receivers.
- Updated requirements to support the Microsoft Quatro Pulse remote control IR protocol, as follows:

Receivers must wake the system from *both* Microsoft RC-6 and Microsoft Quatro Pulse IR protocols.

- Updated the Port Driver specifications to include the version 2 Device Driver Interface (version 2 DDI), as follows:
 - The version 2 DDI adds robust reporting of hardware capabilities so that Windows Media Center UI can appropriately adjust based on what the hardware can support.
 - Better support for programmability of the Sleep button.
- Added support for discrete sleep and wake keys, which may require emulator firmware updates.
- Added the HID descriptor used for producing HID events for reference.

Remote Control Specifications

This section contains the specifications and requirements for the remote control and the Green Start button.

Remote Control Requirements

This section provides an overview of the required remote control functionality for Windows Media Center Technologies (referred to as a *Windows Media Center computer*).

The list of required remote control buttons and button functionality is included in this document.

Range Requirements

- IR remote controls: The range of transmission of the remote control shall be at least 5 meters at both the center of the receiver and up to 2 meters off center.
- Non-IR remote controls: The range of transmission of the remote control shall be at least 5 meters in all directions (regardless of the position of the receiver that is attached to the PC).

IR Protocol

Microsoft recommends that OEMs use either the Philips/Microsoft RC-6 or the Microsoft/SMK QP IR protocol.

Button Code Set Definition

The button code is the integer that the IR remote control sends to represent the button that was pressed on the remote control. This button code is then translated by the Windows Media Center IR drivers into commands used by the operating system.

Windows Logo Program

For a Windows Media Center remote control to be certified for the Windows logo, it must follow the Windows Logo Program. The Windows Logo Program requires that a series of tests be performed to verify that the remote control functions properly. The test tool will be included with the Windows Logo Kit. Existing Windows Media Center remote controls must meet the latest released Remote Control Specifications described in this document.

OEM Requirements

The Windows Media Center remote control is the primary device that is used to interact with and perform tasks in Windows Media Center. Therefore, a set of requirements is defined to ensure that the devices work together, work consistently, and create a predictable user interface. These requirements cover three main areas: experience branding, design overview, and button functionality.

Experience Branding

All Windows Media Center Remote Controls will use the same button treatment for the Windows Media Center Green Start button.

The Green Start button presents the Microsoft product branding and serves as an integral part of the overall user experience. Users should be able to relate easily to any device in the Windows Media Center ecosystem as part of the same Windows Media Center user experience. Microsoft has chosen to use the Green Button Assembly and its physical appearance as the branding mechanism. This branding should be visible but not overpowering and have a functional role in

controlling the user interface in Windows Media Center. The Green Button is used exclusively for starting Windows Media Center.

This brand extends beyond the remote into print and on-screen iconography. To extend the branding, all Windows Media Center remotes will have the same Green Start button. The Windows Key Logo license agreement specifies the requirements for licensing the logo.

The Windows Media Center Green Start button assembly size on a remote should be based on the type of remote that is required. For the standard rubber-based Windows Media Center remote control for a computer, the 11 mm Green Start button must be used. For a membrane-based remote control, the Green Start button artwork that is provided can be scaled to any size between 6.6 mm and 11 mm. However, the Green Start button image cannot be smaller than the largest button image for any of the other buttons that appear on the remote control. The 6.6 mm Green Start button assembly, which is discussed later in this document, can be used only when the height of the remote control housing is 12 mm or less. For more information about the Green Start button, including implementation details, see the Green Start Button Requirements section later in this document.

Design Overview

The following illustration is a conceptual industrial design of a remote control for Windows Media Center.



Figure 1: Illustration of a conceptual industrial design of a Windows Media Center remote control

This conceptual remote control represents the required set of functionality that must be provided by the Windows Media Center remote control. OEM partners can add additional functionality; however, this functionality must not interfere with or displace the required functionality.

The Windows Media Center remote can be separated into the functional areas described in the following list. The buttons in each functional area on the remote control should remain clustered together in the final remote control design because the buttons provide related functionality in terms of how the user interacts with and controls Windows Media Center.

Microsoft will provide localized text for the remote control. These text strings must be used to maintain consistency between the program's user interface text and the remote control.

- **Navigation controls.** The navigation controls form the main interaction point with the user interface. These controls enable the user to control Windows Media Center easily. This interface is based on a focus point that can be moved around the screen to perform a function or task in Windows Media Center. This “tab interface” lets the user to navigate through the user interface when they are sitting farther away from their display (compared to where they are sitting in a typical computer scenario).
- **Transport controls.** The transport controls are used to manage and play digital media content in Windows Media Center. This includes playing digital audio and video files and streams, playing and recording TV, playing CDs and DVDs, and playing slide shows. All buttons in the transport control cluster must be grouped together on the remote control, and additional buttons cannot be placed in the middle of the cluster of transport control buttons.
- **Audio and video controls.** The audio and video (A/V) controls enable the user to do tasks such as adjust the volume and change channels. The Standby button is included in this functional group even though the button is placed on a different part of the remote control (to prevent the user from accidentally pressing the button when using the remote control).
- **Numeric keypad.** The numeric keypad is used to enter numbers in Windows Media Center.
- **Interactive TV buttons.** The Interactive TV buttons are dedicated to starting and navigating TV experiences such as Teletext or Broadcast Markup Language (BML).

The following illustration is a conceptual industrial design of a remote control that includes Interactive TV controls for Windows Media Center. The conceptual design of the remote control includes a series of Windows Media Center shortcut buttons that let the user quickly go to an experience—such as to play or record TV, display the Guide, or play a DVD—by pressing a single button.

The following is a list of suggestions to follow, in addition to the button requirements listed later in this document, when designing a Windows Media Center remote control:

- The number 5 button on the numeric keypad should have a raised nub or dimple to help the user locate the center of the numeric keypad by touch.
- The Record button should be flush with the remote control case to reduce the risk of accidentally pressing the button.
- Place a chamfer on the Skip Forward and Skip Backward buttons to help physically differentiate these buttons from the Fast Forward and Rewind buttons on the remote control. The Skip Forward and Skip Backward buttons are commonly used when the user is playing recorded TV, where it is important for the user to differentiate the buttons by touch.
- The Play and Pause buttons should be grouped together in a recessed area or have a graphical border treatment. This helps to convey the message to the user that the buttons are related and often used together.

- In addition to providing visible icons or labels for the directional pad in the navigation controls, adding physical effects such as etching the Up, Down, Left, and Right Arrows into hard buttons will help the user to discover and use these buttons.

Remote Control Button Requirements

The following sections describe the required and optional buttons on the remote control. The following sections are separated according to the button category—Microsoft Required, Microsoft Recommended, Microsoft Optional, Microsoft Reserved, or Microsoft Retired buttons.

These button categories are used throughout this document when listing Windows Media Center remote control buttons and describing button functionality.

Microsoft Required Buttons

All Windows Media Center remote controls must include all Microsoft Required buttons.

Microsoft Required buttons are buttons that must be included on the remote control so that the user can fully interact with the Windows Media Center user interface, and can play and manage different media experiences in Windows Media Center.

The tables in this section identify the Microsoft Required buttons. Which remote control buttons are required will differ based on whether the remote control will be distributed with a Windows Media Center computer that has TV tuner hardware.

Navigation Buttons

The following table identifies the Microsoft Required navigation buttons for Windows Media Center computers.

Button name	Required for computers that include TV tuner hardware	Required for computers that do not include TV tuner hardware
Green Start button	Yes	Yes
Up	Yes	Yes
Down	Yes	Yes
Left	Yes	Yes
Right	Yes	Yes
OK	Yes	Yes
More	Yes	Yes
Back	Yes	Yes

Transport Controls

The following table identifies the Microsoft Required transport control buttons for Windows Media Center computers.

Button name	Required for computers that include TV tuner hardware	Required for computers that do not include TV tuner hardware
Play/Pause combo	Yes	Yes
Play	Yes	Yes
Pause	Yes	Yes
Stop	Yes	Yes
Rewind	Yes	Yes
Fast Forward	Yes	Yes
Skip Backward	Yes	Yes
Skip Forward	Yes	Yes
Record	Yes	No

Note Windows Media Center remote controls must have the ability to activate a Play and a Pause command. This can be implemented as either a separate button for each command or as a single button that toggles between the two commands. The current conceptual design for the Windows Media Center remote control uses separate buttons for Play and Pause.

Audio and Video Controls

The following table identifies the Microsoft Required audio and video control buttons for Windows Media Center computers.

Button name	Required for computers that include TV tuner hardware	Required for computers that do not include TV tuner hardware
Sleep Toggle (formerly Power Toggle)	Yes	Yes
Wake*	Yes	Yes
Sleep*	Yes	Yes

Button name	Required for computers that include TV tuner hardware	Required for computers that do not include TV tuner hardware
Volume Up	Yes	Yes
Volume Down	Yes	Yes
Mute	Yes	Yes
Channel/Page Up	Yes	No
Channel/Page Down	Yes	No

*** Note** All Windows Media Center remote controls must have either the single Sleep toggle button or both the Wake and Sleep buttons. No remote control should have all three buttons combined on one remote control. These three buttons are the only buttons on the remote control that can wake the system from the sleep mode. However, any shortcut or extensibility button that is designated by Microsoft can wake the system first before starting a program that is designed and developed for the 10-foot experience in Windows Media Center.

Windows Media Center Shortcut Keys

The following table identifies the Microsoft Required Windows Media Center shortcut key for Windows Media Center computers.

Button name	Required for computers that include TV tuner hardware	Required for computers that do not include TV tuner hardware
Guide	Yes	No

The following table identifies the Microsoft Required buttons for Windows Media Center computers. Buttons are required in locales that support Interactive TV functionality.

Button name	Required for computers that include TV tuner hardware	Required for computers that do not include TV tuner hardware
Red	Yes	No
Green	Yes	No
Blue	Yes	No
Yellow	Yes	No
Teletext On/Off	Yes	No

Microsoft Recommended Buttons

Windows Media Center remote controls can also include any or all of the Microsoft Recommended, Microsoft Optional, or Microsoft Reserved buttons.

Microsoft Recommended buttons are buttons that have consistently been shown to score high in usability studies about the perceived need by the end users.

Microsoft strongly recommends that the buttons in the following list be included on the remote control to highlight the user experience in Windows Media Center. However, these buttons are not required.

The following list identifies the Microsoft Recommended buttons for a Windows Media Center remote control:

- **DVD.** DVD Menu.
- **Numeric keypad.** Individual buttons for numbers 0 to 9, CLEAR, ENTER, #, and *. If a numeric keypad will appear on the remote control, then all these buttons need to be implemented on the remote control.
- **Audio and volume controls.** Channel Up and Channel Down.

If a remote control includes a numeric keypad, then the audio and volume controls must also be included on the remote control.

- **Windows Media Center shortcuts.** Live TV and Recorded TV.
- **Windows Media Center remote control buttons for ISDB-T markets.** 10, 11, 12, Sub Audio, and Input Channel. The 10,11,12 are highly recommended for this market as they represent the national TV channels.
- **Other buttons.** Zoom.

Microsoft Optional Buttons

Microsoft Optional buttons are buttons that are supported by Windows Media Center but are not required on a Windows Media Center remote. The remote control concept design does not include all of these buttons. However, the functionality for these buttons is available if a manufacturer wants to include these buttons.

The following list identifies the Microsoft Optional buttons for a Windows Media Center remote control:

- **Windows Media Center shortcuts.** Music, Pictures, Videos, Radio, and Extras.
- **DVD.** DVD Angle, DVD Audio, and DVD Subtitle.
- **Extensibility buttons.** Ext0 through Ext11 (12 buttons total).
- **Xbox 360.** For more information, send e-mail to xremotes@microsoft.com.
- **Other buttons.** Print, Network Selection, Video Selection, and Closed Captioning On/Off.

Microsoft Reserved Buttons

Microsoft Reserved buttons are defined now to allocate space in the button map and infrared (IR) stack. Buttons in this class do not currently have related functionality that is implemented in Windows Media Center. However, these button functions could be included in future releases. While we welcome developers to use these messages in their products, future uses of these buttons by Microsoft products might cause incompatibility with preexisting usages.

The following list identifies the Microsoft Reserved buttons for a Windows Media Center remote control:

- **Windows Media Center remote control miscellaneous buttons.** Channel Info.
- **DVD.** DVD Top Menu and Eject
- **Blu-ray buttons.** BD Tool.
- **Other buttons.** Display and Exit.

Microsoft Retired Buttons

Microsoft Retired buttons support functionality that will no longer be available in Windows Media Center. Retired buttons cannot be included on new Windows Media Center remote controls.

Due to design changes in the user interface of Windows Media Center, the following buttons cannot be included on new Windows Media Center remote controls:

- My TV
- Messenger
- Media Center Edition Power Menu
- Media Center Edition On

Button Functionality and Windows Mapping

This section provides a detailed description of button functionality, the corresponding Windows mapping, and related icons and labels when applicable. The buttons are grouped by functional areas.

Navigation Control Buttons - Required

This section describes the required navigation control buttons that are shown in the following figure.



Figure 2: Illustration of the required navigation controls for a Windows Media Center remote control

Green Start Button

Button space: Microsoft Required

Icon: 

Label: MEDIA CENTER

Button code: 13

First press action: Takes the user to the Windows Media Center Start menu. If Windows Media Center is not already started, Windows Media Center starts and the Windows Media Center Start menu appears.

The Green Start button is the only button on a Windows Media Center remote control that can start Windows Media Center and display the Windows Media Center Start menu. The Green Start button may not be used for any other purpose.

On a keyboard with Media Center functionality, Microsoft requires that the Green Start button is used to start Windows Media Center.

On a front panel or laptop with Media Center functionality, Microsoft strongly recommends that the Green Start button can be used to start Windows Media Center.

Second press action: Dismisses the Windows Media Center Start menu and takes the user to the last page that was previously viewed. If Windows Media Center is in windowed mode, pressing the Green Start button will put Windows Media Center into full-screen mode.

Auto-repeat: No

Remarks: The Green Start button assembly must be used on Windows Media Center remote controls. The provided artwork can be used if a manufacturer is creating an on-screen remote or membrane-based remote control. For more information about the Green Start button, including implementation details, see the Green Start Button Requirements section later in this document.

Important Microsoft encourages the OEM to add the text " Windows Media Center" under the Windows Media Center Green Start button to identify more clearly the function of the button for the consumers.

Up

Button space: Microsoft Required

Icon: 

Label: No label

Button code: 30

First press action: Moves the focus point up one position. If the focus point is at the top of the screen, pressing this button results in no action.

Second press action: Repeats first press action.

Auto-repeat: Yes

Down

Button space: Microsoft Required

Icon: 

Label: No label

Button code: 31


First press action: Moves the focus point down one position. If the focus point is at the bottom of the screen, pressing this button results in no action.

Second press action: Repeats first press action.

Auto-repeat: Yes

Left

Button space: Microsoft Required

Icon: 

Label: No label

Button code: 32

First press action: Moves the focus point to the left one position. If the focus point is at the left-most part of the screen, the focus point goes to the previous page in the user interface stack.

Second press Action: Repeats first press action.

Auto-repeat: Yes

Right

Button space: Microsoft Required

Icon: 

Label: No label

Button code: 33

First press action: Moves the focus point to the right one position. If the focus point is at the right-most part of the screen, pressing this button results in no action.

Second press action: Repeats first press action.

Auto-repeat: Yes

OK

Button space: Microsoft Required

Icon: OK

Label: OK

Button code: 34

First press action: Performs the action at the focus point.

Second press action: Repeats the action at the new focus point (if action can be taken at the new focus point).

Auto-repeat: No

Back

Button space: Microsoft Required

Icon: ←

Label: BACK

Button code: 35

First press action: Moves back one position in the user interface stack.

Second press action: Repeats first press action.

Auto-repeat: No

More Info Button

Button space: Microsoft Required

Icon: ⓘ

Label: MORE INFO or INFO

Button code: 15

First press action: Provides information for the focus point (if information is available).

Second press action: Hides displayed information.

Auto-repeat: No

Transport Control Buttons – Required

This section describes the required transport control buttons that are shown in the following figure.



Figure 3: Illustration of the transport controls for a Windows Media Center remote control

Play

Button space: Microsoft Required

Icon: ▶

Label: PLAY

Button code: 22

First press action: Starts playing media at the current position. If playback is paused, pressing this button begins playback from the current position.

Second press action: None

Auto-repeat: No

Pause

Button space: Microsoft Required

Icon: 

Label: PAUSE

Button code: 24

First press action: Pauses media playback at the current position.

Second press action: Toggles between pausing and continuing playback at the current position.

Auto-repeat: No

Play/Pause Combination

Button space: Microsoft Required

Icon: 

Label: PLAY/PAUSE

Button code: 110

First press action: Starts playing media at the current position. If paused, pressing this button begins playback from the current position.

Second press action: Toggles between pausing and continuing playback at the current position.

Auto-repeat: No

Remarks: Windows Media Center remote controls must have the ability to activate a Play and a Pause command. This functionality can be implemented as either a separate button for each command or as a single button that toggles between the two commands.

Stop

Button space: Microsoft Required

Icon: 

Label: STOP

Button code: 25

First press action: Stops playing media at the current position and moves the pointer to the starting point of the digital media file.

Second press action: None

Auto-repeat: No

Record

Button space: Microsoft Required (for Windows Media Center computers that include TV tuner hardware)

Icon: 

Label: REC

Button code: 23

First press action: Records the TV program as a file on a hard disk. If the pause buffer enables it, recording starts at the beginning of a show as defined by the Guide in Windows Media Center.

Second press action: None

Auto-repeat: No

Remarks: Microsoft recommends using a red record button on the remote control.

Fast Forward

Button space: Microsoft Required

Icon: 

Label: FWD

Button code: 20

First press action: Speeds up the time base of the digital media file to the first fast forward value (3X). If a slide show is currently playing, the next picture is displayed when this button is pressed.

Second press action: Cycles through the fast forward speed values looping through 0 (normal speed), 3X, 20X, and 60X. If a slide show is currently playing, the next picture is displayed when this button is pressed.

Auto-repeat: Yes

Rewind

Button space: Microsoft Required

Icon: 

Label: RWD

Button code: 21

First press action: Reverses direction of the digital media stream and speeds up the time base to the first rewind value (3X). If a slide show is currently playing, the previous picture is displayed when this button is pressed.

Second press action: Cycles through the rewind speed values looping through 0 (normal speed), 3X, 20X, and 60X. If a slide show is currently playing, the previous picture is displayed when this button is pressed.

Auto-repeat: Yes

Skip Forward

Button space: Microsoft Required

Icon: 

Label: SKIP FWD

Button code: 26

First press action: Skips forward one increment when this button is pressed. The meaning of an increment depends on the type of media that is playing:

- If a CD or playlist is playing, the next song is played.
- If a DVD is playing, the next chapter is played.
- If a slide show is playing, the next picture is displayed.
- If a recorded TV show is playing, playback skips ahead 29 seconds.

Second press action: Repeats first press action.

Auto-repeat: Yes

Skip Backward

Button space: Microsoft Required

Icon: 

Label: SKIP BACK

Button code: 27

First press action: Skips backward one increment when this button is pressed. The meaning of an increment depends on the type of media that is playing:

- If a CD or playlist is playing, the previous song is played.
- If a DVD is playing, the previous chapter is played.
- If a slide show is playing, the previous picture is displayed.
- If a recorded TV show is playing, playback skips backward 7 seconds.

Second press action: Repeats first press action.

Auto-repeat: Yes

Audio and Video Control Buttons – Required, Recommended, and Optional

This section describes the audio and video buttons. They are all required.

The following figure shows the audio and video controls for a Windows Media Center remote control.



Figure 4: Illustration of the audio and video controls for a Windows Media Center remote control

Sleep Toggle (formally the Power Toggle button)

Button space: Microsoft Required

Icon:  or 

Button code: 12

First press action: This button performs the action the user has configured for the Windows Sleep button. Typically, that action puts the Windows Media Center computer into standby or sleep mode if the computer is running. If the computer is in sleep mode, the computer wakes from sleep mode.

Second press action: Toggles the sleep state of the Windows Media Center computer.

Auto-repeat: No

Remarks: This sleep icon is optional. The OEM can use either the previous power icon or the sleep icon (recommended).

Wake

Button space: Microsoft Required (if choosing discrete Sleep On/Off buttons in design)

Icon: Determined by OEM.

Label: Wake

Button code: 41

First press action: Wakes the computer if the computer is in sleep mode.

Second press action: Takes no action if the computer is in awake mode.

Auto-repeat: No

Remarks: This sleep icon is optional. The OEM can use either the previous power icon or the sleep icon (recommended).

Sleep

Button space: Microsoft Required (if choosing discrete Sleep On/Off buttons in design). See Remarks below.

Icon: Determined by OEM.

Label: Sleep

Button code: 42

First press action: If the Windows Media Center computer is turned on, pressing this button performs the action that is set for the Sleep button.

Second press action: If computer is asleep, pressing this button results in no action.

Auto-repeat: No


Remarks: This sleep icon is optional. The OEM can use either the previous power icon or the sleep icon (recommended).


The icon can be either the Power icon or the Sleep icon.

All Windows Media Center remote controls must have either a Sleep (Standby) button or both the Sleep and Wake buttons. No remote should have all three buttons combined on the remote control.

Volume Up

Button space: Microsoft Required

Icon: 

To indicate that the volume is increasing, the following icon is recommended: 

Label: VOL +

Button code: 16


First press action: Increases the current volume by one unit. If the audio is currently muted when this button is pressed, muting is disabled and the volume increases by one unit.


Second press action: Repeats first press action.

Auto-repeat: Yes

Volume Down

Button space: Microsoft Required

Icon: 

To indicate that the volume is decreasing, the following icon is recommended: 

Label: VOL -

Button code: 17


First press action: Decreases the current volume by one unit. If the audio is currently muted when this button is pressed, muting is disabled and the volume decreases by one unit.


Second press action: Repeats first press action.

Auto-repeat: Yes

Channel Up

Button space: Microsoft Required for systems that include TV tuner hardware and for remote controls that have a numeric keypad. If the system does not include TV tuner hardware and the remote control does not contain a numeric keypad, this button is Microsoft Recommended.

Icon: 

To indicate that the channel number is increasing, the following icon is recommended: 

Label: CH +

Button code: 18


First press action: Increases the current channel number by one. If Windows Media Center is in a list view when this button is pressed, the focus point moves forward one page in the list.


Second press action: Repeats first press action.

Auto-repeat: No

Channel Down

Button space: Microsoft Required for systems that include TV tuner hardware and for remote controls that have a numeric keypad. If the system does not include TV tuner hardware and the remote control does not contain a numeric keypad, this button is Microsoft Recommended.

Icon: 

To indicate that the channel is decreasing, the following icon is recommended: 

Label: CH -

Button code: 19


First press action: Decreases the current channel number by one. If Windows Media Center is in a list view when this button is pressed, the focus point moves backward one page in the list.

Second press action: Repeats first press action.

Auto-repeat: No

Mute

Button space: Microsoft Required

Icon: 

Label: MUTE

Button code: 14

First press action: Mutes the computer audio.

Second press action: Toggles the mute state.

Auto-repeat: No

Closed Captioning On/Off

Button space: Microsoft Optional

Icon: Determined by OEM.

Label: CC

Button code: 43

First press action: Shows or hides closed captioning. If closed captioning is not currently displayed, pressing this button will turn on the closed captioning display. If closed captioning is currently displayed, then pressing this button will turn off closed captioning.

Second press action: Toggles between displaying closed captioning and turning off closed captioning.

Auto-repeat: No

Interactive TV (Teletext and ISDB-T) Buttons – Required and Reserved

This section describes the Interactive TV buttons that are supported by Windows Media Center. If Interactive TV data is supported in the TV signal for the locale that the remote will be distributed to, Interactive TV buttons must appear on the remote control. These buttons are also used for Integrated Services Digital Broadcasting (ISDB) Interactive TV functions.

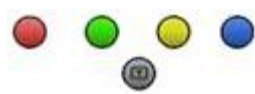
Outside of these two primary video streaming standards, Microsoft reserves these buttons. No specifications are available for intended Microsoft usage of these buttons. Future use of these buttons by Microsoft might be incompatible with any implementations generated before the intended usage is specified.

Additionally, these buttons cannot be overloaded with functionality that works in Windows Media Center or other programs. These buttons are for Interactive TV functions.

In the Japanese market, the color order for the interactive buttons are (left to right) Blue, Red, Green, and Yellow.



In other markets the color order for the interactive buttons are (left to right) Red, Green, Yellow and Blue.



Note For countries or regions that do not support Interactive TV functions, the colored buttons can be used as OEM extensibility buttons.

Interactive TV On/Off: Teletext On/Off or ISDB-T Data

Button space: Microsoft Required (if implemented), use OEM extensibility in countries or regions not implementing Interactive TV.

Icon:  or 

Icon: d (for Japan)

Label: TELETEXT

Button code: 90

First press action: Turns Teletext on and off. If Teletext is off, Teletext is turned on when this button is pressed. If Teletext is on, Teletext is turned off when this button is pressed.

Second press action: Takes no action.

Auto-repeat: No

Red

Button space: Microsoft Required

Icon: Solid Color Red Button

Label: No required label

Button code: 91

First press action: Goes to the red shortcut link in Interactive TV mode.

Second press action: Takes no action.

Auto-repeat: No

Green

Button space: Microsoft Required

Icon: Solid Color Green Button

Label: No required label

Button code: 92

First press action: Goes to the green shortcut link in Interactive TV mode.

Second press action: Takes no action.

Auto-repeat: No

Yellow

Button space: Microsoft Required

Icon: Solid Color Yellow Button

Label: No required label

Button code: 93

First press action: Goes to the yellow shortcut link in Interactive TV mode.

Second press action: Takes no action.

Auto-repeat: No

Blue

Button space: Microsoft Required

Icon: Solid Color Blue Button

Label: No required label

Button code: 94

First press action: Goes to the blue shortcut link in Interactive TV mode.

Second press action: Takes no action.

Auto-repeat: No

Exit

Button space: Microsoft Reserved

Icon: To be determined.

Label: To be determined.

Button code: 108

First press action: To be determined.

Second press action: Repeats message.

Auto-repeat: No

Numeric Keypad - Recommended

This section describes the recommended numeric keypad buttons that are shown in the following figure.




Figure 5: Illustration of a numeric keypad for a Windows Media Center remote control

Note Although it is not required, Microsoft strongly recommends that the letters appear above the numbers in the number pads. Usability studies have seen significant issues for the consumer when the letters are below the button.

Button space: Microsoft Recommended

The following table shows the individual buttons of a numeric keypad and the corresponding icons and button codes.

Button	Icon	Button code
0	0 - 	0
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8
9	9	9
Clear	CLEAR	10
Enter	ENTER	11
	#	28

Button	Icon	Button code
*	. (The "." should be placed on the label, not on the button itself.) Or */.	29

Remarks: If a numeric keypad is included on the Windows Media Center remote control, the remote control must include all the buttons shown for the numeric keypad in the preceding table and also the Channel Up and Channel Down controls.

Microsoft will provide localized text for the remote control. These text strings must be used to maintain consistency between the program's user interface text and the remote control.


Shortcut Buttons – Required, Recommended, or Optional

This section describes the shortcut buttons for Windows Media Center. These buttons provide a quick way for users to access key media experiences in Windows Media Center.

The following shortcut button descriptions list the required button first, the recommended buttons next, and the optional buttons last. See the "Button space:" entry at the top of each listing.

Guide

Button space: Microsoft Required

Icon: 

Label: GUIDE

Button code: 38


First press action: Displays the Guide in Windows Media Center.

Second press action: Cycles through different Guide options.

Auto-repeat: No

Live TV

Button space: Microsoft Recommended

Icon: 

Label: LIVE TV

Button code: 37

First press action: Shows live TV on the currently-selected channel.

Second press action: Takes no action.

Auto-repeat: No

Recorded TV

Button space: Microsoft Recommended

Icon: 

Label: REC TV

Button code: 72

First press action: Displays the Recorded TV page in Windows Media Center.

Second press action: Takes no action.

Auto-repeat: No

Music

Button space: Microsoft Optional

Icon: 

Label: MUSIC

Button code: 71


First press action: Displays the Music Library in Windows Media Center.

Second press action: Takes no action.

Auto-repeat: No

Pictures

Button space: Microsoft Optional

Icon: 

Label: PICTURES

Button code: 73


First press action: Displays the Picture Library in Windows Media Center.

Second press action: Takes no action.

Auto-repeat: No

Videos

Button space: Microsoft Optional

Icon: 

Label: VIDEOS

Button code: 74

First press action: Displays the Video Library in Windows Media Center.

Second press action: Takes no action.

Auto-repeat: No

Radio

Button space: Microsoft Optional

Icon: 

Label: RADIO

Button code: 80

First press action: Displays the main Radio page in Windows Media Center.

Second press action: Takes no action.

Auto-repeat: No

Extras (formally Online Media)

Button space: Microsoft Optional

Icon: To be determined.

Label: Extras

Button code: 60

First press action: Displays Extras Library in Windows Media Center.

Second press action: Takes no action.

Auto-repeat: No

Extensibility Buttons - Optional

This section describes the extensibility control buttons, which are optional. The standard Windows Media Center IR protocol provides for 12 extensibility buttons. These buttons map to EXT0 through EXT11 in the Windows Media Center human interface design (HID) collection. The IR stack is programmed to pass these buttons through as HID usage as defined later. For example, an extensible button could be used to teach the user how to program a button on the remote control in Windows Media Center, start a program, or control home automation.

EXT

Button Space: Microsoft Optional

Icon: Determined by OEM.

Label: Determined by OEM.

Button code: The following table provides the extensible button names and the corresponding Windows mapping.

Button name	Button code mapping
EXT0	50
EXT1	51
EXT2	52
EXT3	53
EXT4	54
EXT5	55
EXT6	56
EXT7	57

Button name	Button code mapping
EXT8	58
EXT9	128
EXT10	129
EXT11	111

First press action: Generates EXTn HID message in the Media Center Vendor Specific Collection (page 0xFFBC, usage 0x88).

Second press action: Repeats message.

Auto-repeat: No

DVD Buttons - Recommended, Optional, or Reserved

This section describes DVD buttons that provide additional control for playing a DVD in Windows Media Center. The recommended button is listed first, followed by the optional and reserved buttons.

DVD Menu

Button space: Microsoft Recommended

Icon: 

Label: DVD MENU

Button code: 36

First press action: Displays the DVD menu.

Second press action: Takes no action.

Auto-repeat: No

DVD Angle

Button space: Microsoft Optional

Icon: 

Label: DVD ANGLE

Button code: 75

First press action: Changes the camera angle when watching a DVD. This button appears on some but not all remote controls.

Second press action: Continues to cycle through the available DVD camera angles for the current DVD.

Auto-repeat: No

DVD Audio

Button space: Microsoft Optional

Icon: 

Label: DVD AUDIO

Button code: 76

First press action: Plays the next available audio track on a DVD.

Second press action: Repeats first press action.

Auto-repeat: No

DVD Subtitle

Button space: Microsoft Optional

Icon: 

Label: SUBTITLE

Button code: 77

First press action: Displays the DVD subtitles when watching a DVD.

Second press action: Repeats first press action.

Auto-repeat: No

DVD Top Menu

Button space: Microsoft Reserved

Icon: To be determined.

Label: DVD Top Menu

Button code: 67

First press action: To be determined.

Second press action: To be determined.

Auto-repeat: No

Eject

Button space: Microsoft Reserved

Icon: 

Label: EJECT

Button code: 40

First press action: Ejects a DVD drive.

Second press action: Repeats first press action.

Auto-repeat: No

Miscellaneous Buttons - Recommended or Optional

This section describes miscellaneous buttons that control additional functionality in Windows Media Center. The recommended button is listed first, followed by the optional button.

Zoom

Button space: Microsoft Recommended

Icon: 

Label: ZOOM

Button code: 39

First press action: Toggles between various aspect modes in TV.

Second press action: Repeats first press action.

Auto-repeat: No

Print

Button space: Microsoft Optional

Icon: 

Label: PRINT

Button code: 78

First press action: Prints an item in Windows Media Center by using a program.

Second press action: Repeats first press action.

Auto-repeat: No

Miscellaneous Buttons - Reserved

This section describes reserved buttons that are defined now to allocate space in the button map and IR stack. At this time, buttons in this class do not have functionality implemented in Windows Media Center. However, the system message is produced.

These functions might be in future releases. While we welcome developers to use these messages in their products, future uses of these buttons by Microsoft products might cause incompatibility with preexisting uses. No specifications are available for intended Microsoft use of these buttons. Future use of these buttons by Microsoft might be incompatible with any implementations generated before the intended use is specified.

Display

Button space: Microsoft Reserved

Icon: Determined by OEM.

Label: DISPLAY

Button code: 79

First press action: Generates OEM2 HID message in the Media Center Vendor Specific Collection. This button is intended to control the front panel display of home entertainment computers. When this button is pressed, the display could be turned on or off, or the display mode could change.

Second press action: Repeats message.

Auto-repeat: No

Kiosk

Button space: Microsoft Reserved

Icon: To be determined.

Label: KIOSK

Button code: 106

First press action: To be determined.

Second press action: Repeats message.

Auto-repeat: No

Additional Buttons for Remote Controls for ISDB-T Market – Recommended, Optional and Reserved

This section describes buttons that are unique to the ISDB-T market in Japan.

The reserved buttons are now defined to allocate space in the button map and IR stack. At this time, buttons in this class do not have functionality implemented in Windows Media Center. However, the system message is produced. These functions might be in future releases. While developers can use these messages in their products, future uses of these buttons by Microsoft products might cause incompatibility with preexisting uses. No specifications are available for intended Microsoft use of this button. Future use of this button by Microsoft might be incompatible with any implementations generated before the intended use is specified.

10

Button space: Microsoft Recommended

Icon: “10”

Label: Determined by OEM (locale specific).

Button code: 62

First press action: Changes channel to channel 10.

Second press action: Repeats first press action.

Auto-repeat: No

11

Button space: Microsoft Recommended

Icon: “11”

Label: Determined by OEM (locale specific).

Button code: 63

First press action: Changes channel to channel 11.

Second press action: Repeats first press action.

Auto-repeat: No

12

Button space: Microsoft Recommended

Icon: “12”

Label: Determined by OEM (locale specific).

Button code: 64

First press action: Changes channel to channel 12.

Second press action: Repeats first press action.

Auto-repeat: No

Channel Input

Button space: Microsoft Recommended

Icon: Determined by OEM.

Label: Determined by OEM (locale specific).

Button code: 66

First press action: Brings up UI that allows users to enter three-digit channels.

Second press action: Repeats first press action.

Auto-repeat: No

Sub Audio

Button space: Microsoft Recommended

Icon: Determined by OEM

Label: Determined by OEM

Button code: 45

First press action: Brings up a menu that allows users to select different audio.

Second press action: Repeats first press action.

Auto-repeat: No

Remarks: Selects the audio language.

Network Selection Button

Button space: Microsoft Optional

Icon: Network

Label: NETWORK

Button code: 44

First press action: Cycles between broadcast types.

Second press action: Repeats message.

Auto-repeat: No

Remarks: Selects between satellite/cable and terrestrial TV.

Video Selection Button

Button space: Microsoft Optional

Icon: Video

Label: Video

Button code: 97

First press action: Once the command code is pressed, Windows Media Center checks if the current transport stream (channel) contains more than one video stream. If a channel has only one video stream, the button does nothing because the user cannot select anything.

If the channel contains more than one video stream, the "Now Playing" TV context menu is displayed, focused on the Video pane. This Video pane displays all of the sub-video streams and allows users to navigate to or select a different video. Depending on the information in the video stream, they may be labeled simply as "Video1" and "Video2", or labeled more specifically, such as "High Resolution" and "Low Resolution" as is sometimes used in satellite broadcasting.

Second press action: Repeats action.

Auto-repeat: No

Channel Info

Button space: Microsoft Reserved

Icon: To be determined.

Label: To be determined.

Button code: 65

First press action: To be determined.

Second press action: Repeats message.

Auto-repeat: No

Blu-ray Button - Reserved

This section describes a button that was added to support OEMs developing their own Blu-ray program for Windows Media Center. This functionality is not implemented in Windows Media Center. However, the system message is produced. No specifications are available for intended Microsoft use of this button. Future use of this button by Microsoft might be incompatible with any implementations generated before the intended use is specified.

BD Tool

Button space: Microsoft Reserved

Icon: To be determined.

Label: To be implemented per Blu-ray requirements.

Button Code: 120

First press action: To be determined.

Second press action: To be determined.

Auto-repeat: No

Feedback LED

It is highly recommended that the remote control provide the user with visual feedback that a button has been pressed. This feedback can be displayed using an LED (of any color) that is connected to the output circuitry for the remote control.

Backlight

Microsoft recommends including backlight functionality in the remote control because the user might be using the Windows Media Center remote control in a dark room. At a minimum, the transport and navigation controls should be illuminated by pressing a backlight button. The backlight button should be on the side or face of the remote control. If the button is placed on the face of the remote control, Microsoft recommends that the backlight button glow in the dark to help the user locate the button easily. Also, pressing any button on the remote can enable backlighting.

Additional Navigation Methods

The Windows Media Center interface is a tabbed interface and usability tests show that it works best with discrete arrows and an OK button.

An OEM must support the discrete directional arrow interface in an approved manner (to include discrete physical buttons, round disk configuration with artwork-indicated discrete direction, or capacitance-driven discrete arrows).

The OEM has the option of adding any additional methods of navigation to the remote control .

Additional Audio/Video Control Methods

Other audio/video control methods can be implemented on a Windows Media Center remote control, such as a jog wheel to control volume. If an alternative method is used, the control must have the ability to transmit one and only one button press at a time in order to pass certification.

Multifunction Remote Controls - Highly Recommended

In addition to controlling Windows Media Center computers, multifunction remote controls can control TV power and volume. No other devices or TV button functionality can be supported by this remote control.

Multifunction remote controls:

- Must have a separate Power button for the TV power.
- Must be designed so that after the TV button is configured, the volume buttons must be configured for TV volume.
- Must support IR learning.
- May contain a universal database and may contain more than one brand.
- Must be designed so that mute functionality cannot be remapped to the TV. (This ensures that closed captioning is correctly displayed in Windows Media Center.)
- Should be set up through a 10-foot extensibility application.

Remote Control for OEM Bundled PC and Consumer Electronic Equipment

When an OEM bundles a computer with other consumer electronic equipment, the OEM can create a Windows Media Center remote control that controls these multiple devices from the same manufacturer. The remote control is limited to use with only that brand.

These types of remotes:

Remote Control and Receiver-Transceiver Specifications and Requirements
for Windows Media Center in Windows Operating Systems

- Are required to have all of the required Windows Media Center buttons to qualify as a Green Button logo device. The remote must be configured out-of-the-box as a Windows Media Center remote. Other functions for consumer electronic devices can be available by switching modes.
- May contain a universal database that may contain more than one brand.

Windows Media Center Universal Remote Controls

Windows Media Center remote controls that manage more than the TV power and volume are considered *universal* remote controls.

To qualify as a Windows Media Center universal remote control, the following are required:

- The Windows Media Center Green Start button must be persistent in all modes of the universal remote control. Regardless of the mode that is currently selected, pressing the Green Start button must always take the user to the Start menu in Windows Media Center.
- The Windows Media Center Green Start button cannot be overloaded and must always take the user to the Start menu in Windows Media Center.
- To be certified under the Windows Logo Program, the remote control must have all of the required Windows Media Center buttons either on the remote control as hardware buttons or on the screen as software buttons. The remote must be configured out-of-the-box as a Windows Media Center remote, and other functions for other devices can be available in other modes.
- When in Windows Media Center mode, all required buttons must be directly available. All labels and/or icons must reflect the function that the button provides in Windows Media Center.
- The remote control must be programmable or support learning with an IR database.
- The remote control must have support for a wide range of manufacturers. It cannot support just one brand or device manufacturer.
- The universal remote control mode must be readily apparent and visible.
- If switches or sliders are used, the end user must be able to determine easily the current remote control mode.
- Transport controls cannot be overloaded with other functions.
- Windows Media Center shortcut buttons must maintain Windows Media Center labels in primary positions. Icons cannot be altered or added on the physical shortcut buttons to reflect additional functionality.

The following are recommended and Microsoft strongly encourages all manufacturers to support these recommendations:

- Microsoft strongly recommends that the universal remote control setup be done through a 10-foot extensibility program.
- Slide or manual switches for the universal modes are strongly discouraged. If switches or sliders are used, the end user must be able to determine easily which mode the remote control is in.
- Navigation controls and the OK button must function as navigation controls in all modes.
- Additional buttons added to the remote for universal functionality should be grouped logically with other buttons that provide similar functionality.

TV standards and Windows Media Center Remote Controls

For regions where Windows Media Center provides native support for TV broadcast standards, the remote controls must meet the general remote control requirements in this document.

In those regions where TV has a secondary 10-foot application, the remotes must follow the guidelines defined below.

If you are distributing a secondary 10-foot TV application on a Windows Media Center computer, you must distribute a remote control that meets the requirements in this document. In addition, manufacturers have an option to create a Windows Media Center remote control that supports a secondary 10-foot TV application. This remote control must have the Windows Media Center Green button and can have a secondary 10-foot TV application launch button. The buttons for the secondary 10-foot application can use either of the recommended IR protocols or the protocol currently used in the secondary 10-foot TV application.

Windows Media Center Two-Way Remote Control with Compatible Auxiliary Display for Windows SideShow

An OEM can build a two-way remote control with an auxiliary display for controlling music playback, guiding navigation and other features. A Windows SideShow gadget for Windows Media Center will support this functionality. For more information, contact ssremote@microsoft.com.

Extender for Windows Media Center Remote Controls

Media Center Extender technology can provide full access to personal content stored on Digital Living Network Alliance (DLNA)-compatible digital media servers. For more information about remote controls for these products, contact: mcxprtnr@microsoft.com.

Remote Address

The IR Protocols support multiple remotes and multiple Windows Media Center computers in the same room. The total number of separate remote control addresses is eight. The user should be able to change the address of the remote with a few simple keystrokes.

On the remote control, the end user simultaneously presses the DVD MENU button plus a numeric button [1-8] for more than three seconds to change the remote control address. To indicate that the remote control address was changed successfully, the LED will blink twice. It is strongly recommended that all Windows Media Center remote controls support addressability.

Alternatively, the remote control address can also be changed by pressing the DVD MENU or MORE INFO button and a transport button for longer than three seconds. The transport buttons are Stop, Record, Pause, Rewind, Play, Fast Forward, Skip Back, and Skip Forward, and these buttons map to 1 through 8 respectively.

Labels and Icons for the Remote Control

The remote control can be icon-based only, label-based only, or have both icons and labels.

The following implementations are required:

- If using icons, the icons can be placed either above, below, or on the button.
- If using labels, the label placement must be consistent within button clusters.

In addition to the preceding required elements, Microsoft strongly recommends that:

- If you are designing a new remote control, make sure that the buttons are large enough so that the icons can be located on the buttons.

Windows Media Center Keyboards

The purpose of a Windows Media Center keyboard is to enhance the user experience by giving the user all the buttons and functions they need in one easy-to-use device.

The keyboard does not replace the requirements to distribute a Windows Media Center remote control with a Windows Media Center computer. When you are designing a Windows Media Center keyboard, Microsoft strongly recommends placing the characters above the number keys. The Green Start button on a Windows Media Center keyboard must meet the same requirements that are specified for a standard Windows Media Center remote control.

Mouse functionality can be integrated into the keyboard.

The following Microsoft Required buttons must appear on a Windows Media Center keyboard:

- Green Start button

- Volume Up
- Volume Down
- Mute
- Channel Up
- Channel Down
- Play
- Pause or Play/Pause combo
- Stop
- Rewind
- Fast Forward
- Skip Forward
- Skip Backward
- More

The following Microsoft Optional buttons can appear on a Windows Media Center keyboard:

- Back
- Enter
- Left
- Right
- Up
- Down
- OK
- Guide
- Live TV
- Recorded TV
- Radio
- Numeric keypad
- Clear
- Enter
- *
- #

Green Start Button Requirements

The Green Start button is made up of a keycap and dome lens (referred to as the *Green Start button subassembly*) and is implemented within a device housing that has a sloped edge (called a *chamfer*).

The Green Start button is designed to provide an attractive and discoverable actuator to display the Windows Media Center Start menu.



Figure 6: Illustration of top view and perspective view of the Green Start button for Windows Media Center Technologies

These guidelines provide the details necessary to obtain a high-quality Green Start button subassembly and to implement it on a Windows Media Center remote control or other device to create a lasting, positive user experience for customers. Device manufacturers must purchase the Green Start button subassembly from a certified supplier.

Key Logo License Agreement and the Green Start Button

When adding and implementing the Green Start button on a device, the device manufacturer must adhere to the requirements specified in the *Key Logo License Agreement* for that device. For more information or to get a copy of the *Key Logo License Agreement* for a device, contact the primary Microsoft contact for the device specification.

Green Start Button Requirements

This section includes the Green Start button requirements. See the topic “Sample Implementation and Design Variations” later in this document for an example implementation and possible design variations.

The Green Button function is governed by the Key Logo License Agreement and the Remote Control Specifications section in this document.

Only one Green Start button can be implemented on a single device. If a second Windows key is included, it must not be a Green Start button.

Green Start Button Must Meet Size Requirements of Device

The dome lens for the Green Start button can be manufactured in one of three authorized sizes: 11 millimeters (mm), 9 mm, and 6.6 mm.

The authorized sizes for the Green Start button (11 mm, 9 mm, and 6.6 mm) refer to the diameter of the dome lens and do not reflect the total size of a manufactured Green Start button subassembly.

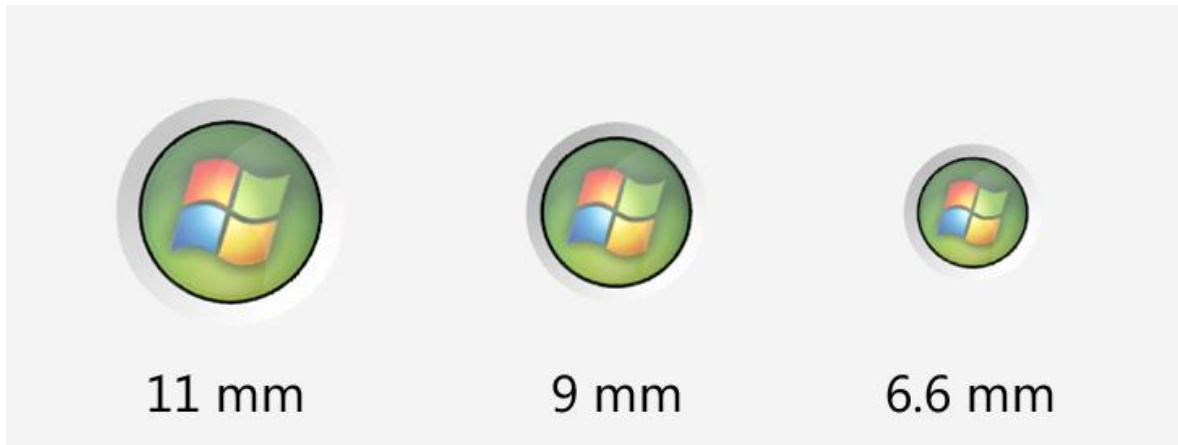


Figure 7: Illustration of the three different sizes in which the Green Start button can be implemented (illustration is not to scale)

The size of the Green Start button that should be implemented on a device is determined by the device type. To determine the button sizes that can be implemented on a particular device, refer to the hardware specification for that device.

For example, if you are preparing to manufacture a remote control for Windows Media Center, refer to the Remote Control Specification for Windows Media Center Technologies to find out which size Green Start button should be included on the remote control.

Green Start Button Subassembly

The Green Start button subassembly is made up of two parts: the keycap and the dome lens. As shown in Figure 8, the dome lens is made up of three layers. The dome lens adheres to the keycap so that the Green Start button can withstand high levels of force that may occur during regular use.

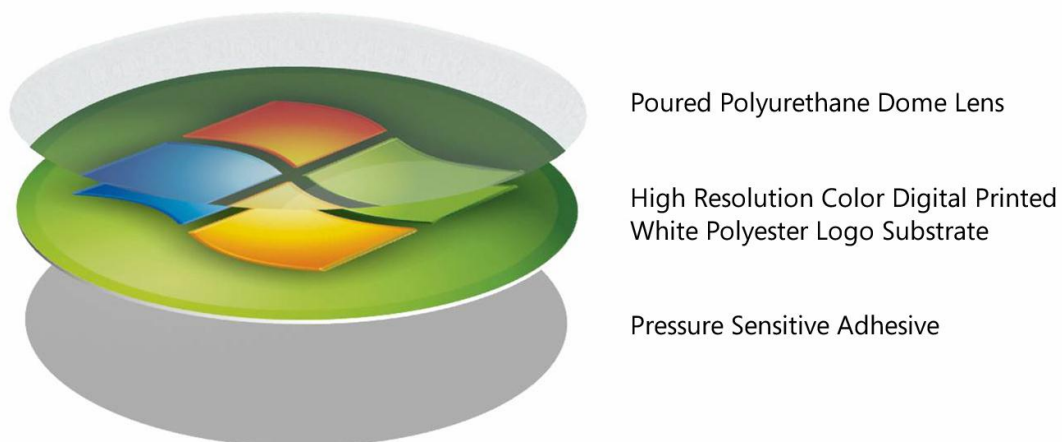


Figure 8: Illustration that shows the three layers of the dome lens

The keycap is manufactured in a clear material to better integrate with any device design and color scheme. The keycap design includes an orientation feature to ensure that the Green Start

button is correctly oriented to the device axis when assembled. For more information, see the topic "Green Start Button Must Be Correctly Oriented on the Device", later in this document.

The keycap has a ridge that surrounds it and helps to secure the dome lens. The ridge ensures that the dome lens is centered on the keycap and also prevents the dome lens from being dislodged easily by multidirectional force.

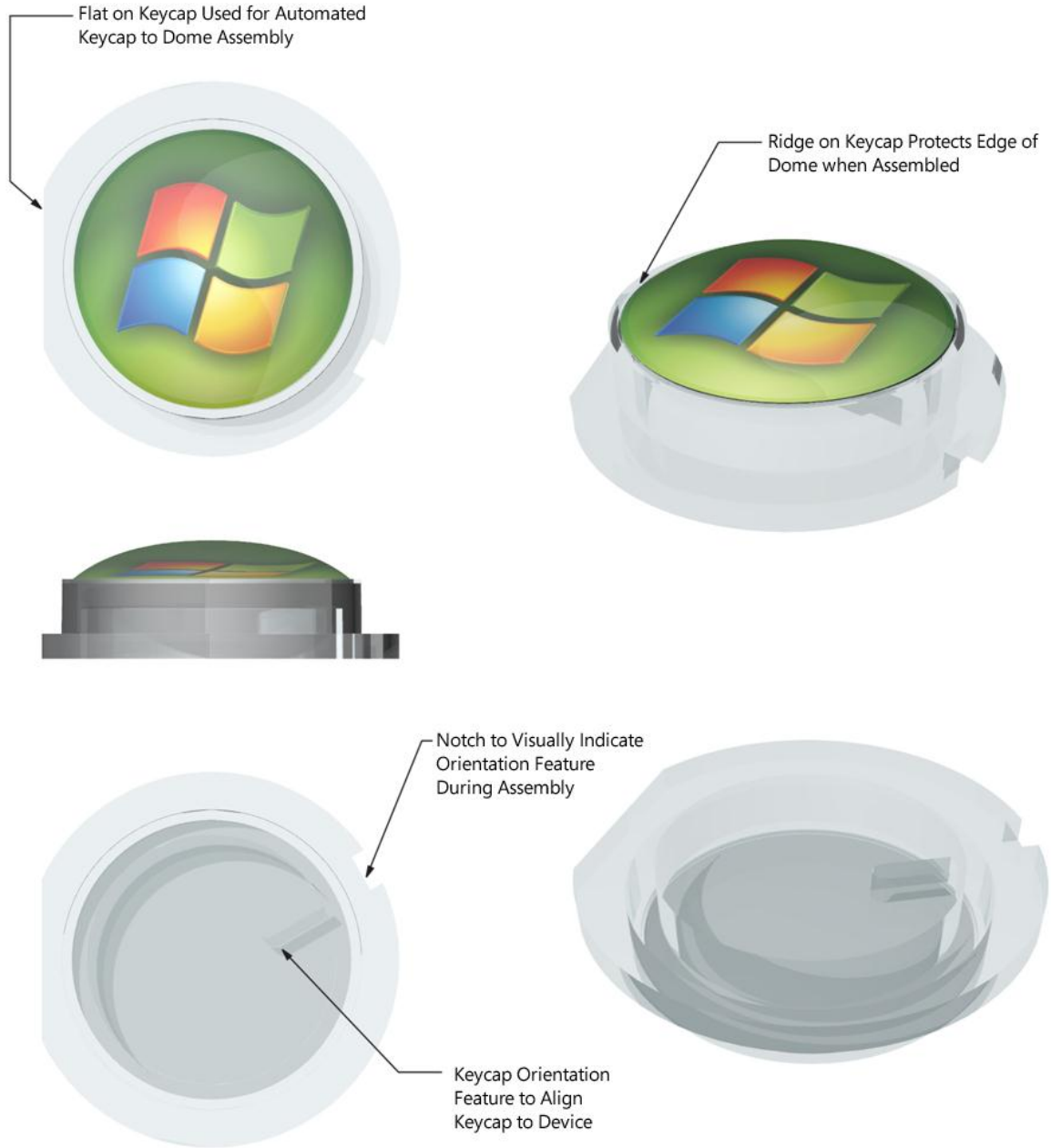


Figure 9: Illustration of subassembly with ridge and orientation features

Green Start Button Subassembly Must Be Obtained from a Certified Supplier

To achieve the Microsoft standard of consistency and quality in the Green Start button subassembly, device manufacturers must obtain the entire part from a certified supplier. Each certified supplier must manufacture the Green Start button subassembly according to requirements specified by Microsoft. The supplier that is certified to manufacture and distribute the Green Start button subassembly is listed in the “Resources” section later in this document under the heading Certified Supplier.

For detailed drawings of the Green Start button subassembly that is provided by the certified supplier, look in the “Resources” section under the heading Art Files for Green Start Button Subassembly. A STEP file and an Encapsulated PostScript (EPS) file is provided for each button size.

Device Housing and Orientation

Design Requirements for Chamfer on Device Housing

The area on the device housing around the Green Start button assembly contains a sharp-edged chamfer (or groove) that slopes down to the level of the keycap ridge.

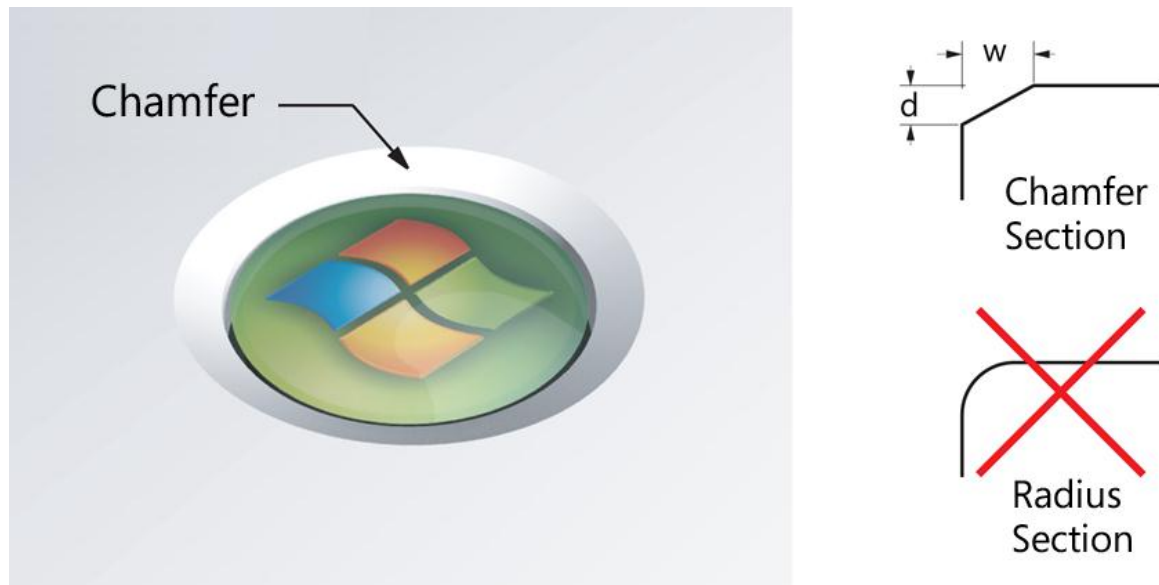


Figure 10: Illustration of chamfer on device housing

The design requirements for the chamfer on the device housing vary and depend on the button size. The following sections and diagrams illustrate the design and measurements for the three authorized button sizes.

Device Housing Design for 11 mm Green Start Button

The chamfer on the device housing should measure 15.05 mm in diameter and 0.75 mm in depth. The chamfer outer rim must be a sharp edge that is not rounded.

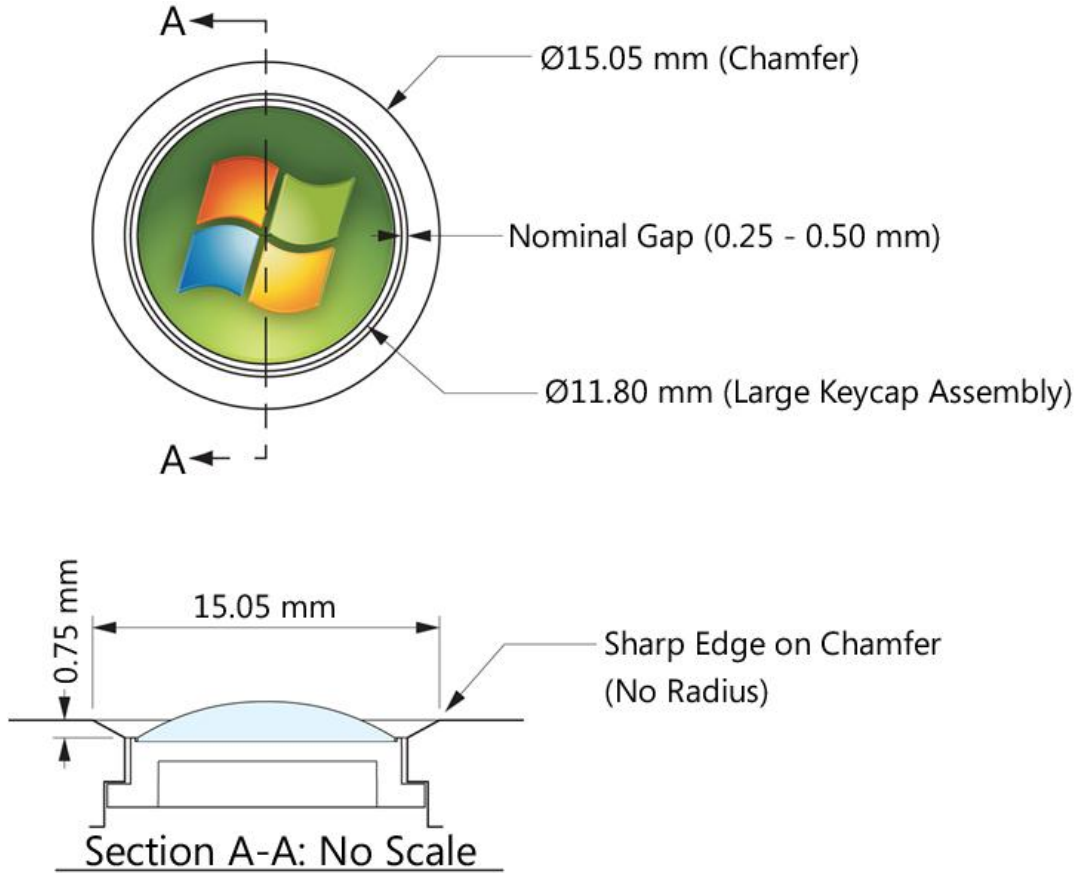


Figure 11: Illustration showing device housing detail for 11 mm Green Start button

Device Housing Design for 9 mm Green Start Button

The chamfer on the device housing should measure 12.55 mm in diameter and 0.65 mm in depth. The chamfer outer rim must be a sharp edge that is not rounded.

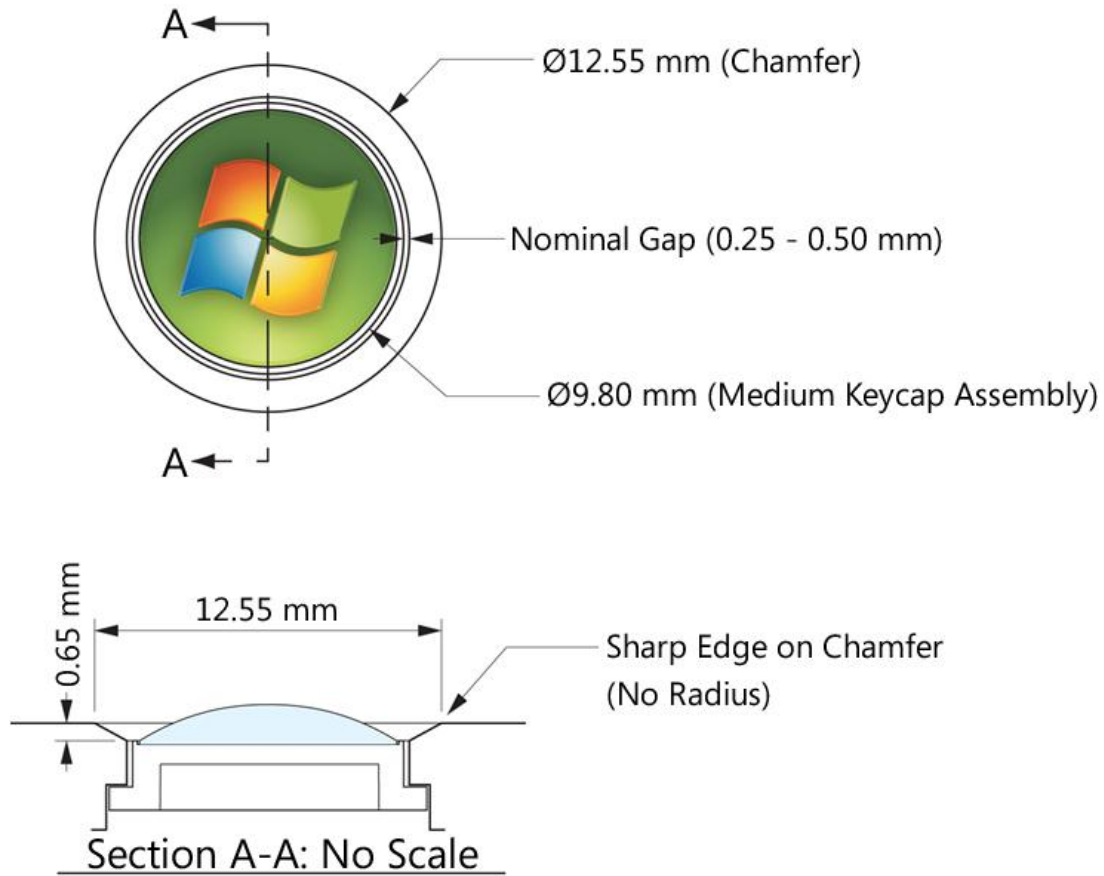


Figure 12: Illustration showing device housing detail for 9 mm Green Start button

Device Housing Design for 6.6 mm Green Start Button

The chamfer on the device housing should measure 9.4 mm in diameter and 0.5 mm in depth. The chamfer outer rim must be a sharp edge that is not rounded.

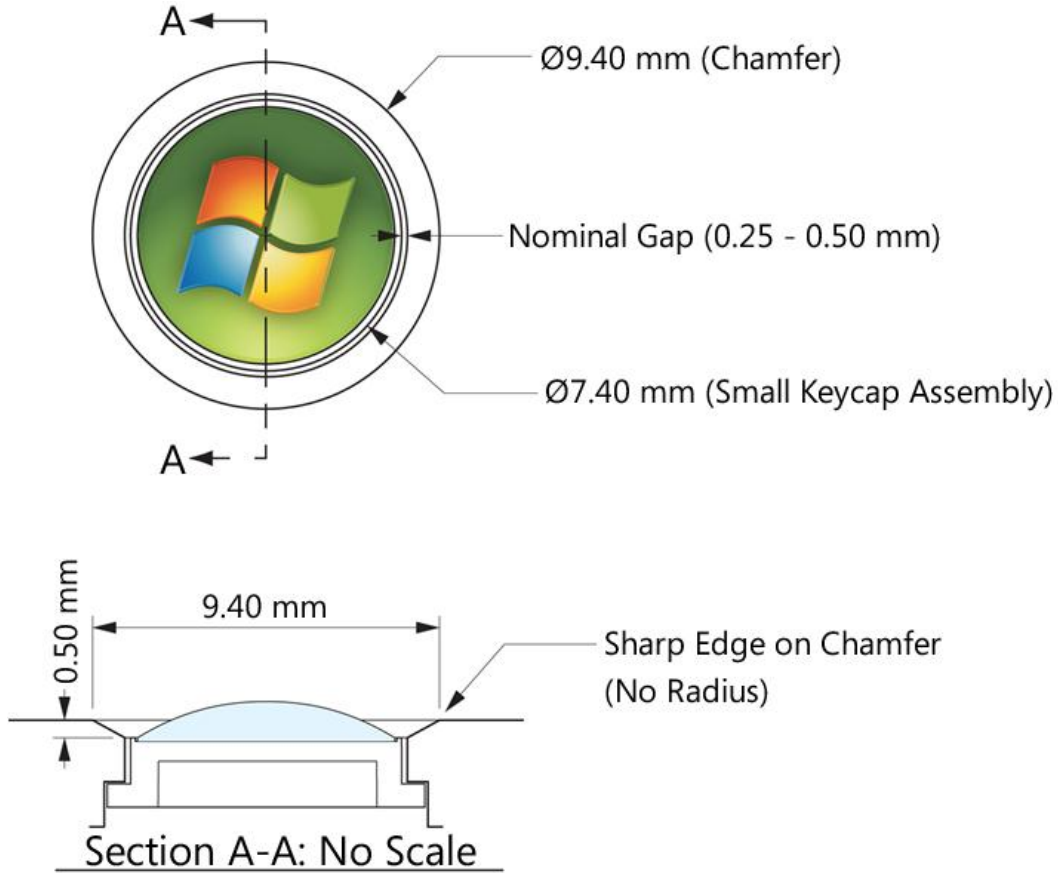
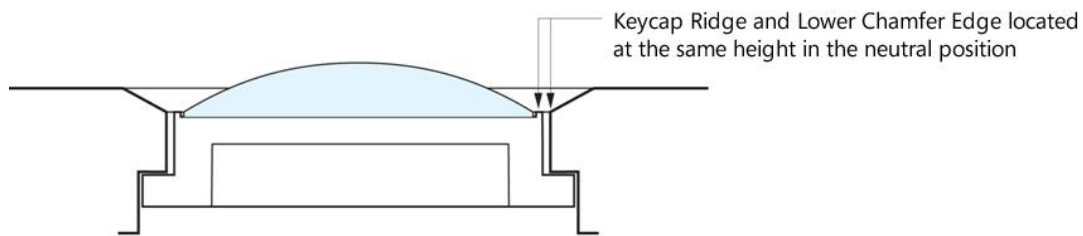
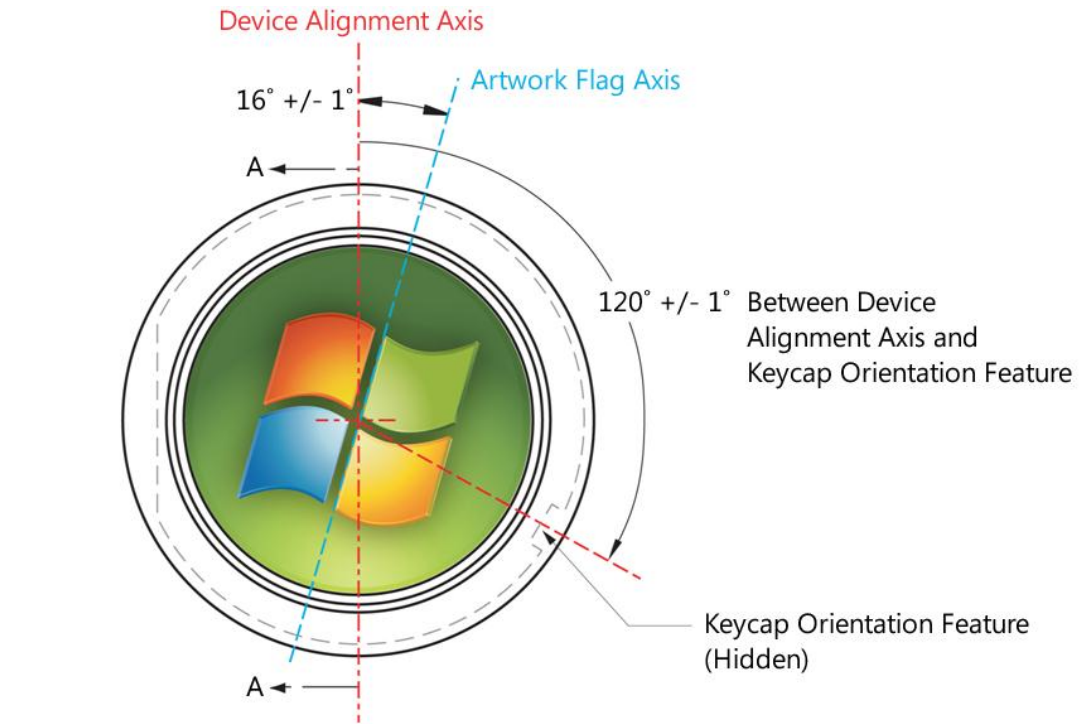


Figure 13: Illustration showing device housing detail for 6.6 mm Green Start button

Green Start Button Must Be Correctly Oriented on the Device

As shown in Figure 14, the orientation of the Green Start button subassembly to the device alignment axis must be held to ± 1 degree when assembled. Also, the keycap ridge and lower chamfer edge must be at the same height when the button is in the neutral position.



Section A-A: No Scale

Figure 14: Illustration of Green Start button correctly oriented to the device axis

The Green Start button is correctly oriented when the red portion of the Windows logo flag appears in the upper-left corner when the device is upright. Refer to the device specification for details on the correct orientation of the Green Start button. See Figure 15 for an example of a Green Start button that is correctly oriented to the device axis on a remote control.

Device Alignment Axis

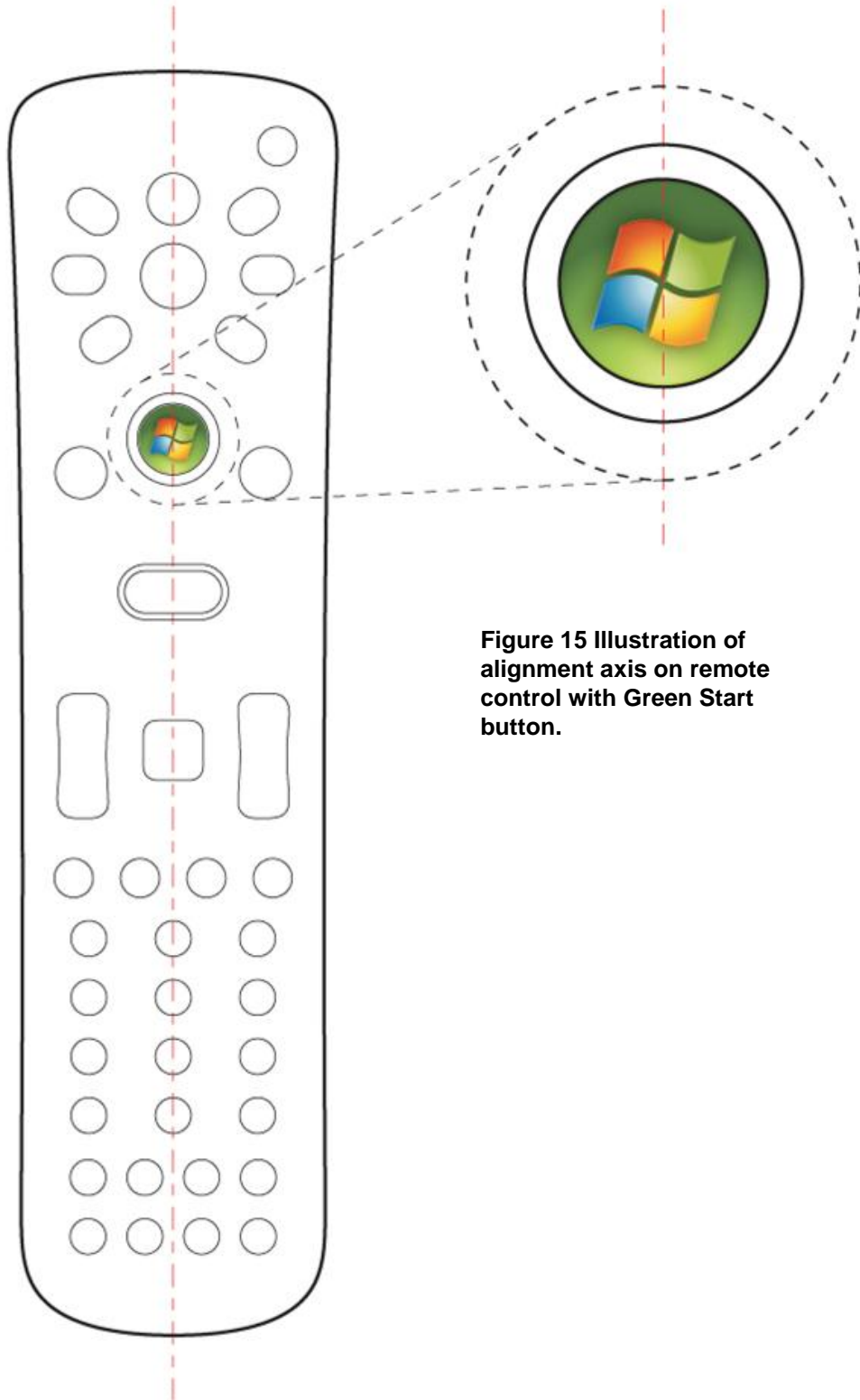


Figure 15 Illustration of alignment axis on remote control with Green Start button.

Remote Control (Top)

Do Not Place Additional Device Features Within a Recommended Proximity of the Green Start Button

No additional features, buttons, design details, graphics, or colors should be placed inside the specified white space that surrounds the Green Start button on the device housing.

The specification for white space diameter is as follows:

- Recommended: 190% of Green Start button assembly diameter.

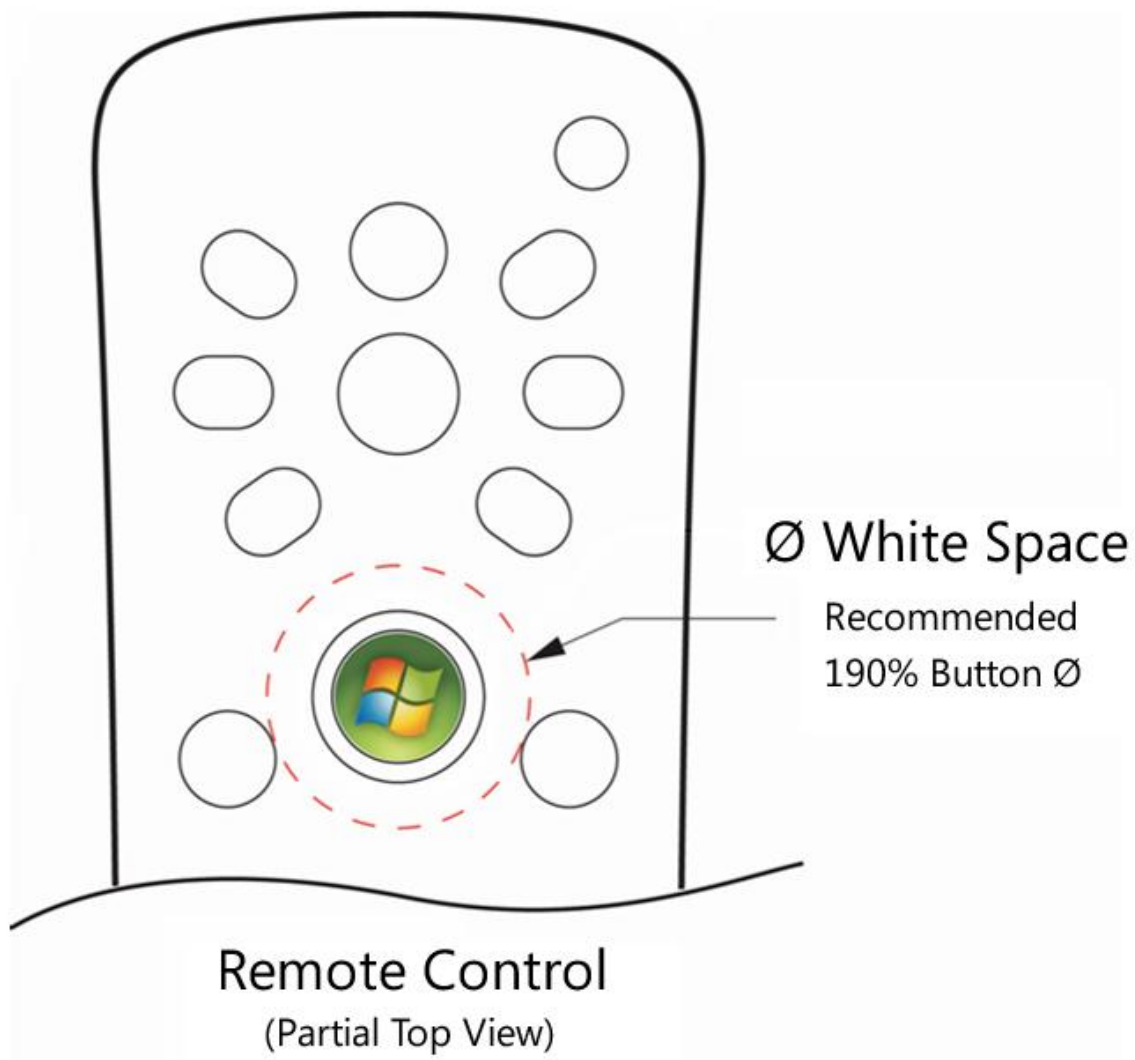


Figure 16: Illustration of a button layout that provides the recommended diameter of white space surrounding the Green Start button on a remote control

Green Start Button Travel Must Fall Within a Recommended Range

The keycap and dome subassembly must be positioned on the device housing so that the button travel falls within a recommended range. The recommended range of button travel for a rubber

membrane is 0.5 mm-1.0 mm. For a dome or microswitch, the recommended button travel distance is 0.3 mm-0.6 mm.

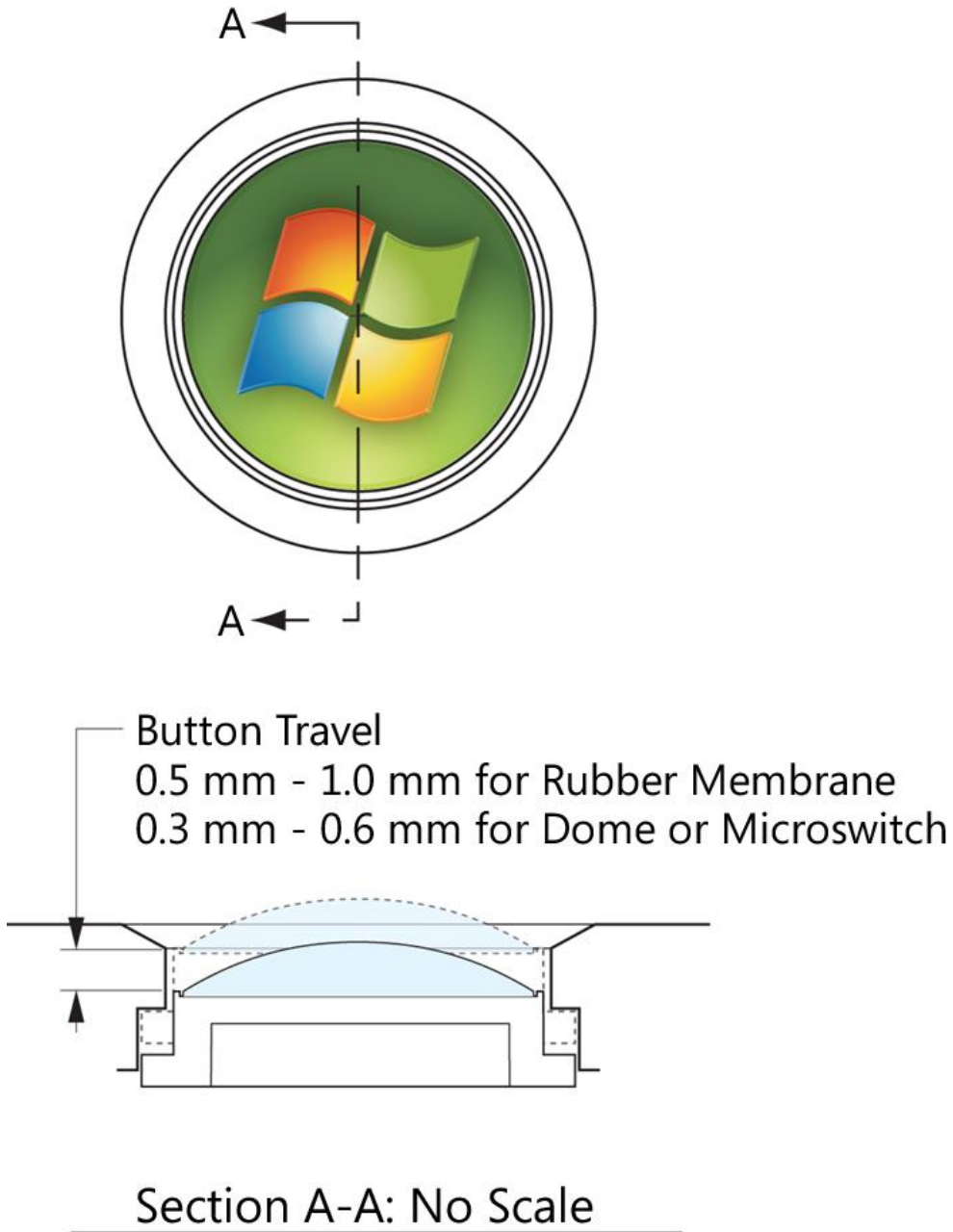


Figure 17: Illustration showing button travel distance parameters

Sample Implementation and Design Variations

Sample Implementation: 11 mm Green Start Button on Rubber Membrane

One construction that can be used to implement the Green Start button is to place the keycap and dome on a rubber actuation membrane. The rubber membrane must include an orientation feature to align with the orientation feature on the underside of the keycap.

Alignment of the orientation features on the keycap and rubber membrane ensures that the Green Start button subassembly is oriented to the device axis correctly. The following drawing illustrates this type of construction.

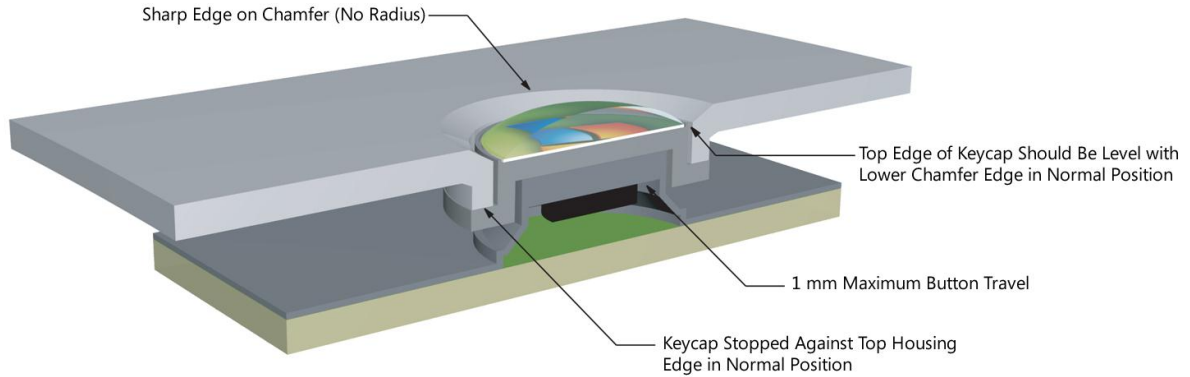


Figure 18: Illustration of rubber membrane construction

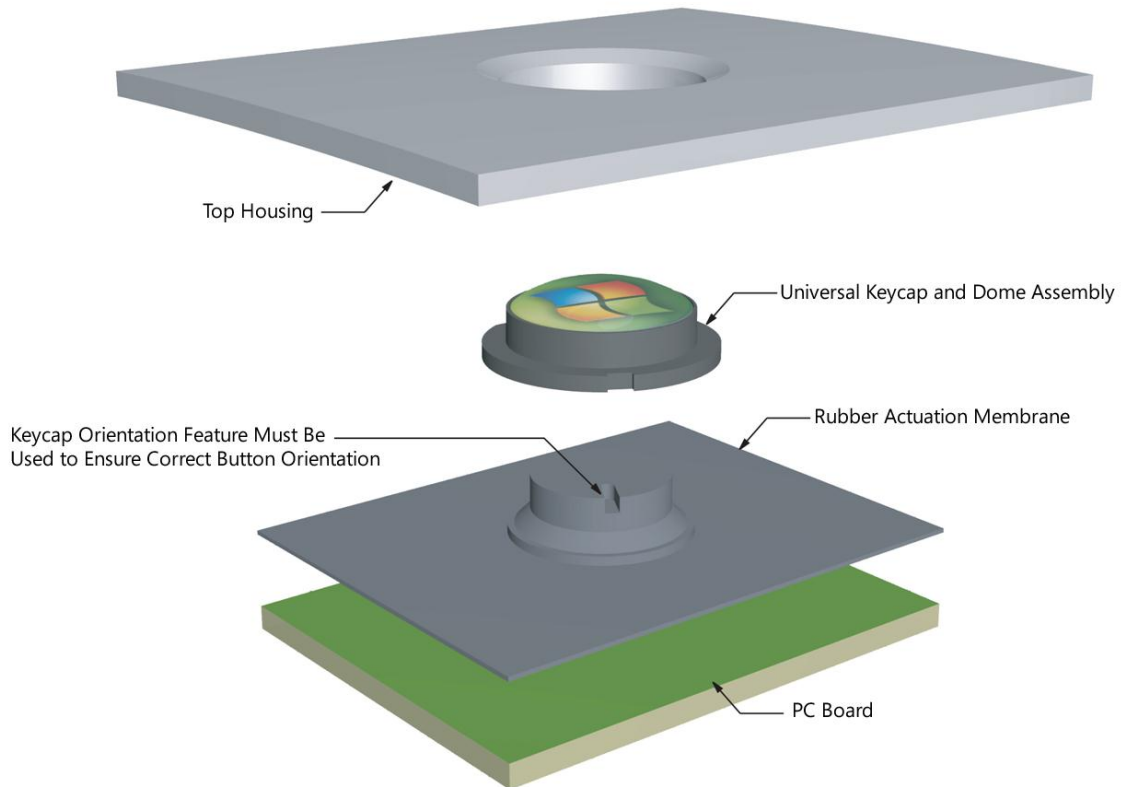


Figure 19: Exploded view of rubber membrane construction

Design Variation: Green Start Button with Custom Keycap

If the Green Start button subassembly does not meet your device design requirements because of incompatibility with device depth, switch type, or general device housing and design, a custom keycap and subassembly can be constructed using the requirements that are shown in Figure 20. While the ridge and dome dimensions shown in Figure 20 are required, the remaining attributes of your custom keycap are up to you.

All custom keycap designs for the Green Start button subassembly must be manufactured and obtained through a certified supplier. The supplier that is certified to manufacture and distribute the Green Start button subassembly is listed in the “Resources” section under “Certified Supplier for Green Start Button Subassembly”, later in this document.

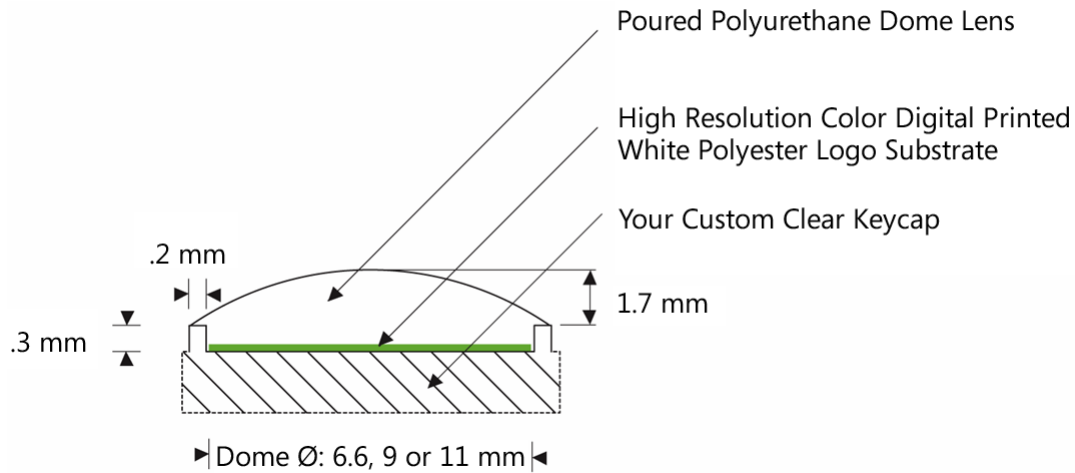


Figure 20: Illustration showing an example custom keycap with a ridge and poured dome lens

Design Variation: Glowing Translucent Chamfer Insert

Optional design variations can be implemented on the device housing to add emphasis to the Green Start button. For example, adding a transparent chamfer piece with a backlight around the Green Start button can create a glowing chamfer effect. The recommended color for backlighting the transparent chamfer is green.



Figure 21: Illustration of a translucent chamfer insert when it is not glowing

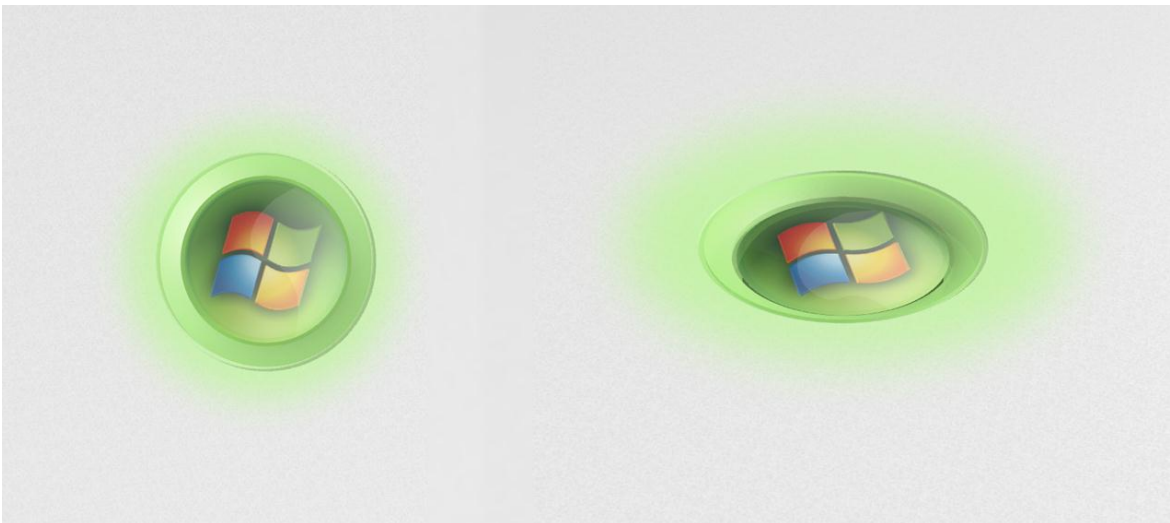


Figure 22: Illustration of a translucent chamfer insert when it is glowing

Design Variation: Membrane-Based Remote Control

Some Windows Media Center PCs that are laptop computers are paired with membrane-based remote controls (often referred to as “credit-card size” or “thin-profile” remotes) for increased portability. The construction and design of the thin-profile remote requires a different Green Start button design than the one described in this specification. As shown in Figure 23, the membrane-based remote requires a screen-printed Green Start button instead of a keycap and dome implementation.



Figure 23: Illustration of a membrane-based remote control with a 9 mm Green Start button that is printed directly on the top surface

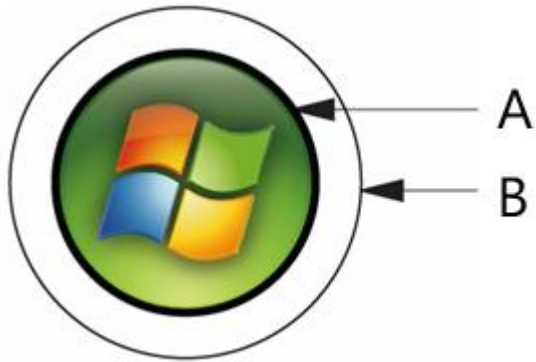


Figure 24: Visual treatment for the Green Start button image on a membrane-based remote control

An Adobe Illustrator file and an Encapsulated PostScript file of the authorized image for use in printing on the top surface of the remote control are provided in the “Resources” section under Art Files for Green Start Button for Membrane-based Remote Control. In this artwork, line A represents the edge of the button surface, and line B represents the edge of a simulated chamfer. Line B is optional and is used to mimic the keycap and dome implementation more closely. If you apply this artwork to a domed button surface, the green portion of the artwork should extend across the domed surface of the button, while the surrounding line work should appear outside of this domed or active surface. The Green Start button artwork that is provided can be scaled to any size between 6.6 mm and 11 mm. However, the Green Start button image cannot be smaller than the largest button image for any other buttons that appear on the remote control.

Alternate device design implementations must be reviewed and approved by Microsoft Corporation. For more information, please contact the primary Microsoft contact for that device specification.

Design Variation: Screen-Based Green Start Button

A screen-based Green Start button can be used for devices with a small digital display instead of buttons. For example, a remote control might have a digital touch-screen instead of buttons. The image of the Green Start button must not be smaller than 9 mm on the screen so that the end user can find the Green Start button quickly.

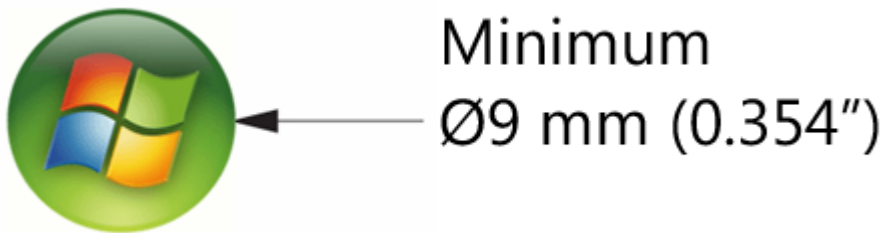


Figure 25: Illustration of a screen-based Green Start button (illustration not to scale)

Artwork for use in a screen-based implementation can be found in the file provided in the “Resources” section under Art File for Screen-based Green Start Button. Alternative device designs must be reviewed and approved by Microsoft Corporation. For more information, please contact the primary Microsoft contact for that device specification.

Resources

Certified Supplier for Green Start Button Subassembly

Company	The Auld Company
Address:	180 Outerbelt Street Columbus, Ohio 43213-1527 USA

Company	The Auld Company
Fax:	(614) 892-2929 back-up fax (614) 755-2329
Customer Service:	Eva Crompton
Phone:	(614) 755-2853 extension 2120
E-mail:	ecrompton@auldco.com
Sales Representative:	Dan Auld
Phone:	(614) 755-2853 extension 2200
E-mail:	dauld@auldco.com

Art Files for Green Start Button Subassembly

Files for 11 mm button:

- MCB_11mm_Universal_keycap_ASM.stp
- MCB_11mm_Universal_keycap_ASM.eps

Files for 9 mm button:

- MCB_9mm_Universal_keycap_ASM.stp
- MCB_9mm_Universal_keycap_ASM.eps

Files for 6.6 mm button:

- MCB_6_6mm_Universal_keycap_ASM.stp
- MCB_6_6mm_Universal_keycap_ASM.eps

Art Files for Green Start Button for Membrane-Based Remote Control

- Adobe Illustrator file: Printed_Thin-profile_Button_Art_CMYK.ai
- Encapsulated PostScript file: Printed_Thin-profile_Button_Art_CMYK.eps

Art File for Screen-Based Green Start Button

- Portable Network Graphics (PNG) file: Screen_Button_Art_RGB.png

Receiver/Transceiver Specifications

This section provides information about building receivers and transceivers that receive input from a control and convert them into actions to control Windows Media Center and a set-top box in the case of a transceiver.

Overview of IR Receiver Options

Deciding what type of infrared receiver to build for Windows Media Center can be confusing. There are several different options available:

- Options that use Microsoft provided Hardware designs and Microsoft provided software.
- Options that use Microsoft provided software with third-party hardware designs.
- Options that use third-party software and third-party hardware.

Each option has different tradeoffs, including things such as:

- BOM cost
- Hardware design cost
- Software design cost
- Functionality provided
- Risk

Additionally, not all options are suitable for all locales. Finally, many of the options are only suitable for Windows 7 and may not be distributed with any previous Windows Media Center products (such as Windows Vista or Windows XP Media Center Edition).

This section discusses these options, tradeoffs, and when specific options are allowable and not allowable.

Concepts

IR Protocol

The IR protocol defines how a series of infrared light flashes can be used to encode a remote control keypress, a keyboard event, or a mouse move. Windows Media Center supports several different protocols, including, but not limited to, Philips RC-6, SMK QP and an internal protocol called MCIR-2 which is used for the Windows Media Center infrared keyboard.

In designing your hardware, it is important to realize that, although RC-6 and SMK QP are the most common for an IR receiver, your receiver needs to do more than receive RC-6 and SMK QP. You also need to:

- Process IR input with *any* protocol that is supported by Windows Media Center. Software decoding is necessary.
- Implement IR learning.
- Implement IR emitting.
- Wake on certain IR signatures (power key wake).

Hardware Decoding Versus Software Decoding

In designing an IR receiver, there are two options: decoding the IR protocol in hardware and decoding the IR protocol in software.

- Decoding in hardware requires the details of the IR protocol to be stored in the hardware, either in firmware or in discrete logic. The hardware is hard-wired for a specific protocol or a set of protocols. Any change in the protocol or in the behavior of the protocol requires a change to the hardware.
- Decoding in software allows the details of the IR protocol to change with a software patch. In this case, the hardware returns the timing of the IR flashes to the software and the software interprets the protocol and converts the flashes into a keypress.

In order to do software decoding of protocols, you need to use a legacy device, an emulator device, or a port/class device.

In almost all cases, Windows Media Center requires software decoding of the protocol.

Multiple Software-Based Protocol Support

Because Windows Media Center supports decoding of the IR protocol in software, Windows Media Center can receive input from multiple types of remotes using the same hardware. This means, for instance, that the same receiver can receive input from a Philips RC-6 remote and also from a Windows Media Center IR Keyboard

Run Length Coding (RLC)

Run Length Coding (RLC) is the method used by Windows Media Center to communicate infrared information from IR receiver to the software decoders. RLC communicates IR pulses as a set of durations. These numbers are either duration on, which is the time that a (modulated) IR signal is present, or duration off, which is the time that there is no IR signal present. The numbers are represented in a count of microseconds, with a positive number indicating that the IR signal is present and a negative number indicating that the IR signal is absent.

So for this (demodulated) IR signal, which represents a single RC-6 keypress, the corresponding RLC is as follows:

2656 -888 444 -444 444 -444 444 -888 444 -888 1332 -888 444 -444 444 -444 444 -444 444 -444 444 -444 444 -444 444 -
444 444 -444 444 -444 444 -444 888 -444 444 -444 444 -444 444 -888 444 -444 444 -444 444 -444 444 -444 888 -888 444 -
444 444 -444 444 -444 444 -444 444 -444 444 -444 444 -444 444 -444 444

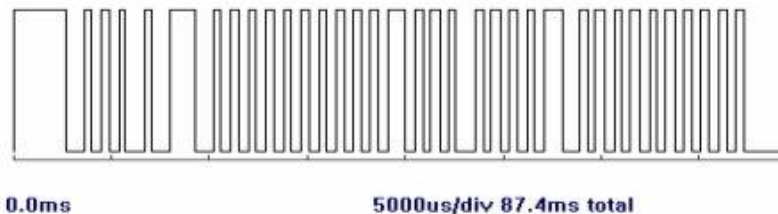


Figure 26: Demodulated IR signal

This means that the signal was high for 2656 usec, then low for 888 usec, then high for 444 usec, and so on. The one piece of information missing here is the carrier frequency. Depending on the situation that the RLC was used in, the carrier is either implied, explicit, or undefined.

It is important to recognize that your hardware does not need to send RLC to the software decoders in this exact form. There is a piece of software between your hardware and the software decoders that can convert the IR from some hardware-specific form into RLC. This software is called the “Port Driver”.

Port Driver/Class Driver Model

Windows Vista and Windows 7 supports the concept of an IR Port Driver/Class Driver model. In this model, there are two drivers installed in the PC. One is a class driver, which is provided by Microsoft. The class driver, called `circlass.sys`, can communicate with port drivers, and can also decode IR protocols. The port driver is typically provided by the hardware manufacturer, and can communicate to the hardware and send the IR information, in RLC form, up to the class driver.

IR Emitting

In addition to receiving IR, Windows Media Center must also transmit IR. IR is used to control cable boxes and satellite boxes, which are referred to collectively as set-top boxes (STBs). Essentially, the Windows Media Center PC acts like a cable-box remote control and transmits the IR necessary to control the cable box. This is needed to record TV shows when the user is not present, and to change the channel on the cable box if the user is using Extender for Windows Media Center technology for another room.

The terms “IR Blasting”, “IR Emitting”, “Transmitting IR”, and “Sending IR” are all used to describe this process. These terms are synonymous.

IR Database

An IR database is stored within the Windows Media Center PC, which can be used to control an extensive list of set-top boxes (STBs). For each STB manufacturer there is a list of IR codesets that is unique to them. These are generally stored as RLC so Windows Media Center transmits this information to control the STBs.

IR Learning and Parse-and-Match

IR Blasting, Learning, and Parse-and-Match must use the Windows Media Center IR Database. Windows Media Center is unable to take advantage of any other IR databases, particularly those stored inside of existing IR hardware solutions.

To control STBs, Windows Media Center must know which IR codes to transmit. There are three ways that Windows Media Center can discover which codes to transmit:

- **Choose from a list.** The user has the option to choose a manufacturer and code set number from amongst the IR database provided. This is very error prone because, a brand of STB might use 5 different IR code sets depending on the STB model. The user would have to select the manufacturer, and then manually try each of the 5 code sets until they found one that worked.
- **Parse-and-Match.** With the remote control, the user can use Parse-and-Match to identify the codeset. Windows Media Center prompts the user to press a key on the remote control, for instance the “zero” button. Windows Media Center receives the RLC for the IR from the IR hardware and compares it to all the codesets in the entire database. When Windows Media Center finds a matching codeset, it uses that codeset to control the cable box. This allows the user to find the correct codeset without the overhead of trying each codeset until a working set is found. Typically, Parse-and-Match should find the users’s codeset with 1-3 keypresses.

- **IR Learning.** If Windows Media Center is unable to identify the user's remote control using Parse-and-Match, then the user needs to go through the IR learning process. In IR learning, Windows Media Center captures the RLC for each keypress and stores it in a database. Windows Media Center needs at least two samples of each key to complete IR learning. Windows Media Center also needs to measure the carrier frequency for the remote control.

Note Windows Media Center does not support learning toggle bit remotes.

Long-Range Receivers and Wide-Band Receivers

Windows Media Center IR receivers need to have two different light-detecting components. One of them (long-range) is used most of the time. The other one (wide-band) is only used for IR Learning.

Long Range Receiver:

- BPF, AGC, and demodulator inside receiver hardware.
- Receive IR Data at 10 meters on center of receiver and 5 meters off center at a distance of 10 meters.
- Returns IR waveform envelope to software for software decoding of IR signal.
- Used for normal day-to-day operation and also for one-time setup of STB control.

Learning Receiver:

- Returns modulated signal to hardware so hardware or port driver can measure carrier frequency.
- Optimized for a distance of 5 centimeters.
- Returns IR waveform envelope to software for software decoding of IR signal.
- Used only for one-time setup of STB control.

Sleep (formally Power) Key Wake

Windows Media Center IR receivers need to support remote wake using a Sleep key on the remote control. This means that, if the PC is in a low power state, the user can use the Sleep or Wake button on the remote control to bring the PC into a higher power state. This is a requirement for all Windows Media Center IR receivers.

The Remote Control Functionality Needed

When designing a system, you have multiple choices about the level of remote control functionality that you can provide to users. This decision will be based on the type of PC that is being built, the peripherals that are being distributed with the PC, the country/region that the PC is being distributed in, and the level of functionality desired by the system designers.

Transmit/Receive Devices

Transmit/Receive devices provide the full set of IR functionality for users. They are able to receive IR input from a remote control and they provide Power Key Wake functionality for the users. In addition to this, they provide IR learning, remote identification using parse-and-match, and IR emitting functionality to control set-top boxes. In many countries or regions, a transmit/receive device is required for most Windows Media Center systems.

Transmit/Receive devices are required when the following three conditions are true:

- The system is a desktop system.
- The system includes a tuner.
- The tuner is capable of supporting set-top boxes. This depends on the video standard used and situation in the country/region where the system is being distributed to.

Transmit/Receive devices are allowed, but not required in other situations, such as laptop systems, and systems without a tuner device unless a remote is distributed with the system, then receivers are required.

Receive-Only Devices

Receive-only devices are devices that accept input from a remote control and are able to wake on the Sleep key, but are unable to do IR emitting. Because of the more limited set of functionality, receive-only devices are cheaper to produce, but they are also only able to distribute in a limited subset of PC systems.

Receive-only devices are useful when remote input functionality is desired, but the three conditions above don't require a transmit/receive devices. This includes, but is not limited to, these three common examples:

- Laptop systems.
- Systems without a tuner.

Receive-only devices can be IR based or RF based.

How Should You Build Your Device

After you decide what functionality your receiver needs, you will need to decide what hardware and software architecture to use when building your device. This decision should be based mostly on cost and risk, but can also be based on the level of functionality required.

Legacy Devices - Beanbag/Snowball/Snowflake

Before Windows Vista, legacy devices were the only option for building IR hardware. Microsoft provided the hardware reference designs, the firmware, and the software drivers. ODMs are required to use all of these and to follow the designs exactly. Because the design work is done and the software is provided, this has the cheapest design cost for ODMs. This also makes this the least risky option. However, because ODMs need to follow designs exactly, the BOM cost for this is fixed and moderately high. Legacy devices are the only option if OEMs are building systems that run Windows XP.

To build a legacy device, ODMs need to acquire the device schematic from Microsoft and build the device following the schematic and BOM requirements. Testing the hardware is possible using the drivers provided with Windows XP Media Center Edition and later versions of Windows operating systems with Media Center. The software for legacy devices is distributed with Windows XP Media Center Edition and later versions of the Windows operating system with Media Center so no additional software download/install is necessary for customers.

Legacy Device Summary

Design Cost	Low
BOM Cost	High
Risk	Low

Legacy Device Summary

Support for ODM customization	Low
Operating System Support	Windows XP or later versions of Windows with Media Center
Connection	USB Only

Emulation Devices

“Emulation devices” are devices which emulate the firmware of the legacy devices. For these devices, the software is provided by Microsoft, but the hardware and firmware design is entirely up to the hardware developers. Because the software is entirely provided by Microsoft, the hardware must communicate with the software in a fixed format. This option allows ODMs to focus on cost-reducing the hardware without incurring the cost of producing any software.

To build an emulation device, the OEM has to design and build the hardware according to the IR Receiver Hardware Requirements noted in this document. OEMs must program the firmware Wake key when the PC is in low power state. OEMs will not need to provide any software drivers. The software for legacy devices is distributed with the Windows operating system, so no additional software download/install is necessary for customers.

Emulation Device Summary

Design Cost	Moderate to High
BOM Cost	Up to ODM
Risk	Low to Moderate
Support for ODM customization	Low to Moderate
Operating System Support	Windows Vista or later
Connection	USB Only

Port Driver Devices

Port driver devices allow the hardware manufacturer almost complete freedom in designing their hardware. The cost of the freedom is that the manufacturer must provide a piece of software – a port driver – that can communicate with the hardware. The software is moderately difficult to write, but it allows the OEM freedom to do things such as:

- Connecting via busses other than USB
- Integrating the IR receiver into other peripherals
- Adding additional functionality into the device

Port driver devices give the OEM the greatest degree of freedom, but they incur the greatest amount of risk and the greatest up-front design cost. In many cases, building a port-driver device is the best choice, but the decision should not be taken lightly.

To build a port driver device, the OEM has to design and build the hardware according to the IR Receiver Hardware Requirements. OEMs need to design the firmware to communicate with the PC using whatever communication they desire. OEMs need to write a port driver that runs on the PC and communicates with the hardware. They need to program the firmware to respond to the Wake key when the PC is in low-power state. Because the OEM is responsible for writing the port

driver, they need to determine the best course for delivering the port driver to end customers. This may involve a driver disc that is distributed with the hardware or a software download.

Port-driver Device Summary

Design Cost	High
BOM Cost	Up to ODM
Risk	Moderate to High
Support for ODM customization	Moderate to High
Operating System Support	Windows Vista or later
Connection	Any

RF Receivers

These devices use Radio Frequency (RF) instead of IR to communicate the keypress information. The easiest way to build these is to use a USB connection and write the firmware to make the USB receiver appear to the operating system as a HID device.

To build an RF receiver device, the ODM needs to design and build the hardware. If the hardware appears as a HID device, they can use the in-box HID drivers. If the hardware does not appear as a HID device, they need to design and distribute the software drivers. They need to be aware of and account for the limitations below in the section “HID Device Limitations”.

RF Receiver Device Summary

Design Cost	Moderate
BOM Cost	Up to ODM
Risk	Moderate
Support for ODM customization	Moderate
Operating System Support	Windows XP Media Center Edition and later
Connection	Any (USB preferred)

IR HID Devices

IR HID devices decode the IR protocol in hardware and produce keystrokes (HID reports) directly.

IR HID Device Summary

Design Cost	Moderate
BOM Cost	Up to ODM
Risk	Moderate
Support for ODM customization	Moderate
Operating System Support	Windows XP Media Center Edition and later
Connection	Any (USB preferred)

More Complicated Receiver Examples

In addition to the basic device types above, it is possible to build more complicated receivers. These receivers appear very desirable at first glance, but they incur a great cost in terms of design cost, BOM cost, and risk.

RF Receivers – Transmit/Receive

These devices combine an RF remote with the ability to transmit IR. To do this, the OEM would basically take an existing IR transmit/receive device and add the ability for RF support. In order to control a set-top box it is not possible to completely remove the IR reception hardware from your device because it is required for Parse-and-Match or Learning functionality.

Remote Control and Receiver-Transceiver Specifications and Requirements
for Windows Media Center in Windows Operating Systems

The hardware for this device would be very expensive. It would cost as much as an IR device plus the additional cost for RF support. Because Windows XP doesn't support port driver devices, you would need to build a legacy device on a board with a USB hub chip and an RF receiver.

Two-Way Remote Devices

Two-way remote devices are devices where the communication between the remote control and the receiver goes in two directions. So, for instance, an LCD on your remote could display the currently playing song. In many ways, this is similar to the above item “RF receivers – transmit/receive” in that the device is expensive. This device would need to be a full function one-way receiver plus the added cost of the two-way protocol.

The hardware for this device would be very expensive. It would cost as much as an one-way IR device plus the additional cost for two-way support. Because Windows XP doesn't support port driver devices, you would need to build a legacy device on a board with a USB hub chip and a second device for the two-way functionality.

Connecting Your Receiver to the PC

If you are building a legacy or emulator device, you need to connect your device using USB. If you are building a port driver device, you have more freedom to decide how to connect your receiver to the PC. However, there are several caveats which make several choices expensive or impractical.

USB

This is the most common way to connect a device to the PC. It is most likely to work successfully and incurs the least risk.

IEEE 1394

1394 is possible, but incurs risk due to the untested nature of this scenario. 1394 may not allow the wake from remote feature.

PS/2

Using a ps/2 connection could work, but there may not be any advantage to this over USB. This would require a port driver on the PC. Making an IR receiver appear to the PC as a keyboard is not possible because existing keyboard drivers do not contain scan code mappings for Windows Media Center-specific keys.

Serial Port/Parallel Port

This would require a port driver on the PC. It would require an out-of-band signal to signal wakeup to the PC when the user presses the Sleep button on the remote. Because serial ports and parallel ports are fairly old technology, the likelihood of colliding with existing devices and software is fairly high. A large amount of integration testing would be necessary to ensure compatibility.

Super-IO Chip

Putting IR functionality onto a Super-IO chip is very desirable from a cost perspective for PC companies. This will require a port driver to make it work.

Integrated with TV Tuner (PCI or USB)

Integrating IR transceiver functionality with a PCI or USB tuner is very desirable from a cost perspective. It will require a port driver to make it work.

Bluetooth

To build a Bluetooth remote, you would basically be building a Bluetooth keyboard in a remote control form factor. This device needs to produce the proper HID codes and will have the same limitations outlined in the section “HID Device Limitations” below.

Things to Remember When Building Your Device

Note the following before you finalize your decision.

Wake from Remote

Wake from remote is the required feature that allows the user to put their PC to sleep using the Sleep button on their remote and then wake it up again using the Sleep button on their remote. This will have several effects on your hardware design:

- It implies that your hardware can draw current when the PC is asleep or off. This is necessary because your hardware needs to watch for and act on the Sleep button.
- It is the one case where your hardware needs to decode the two Media Center IR protocols in order to wake the system (as described in this document).
- It should ideally be field-programmable so that a given manufacturer could have their PC wake with the Sleep button on one protocol, and another manufacturer could have their PC wake with the Sleep button on another protocol

If you use a legacy device, this work is included in the Microsoft-provided design.

Emitter Detection

If you support IR blasting, you need to support an “Emitter Detection” feature. This means that your hardware/driver must be able to detect the presence of an IR emitter plugged into an emitter jack.

If you use a legacy device, this work is included in the Microsoft-provided design.

Two Receivers – Long Range and Wide Band

If your hardware supports IR emitting, it needs two IR receivers: a long-range receiver and a wide-band receiver. The long-range receiver demodulates in hardware, is centered on a particular frequency, and is designed to work ten meters away from the remote. The wide-band receiver doesn't demodulate in hardware, may not be centered on a particular frequency, and is designed to work five centimeters away from the remote.

If you use a legacy device, this work is included in the Microsoft-provided design.

Carrier Counting

When using the wide-band receiver, your hardware and/or port driver must be able to return the carrier frequency of the IR signal to the class driver. This does not need to be an instantaneous measurement – it merely says “for the last sample, the carrier frequency was about X KHz”. This can most easily be done by counting the number of leading edges in a given signal and dividing the time that the demodulated signal is high by the number of leading edges.

If you use a legacy device, this work is included in the Microsoft-provided design.

Not Only RC6 and SMK QP Protocols

The IR receiver that you're building needs to receive more than MC RC-6 and the MC QP protocols. Because the protocol is decoded inside of Microsoft-provided drivers, your receiver and your port driver don't actually need any details about these protocols. The only place you need to know about these protocols is to implement the wake-from remote function.

Windows Media Center will not be able to logo a receiver that only receives MC RC-6 and MC QP protocols.

Addressable Remotes

Remote controls need to be addressable. This is useful in situations where there are two Windows Media Center computers in a room. For instance, one remote can be set to transmit on channel #1 and the corresponding PC can be set to receive only channel #1. And the other remote can be set to transmit on channel #2 and the corresponding PC can be set to receive only channel #2. This way, the #2 remote will never control the #1 PC.

If you use Microsoft-provided IR drivers and the RC-6 protocol, this should not be of concern. If you are building a HID device or a RF remote, you will need to implement this functionality.

Receive-Only Doesn't Support Many Scenarios

Because of the proliferation of cable and satellite set-top boxes, the need to IR blasting is significant. The number of scenarios where a receive-only device is useful is fairly limited. The additional cost of producing a transmit/receive device may well be worth it considering the additional scenarios that this feature enabled.

Multiple Receivers on a Single Computer

You should take into account that there may be multiple IR receivers on a single computer. With IR receivers going into Super IO Chips, becoming integrated in TV tuners, being included inside PC cases, and with external USB receivers, it is very likely that a given user will have more than one IR receiver on their computer.

A receiver, such as a HID receiver, which blindly passes HID events to the operating system is likely to have problems in this area. If you use a legacy device, a port-driver device, or an emulation device, this is accounted for in the software and will not be a problem.

HID Device Limitations

If you are building a device that is not a legacy device, an emulator device, or a port driver device, you will face several limitations in the construction of your device. This is because your device produces HID codes directly without being filtered through our class drivers.

Device Incompatibility

If your HID device is IR, it is probably tied to a single protocol. Because Windows Media Center uses multiple IR protocols, there are going to be "Media Center Compatible" IR remotes that are

not compatible with your IR receiver. Even if you program your hardware to respond to all protocols that Windows Media Center uses today, you will not be able to program it for all protocols that Windows Media Center might support in the future. This will lead to support calls when users expect their “Media Center Compatible” remote to work with your “Media Center Compatible” receiver.

If your device uses a non-IR transport, then it will not work with any Windows Media Center IR remotes.

Multiple Keypress Bugs

With the increased number of options for building IR receivers, PCs that have multiple IR receivers are becoming more and more common. Without software to filter multiple keypresses, a single remote control press might be received by multiple receivers and passed up to the user interface multiple times.

Legacy devices, emulator devices, and port driver devices all go through a layer of software that prevents a single remote control key press from going to the user interface more than once.

HID devices don't go through this layer. As a result, two HID receivers may receive the same keypress and send it up to the user interface, resulting in one keypress per receiver.

Localization Problems

HID Usage Tables, which are defined by the USB Consortium, define number presses based on the layout of the keyboard. For example, if you press the “1” key on your remote control, the HID usage sent is tied to the key that is one row down and one key in on a typical American 101-key keyboard. Certain keyboard layouts, such as Hungarian and French, define this key differently. As such, if you press the “1” key on a remote control connected to a PC on a French or Hungarian PC, you may not actually get a “1” key press.

If you implement a device that sends HID codes directly, testing on all locales and with all keyboard layouts is necessary. You may need to limit the locales where you can distribute your remote control and you may need to produce localized remote controls for locales that have different keyboard layouts.

Registry Changes Necessary for Triple-Tap Operation (Windows Vista Only), IME, and Numeric Input for Windows 7 and Later

Most of the time, the Windows Media Center UI treats remote control button presses exactly the same as keyboard presses. One time when this is not the case is when the user is entering text into a search box using what Microsoft calls “Triple Tap” functionality. More generically, this can be thought of as a “remote control specific IME” (Input Method Editor). In this case, the Windows Media Center UI treats the remote control buttons differently. For instance, if you enter “2” on the remote, it will produce the number “2”. If you enter “2,2”, it will produce the letter “A”. If you enter “2,2,2”, it will produce the letter “B”, etc.

In order to support this, Windows Media Center determines when a specific keypress comes from a remote control by looking up the name of the receiver device in a registry table to see if it matches the list of names of all remote control devices.

To support this scenario with a HID device, you must provide a setup executable that sets this registry key. You will need to test input scenarios in the Windows Media Center UI and make sure they work correctly.

Further details on this feature are available from Microsoft.

IR Receiver/Transceiver Hardware Requirements

This section provides information about IR transceiver and receiver hardware for use with Windows Media Center. It provides requirements for hardware manufacturers and specifies the hardware parameters that IR receivers must support.

The following abbreviations are used in this section:

- BPF: Band Pass Filter
- AGC: Automatic Gain Control

Components of an IR Transceiver

The following sections provide information about the components of an IR transceiver.

Remote Control Input and a Long-Range IR Receiver (Up to Five Meters)

The Windows Media Center user interface is designed to be used with a remote control by an end user who is sitting up to 5 meters away from their Windows Media Center PC. The long-range IR receiver is used to process commands that are sent from the remote control and used to control and navigate through the Windows Media Center user interface. The typical distance between the remote control and IR receiver is typically 2 to 3 meters. However, the remote control and receiver must work properly at up to a minimum of 5 meters apart.

The long-range receiver is also used during the initial setup when the end user first starts Windows Media Center. The long-range receiver parses the remote control IR signal based on an IR data sample to identify the remote control. If Windows Media Center cannot identify a set-top box remote control based on the IR data sample, the user can then perform IR learning. Input functions are required for IR receivers or IR transceivers.

IR Learning (from Five Centimeters or Less)

IR learning is used to capture IR data from a set-top box remote control that is not listed in the Windows Media Center Licensed IR database. After the data is captured in the IR learning process, the data is transmitted using IR emitting to control the set-top box.

IR Emitting

IR emitting is used to send IR commands from the Windows Media Center PC to the set-top box to change channels. This means that the user needs only one remote control to control the Windows Media Center PC. IR emitting also enables Windows Media Center to change channels automatically so that TV shows are recorded as scheduled even when the end user is not present and using the Windows Media Center PC.

Additionally, IR emitting is used when an end user is using a Media Center Extender device to control and play content that is on the Windows Media Center PC. The TV signal and programming that are sent to the Extender device come from the Windows Media Center PC. When a set-top box is present, the Windows Media Center PC must be able to emit an IR signal to the set-top box to change channels when watching TV and using an Extender device.

IR learning is required if a manufacturer is building an IR transceiver.

System-Level Interaction

End users expect a Windows Media Center PC to perform in a way that is similar to other consumer electronics devices that you can control with a remote control. This includes the ability to use the remote control to wake the Windows Media Center PC after it goes into a standby mode. A system function enables the remote control to wake the Windows Media Center PC from a standby state. System functions are required for IR receivers or IR transceivers.

IR Transceiver Requirements

The following sections provide requirements for the components of an IR transceiver.

Remote Control Input and Long-Range Receiver Requirements (Up to Five Meters)

The following list provides remote control and long-range receiver requirements.

- Support carrier frequencies ranging from 30-60 kilohertz (kHz).
- Return IR waveform envelopes to software for decoding an IR signal by using software. The IR signal must not be decoded by using hardware except for using the remote control Sleep button to wake a Windows Media Center PC from a standby state.
- Receive IR data from up to 5 meters at both the center of the receiver and up to 2 meters off center.
- Have Band Pass Filter (BPF), Automatic Gain Control (AGC), and demodulator inside the IR receiver.
- Use BPF centered at 36-38 kHz.
- BPF passes optimized for a signal ranging from 30-60 kHz.
- Recommended to be optimized for 950 nanometers (nm) of light.
- Hardware must sample at 50 μ sec.
- No noticeable degradation in signal quality from a distance up to 5 meters from the center and from a distance and that is up to 2 meters off center (in an indoor room with fluorescent lights without excessive reflection in the room).
- Must be able to sample pulses as short as 216 μ sec with 378 μ sec space to recover between pulses at 36 kHz.
- Must be able to accurately receive Windows Media Center RC-6 protocol.
- Must be able to accurately receive Windows Media Center SMK QP protocol
- Must be able to accurately receive Microsoft Media Center IR Keyboard input.
- The receiver must operate in the 33-50% duty range because most IR signals operate in this range.
- The receiver may introduce up to a roughly 200-250 μ sec error when measuring a 600 μ sec pulse or space.

IR Learning Requirements (from Five Centimeters or Less)

The following list provides IR learning requirements.

- Support carrier frequencies ranging from 30-60 kHz.
- Be able to capture a raw IR data stream.
- Be able to capture an IR carrier frequency.
- Be able to respond to the IOCTL to flash the receiver's LED.

Transmitting IR Data (Emitting) Requirements

The following list provides IR data (emitting) requirements.

- Support carrier frequencies ranging from 30-60 kHz.

- Support independent IR transmitter jacks (2 minimum).
- IR emitter cable should be a minimum of 2 meters in length.
- IR emitters are adhesive.
- IR emitters will provide the consumer with a visible LED.
- Support the Modulated IR mode. Modulated mode transmits a signal modulated with a 30-60 kHz carrier with a sampling resolution of 50 μ sec.

Pulse Mode Remotes

Pulse mode is no longer required.

System-Level Interaction Requirements

The following list provides system-level interaction requirements.

- Must resume from standby mode using the Sleep button for the particular IR protocol for which the hardware is optimized. Resume-from-standby must do hardware decoding of the protocol and operate when the Windows Media Center computer is in a state of lower power consumption.
- Able to wake from S1 or S3. Resuming or waking from S4 or S5 is optional.
- Receiver module should flash the LED when receiving IR.

Important This is highly recommended as a key user scenario.

- Must properly indicate a user presence to the operating system when waking the system. This can be tested by first waking the system with the remote control, and then by running a scheduled task. The monitor should turn on when waking with the remote control, but not when waking from running a scheduled task.
- Power consumption requirements are defined by the bus and architecture used by the IR receiver. For example, USB allows 2.5 mA during suspend and a variable amount of current depending on whether it is a high-power or lower-power device.
- If using a USB device, it is recommended that the device be able to operate correctly when it is plugged into a passive hub.

Emulation Requirements

This section describes how to build a USB consumer infrared receiver (CIR) for Windows Media Center in the next version of Windows. These devices use the IR Transceiver Version 2 or IR Receiver Version 3 wire (USB) protocol, but not the IR Transceiver Version 2 or IR Receiver Version 3 bill of materials. They are referred to as *IR emulation devices*.

Host System Requirements

Emulation devices are only supported by Windows Vista and future operating systems. Devices built according to this information are not supported for and will not work on Windows XP Media Center Edition.

To develop and test this functionality, you will need a Windows Vista or later operating system.

Device Driver Usage

IR emulation devices use the in-box Windows CIR device drivers, including `usbcir.sys`, `circlass.sys`, and `hidir.sys`. No device driver development is necessary to create an IR emulation device.

Device Design Considerations

Considerations for building an emulation device are very similar to considerations when building an IR receiver device using the port driver model.

Unique Serial Numbers

The software requires each device to have its own serial number. This is implemented using the USB **iSerialNumber** string descriptor. The format of this string is not mandated as long as it is unique to the individual device.

The serial number is necessary for the software to discriminate between multiple identical devices that are plugged into the same computer.

This serial number does not need to be truly unique as long as it is consistent. This means that the device could assign a semi-random number to itself the first time it is plugged in as long as it uses the same number on each subsequent use of the device. This allows the manufacturer to avoid serialization of serial numbers at device manufacture time.

Emulator Versioning

There are two versions of the emulator protocol. The first version (called `EMVER_EMULATOR_V1` in the code) was added for the Windows Media Center in the release of Windows Vista. The new version (called `EMVER_EMULATOR_V2` in the code) is added for Windows 7 to correspond with the Consumer IR Version 2 DDI changes to the IR class driver/port driver model.

An `EMVER_EMULATOR_V1` emulator should work correctly with the version 1 DDI and the version 2 DDI. An `EMVER_EMULATOR_V2` should work with the version 2 DDI and should *mostly* work with the version 1 DDI. It may expose device capabilities that the version 1 DDI is unprepared to take advantage of.

It should be assumed that all topics in this section refer to both the `EMVER_EMULATOR_V1` interface and the `EMVER_EMULATOR_V2` interface unless otherwise specified. If a section only refers to the `EMVER_EMULATOR_V2` interface, it will be labeled “`EMVER_EMULATOR_V2` only”.

For more information about how the host learns whether the device is `EMVER_EMULATOR_V1` or `EMVER_EMULATOR_V2`, see `CMD_GETEMVER` and `RSP_EQEMVER`.

Types of Emulation Devices

For version 1 emulation devices, there are two types of devices that you can build as emulation devices:

- IR transceiver emulator: a device that can receive IR input from a remote and can transmit IR to control a set-top box, such as a TV signal cable box or a satellite box.
- IR receiver emulator: a device that can only receive IR input from a remote; it cannot transmit IR to control a set-top box.

For information about the requirements for providing either transmitter or receive-only functionality, see sections Input-0007 and Input-0045 of the Windows Logo Program Hardware Requirements.

The version 2 emulation interface adds additional capabilities bits, which allow a wider range of devices, such as “blast only”, “learn only”, and so on. See `CMD_GET_DEVDETAILS` and `RSP_EQ_DEVDETAILS` for more information.

Carrier Capture

When using the wide-band receiver, the device firmware must return the carrier count for any particular sample. This carrier is an approximate value and applies to an entire sample. It is not necessary to return an instantaneous carrier value. So, for instance, if the software is using the wide-band receiver and asks the user to press the zero key, the firmware would send the RLC for the envelope up to the software, and then send the value of the carrier frequency for the entire key-press to the software.

The carrier information is sent from the firmware to the software as the “duration that the signal was high.” Details are in the response topic `RSP_EQIRRXFCNT`, later in this document.

USB 1.1 Devices

You can build an emulator device that is a Full Speed USB 1.1 device. For Full Speed devices, there is very little difference between USB 1.1 and USB 2.0. To make this work, you can have a Full Speed USB 1.1 chip, but the firmware must follow two rules that are newer for USB 2.0:

- You must set `bcdUSB` to `0x0200` in the Device Descriptor. If the value of `bcdUSB` is hard-coded to a different value in your USB chip, you cannot use that chip.
- You must respond correctly to the `device_qualifier` request. (For more information, see the comment that follows regarding Section 9.6.1 of the USB 2.0 specification.)

The 2.0 specification adds a third device-implementation choice, High-Speed, to the Full-Speed and Low-Speed device implementation choices. It does not eliminate the Full-Speed and Low-Speed device choices. A device can claim it is compliant with the 2.0 specification even if it is a Full-Speed or Low-Speed only device.

A USB 2.0 compliant device that is Full-Speed only is essentially the same as a USB 1.1 compliant device.

Section 9.6.1 of the USB 2.0 specification says this:

The DEVICE descriptor of a high-speed capable device has a version number of 2.0 (0200H). If the device is full-speed only or low-speed only, this version number indicates that it will respond correctly to a request for the `device_qualifier` descriptor (that is, it will respond with a request error).

This means that if a Full-Speed only or Low-Speed only device returns a Device Descriptor with `bcdUSB` set to `0x0200`, it should be prepared to receive a Get Descriptor request for a Device Qualifier Descriptor (see section 9.6.2 in the USB 2.0 specification) and if it receives this request, it should respond to this request with a Request Error, that is, a STALL. (See section 9.2.7 in the USB 2.0 specification.)

Modulated IR Protocol

- In modulated protocols, the IR signal is modulated with a carrier, typically around 30-60 KHz. The information in a given data packet is determined by the width of the pulses in the envelope (pulse width encoding), the width of the space between pulses in the envelope (space encoding), or the timing of edges in the envelope (bi-phase or Manchester encoding). Pulse and space widths in

the envelope are typically in the 200-900 μ sec range. About 95% of remote controls use some sort of modulated protocol. Emulator devices must support this protocol.

Flow Control

When transmitting IR, it is possible that the host will push IR data faster than the device can emit it. Instead of buffering this data, the device should return a negative acknowledge (NAK) handshake to indicate that it is not ready for the additional data yet. For more information, see section 8.5.1 of the USB 2.0 specification.

Bootloaders

Version 2 emulator devices must have a bootloader mode to support wake programming, which is a requirement of the Windows Logo Program. The bootloader is a second mode of the emulator device that the host can use to program the wake pattern. This mode is independent of the main operating mode and is discussed in the Bootloader Implementation section.

Two Methods for Wake Programming

The V2 emulator interface supports two different methods for wake programming. These methods are not necessarily mutually exclusive, but most hardware manufacturers will choose one or the other.

Payload-Based Programming

The first option is to program the wake pattern in your device based on payload. If you choose this method, you would program your device in response to a `CMD_BOOT_SETWAKEPATTERN` request. The `CMD_BOOT_SETWAKEPATTERN` gives your device the Protocol, Payload, and Address for the Wake key and it expects your firmware to watch for this key based on these numbers. This requires your firmware to be able to decode both the RC6 and Quatro Pulse protocols, and it expects this firmware to be parameterized in such a way that it can change its behavior based on these numbers.

This option allows a simple communication between the host and device, but requires a more complicated firmware.

Block-Based Programming

The second option is to program the wake pattern in your device based on a block of firmware. If you choose this method, you would program your device in response to a `CMD_BOOT_WRITEBLOCK` request. The `CMD_BOOT_WRITEBLOCK` request allows the host to send an arbitrarily-sized block of firmware to your device for each `<protocol,payload,address>` combination. This option is less demanding on the firmware that is in your device, but it requires you to install multiple firmware blocks into the user's registry (one block of firmware for each possible wake pattern).

This option allows for a simpler firmware implementation, but it requires the hardware manufacturer to install firmware blocks into the user's registry.

USB Device Descriptors

USB device descriptors are defined by the USB consortium. For information about definitions and usages of these descriptors, see the developer section of the USB 2.0 specification at <http://www.usb.org>.

Descriptors are defined using a pseudo-C syntax. Any specific values defined in this section are required by this specification. You may choose values that are not defined in the specification.

The type **word** is two bytes long and is little-endian, according to USB rules.

The host requests these descriptors through the EP0 OUT (control OUT) endpoint. The device returns these descriptors to the host over the EP0 IN (control IN) endpoint.

Device Descriptor

The device must have a device descriptor and must have at least one configuration. For more information, see section 9.6.1 in the USB 2.0 specification.

The following code shows a device descriptor structure:

```
typedef struct deviceDescriptor {
    byte bLength = sizeof(struct deviceDescriptor);
    byte bDescriptorType = TYPE_DEVICE_DESCRIPTOR;
    word bcdUSB = 0x0200; // USB 2.0 is required for Microsoft Compatible
                        // Device Descriptors to function properly.
    byte bDeviceClass = 0; // Not used.
    byte bDeviceSubclass = 0; // Not used.
    byte bDeviceProtocol = 0; // Not used.
    byte bMaxPacketSize; // This is hardware dependent. Choose a value
                        // that is appropriate for your USB chip.
    word idVendor; // Vendor ID - assigned by USBIF.
    word idProduct; // Product ID - assigned by your company.
    word bcdDevice; // Device release number.
    byte iManufacturer; // String index for manufacturer.
    byte iProduct; // String index for product name.
    byte iSerialNumber; // String index for serial number.
    byte bNumConfigurations; // Count of configurations.
                        // Must be at least 1.
}
```

Note *bcdUSB* must be set to 0x0200, even if the device is a full-speed USB 1.1 device. For more information, see the topic "USB 1.1 Devices" earlier in this document.

Configuration Descriptor

The device must have at least one configuration that has at least one interface. For more information, see section 9.6.3 of the USB 2.0 specification.

The following code shows a configuration structure:

```
typedef struct configurationDescriptor {
    byte bLength = sizeof(struct configurationDescriptor);
    byte bDescriptorType = TYPE_CONFIGURATION_DESCRIPTOR;
    word wTotalLength; // Total size of configuration data.
                    // See section 9.6.3 in USB 2.0 spec.
    byte bNumInterfaces; // Must be at least 1.
    byte bConfigurationValue;
    byte iConfiguration; // String index for configuration name.
    byte bmAttributes = 0xA0; // Remote wakeup required. May also be 0xE0.
    byte bMaxPower;
}
```

The device can support additional interfaces as necessary. These extra interfaces could be used, for example, to support a proprietary firmware download mechanism.

Interface Descriptor

The device must have at least one interface, which must have at least two endpoints: one control endpoint and one communication endpoint. For more information, see section 9.6.5 of the USB 2.0 specification.

For communication, the device can use a single bidirectional endpoint, as the examples in this document illustrate, or it can use two unidirectional endpoints.

The following code shows an interface descriptor structure:

```
typedef struct _interfaceDescriptor {  
    byte bLen = sizeof(struct _interfaceDescriptor);  
    byte bDescriptorType = TYPE_INTERFACE_DESCRIPTOR;  
    byte bInterfaceNumber = 0;  
    byte bAlternateSetting = 0;  
    byte bNumEndpoints = 2;  
    byte bInterfaceClass = 0xFF;  
    byte bInterfaceSubclass = 0xFF;  
    byte bInterfaceProtocol = 0xFF;  
    byte iInterface;  
}
```

Endpoint Descriptor – OUT Endpoint

The device must have an endpoint descriptor that describes how the host communicates infrared data and other information to the device. (Endpoint #0 is the control endpoint and does not require an endpoint descriptor.) For more information, see section 9.6.6 of the USB 2.0 specification.

The following code shows an OUT endpoint descriptor structure:

```
typedef struct _endpointDescriptorEp1Out {  
    byte bLength = sizeof(struct _endpointDescriptorEp1Out);  
    byte bDescriptorType = TYPE_ENDPOINT_DESCRIPTOR;  
    byte bEndpointAddress = 0x01; // EP1 OUT.  
    byte bmAttributes = 0x03; // Interrupt endpoint.  
    word wMaxPacketSize; // Hardware dependent.  
    byte bInterval; // Suggested value = 1ms.  
}
```

Endpoint Descriptor – IN Endpoint

The device must have second endpoint descriptor that describes how the device communicates infrared data and other information to the host. For more information, see section 9.6.6 of the USB 2.0 specification.

The following code shows an IN endpoint descriptor structure:

```
typedef struct _endpointDescriptorEp1In {  
    byte bLength = sizeof(struct _endpointDescriptorEp1In);  
    byte bDescriptorType = TYPE_ENDPOINT_DESCRIPTOR;  
    byte bEndpointAddress = 0x81; // EP1 IN.  
    byte bmAttributes = 0x03; // Interrupt endpoint.  
    word wMaxPacketSize; // Hardware dependent.  
    byte bInterval; // Suggested value = 1 ms.  
}
```

Required String Descriptors

The device must return strings for byte **iManufacturer**, **iProduct**, and **iSerialNumber**. Strings are not required for **iConfiguration** or **iInterface**.

The serial-number string must be unique to the device. This means that two identical devices created by the same manufacturer will each have their own serial numbers. This is necessary because there may be two IR transceivers on a single host and the host uses the serial number to discriminate between the two devices.

Additionally, the Microsoft OS String Descriptor must be returned to the host when the host requests `string ID = 0xEE`. This is necessary to support the Microsoft Compatible Device Descriptor.

Microsoft Compatible Device Descriptor

The Microsoft® Compatible Device Descriptor is used by firmware to tell the host that the device is an IR emulator device. The host then loads the appropriate drivers. For more information about the Microsoft OS Descriptor and the Extended Compat ID Descriptor, see the Windows Hardware Developer Central page on the Microsoft Web site (<http://go.microsoft.com/fwlink/?LinkId=144040>).

To implement the descriptor, two firmware changes are necessary:

- The firmware must return the Microsoft OS String Descriptor when the host requests it. This is how the firmware identifies itself as supporting the Extended Compat ID Descriptor.
- The firmware must return the appropriate Extended Compat ID Descriptor when the host asks for it. This is how the firmware notifies the host that it is an IR Transceiver Version 2 or IR Receiver Version 3 emulation device.

Note The same Extended Compat ID Descriptor is returned for both IR Transceiver and IR Receiver devices. The difference between a transmitter/receiver device (transceiver) and a receive-only device (receiver) is defined by the device when it responds to the CMD_GETIRNUMPORTS request.

Both of these queries happen at device enumeration time.

Microsoft OS String Descriptor

When the host sends a request for string 0xEE to the firmware, the firmware must respond with a string containing the fields described in the following table.

Field	Length (bytes)	Value	Description
<i>bLength</i>	1	0x12	Length of the descriptor
<i>bDescriptorType</i>	1	0x03	String descriptor
<i>qwSignature</i>	14	'MSFT100'	Signature
<i>bMS_VendorCode</i>	1	Vendor Code	Vendor code to fetch other OS Feature Descriptors; equal to GET_CONFIG_DESCRIPTOR
<i>bPad</i>	1	0x00	Pad field

The *bLength*, *bDescriptorType*, *qwSignature*, and *bPad* values must be exactly as described here.

The *bMS_VendorCode* value is defined by the firmware writer and is used by the host to request the Extended Compat ID descriptor. This appears as *bRequest* in the example code in the Extended Compat ID Descriptor topic below.

Note To debug this, you may need to delete the osvc registry key on your computer. For more information, see <http://go.microsoft.com/fwlink/?LinkId=144042>.

If you are using a *bMS_VendorCode* of 1, your string will be the following exact value (in hex):

```
0x12 0x03 0x4D 0x00 0x53 0x00 0x46 0x00 0x54 0x00 0x31 0x00 0x30 0x00 0x30 0x00 0x01 0x00
```

Extended Compat ID Descriptor

The operating system will first issue a vendor specific request to the device with a *wLength* value set to 16. The purpose of this setting is to get the header section of the Extended Compat ID Descriptor. The following is an example trace to retrieve the header section:

```
bmRequestType = 1100000
```

bRequest = 0x01 - bMS_VendorCode
wValue = 0x0000 -
wIndex = 0x0004 - INDEX_CONFIG_DESCRIPTOR
wLength = 16 - Length of the request

After the header is retrieved (and if it matches the format of the header section), the operating system will issue a second request to read the entire Extended Compat ID Descriptor.

The Extended Compat ID Descriptor is an array of bytes containing the fields shown in the following table.

Field	Length (bytes)	Value	Description
dwLength	4	0x28 0x00 0x00 0x00	Length of the descriptor
bcdVersion	2	0x00 0x01	Version of the descriptor
wIndex	2	0x04 0x00	Fixed: INDEX_CONFIG_DESCRIPTOR
bCount	1	0x01	Count of device functions—must be 1
Reserved	7	0x00 0x00 0x00 0x00 0x00 0x00 0x00	Reserved
bFirstInterfaceNumber	1	0x00	First interface for this function
Reserved	1	0x01	Reserved
compatibleID	8	"USBCIR\0" -or- 0x55 0x53 0x42 0x43 0x49 0x52 0x00 0x00	Compatible ID, padded with zeros
subcompatibleID	8	"IR2CMPT\0" -or- 0x49 0x52 0x32 0x43 0x4D 0x50 0x54 0x00	Sub-compatible ID, padded with zeros
Reserved	6	0x00 0x00 0x00 0x00 0x00 0x00	Reserved

IAD/Extended Compat ID Descriptor Interaction

If you build an emulation device that is part of a composite USB device, you will need to modify your Extended Compat ID Descriptor to match the interfaces as defined in your IAD descriptor.

Below is an example of an Extended Compat ID descriptor for a composite device with three functions and three interfaces. The first function is CIR and uses interface 0. The second function is something else and uses interface 1. The third function is something else and uses interface 2.

For more information, see the Extended Compat ID Descriptor documentation.

```

BYTE extendedCompatIDDesc[] = {
    0x58, 0x00, 0x00, 0x00,           // dwLength
    0x00, 0x01,                       // bcdVersion
    0x04, 0x00,                       // INDEX_CONFIG_DESCRIPTOR
    0x03,                               // bCount - 3 interfaces.
    // First interface:
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // Reserved
    0x00,                               // Interface index
    0x01,                               // Reserved
    0x55, 0x53, 0x42, 0x43, 0x49, 0x52, 0x00, 0x00, // CompatibleID
    0x49, 0x52, 0x32, 0x43, 0x4D, 0x50, 0x54, 0x00, // SubcompatibleID
    // Second interface:
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // Reserved
    0x01,                               // Interface Index
    0x01,                               // Reserved
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // CompatibleID
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // SubcompatibleID
    0x00, 0x00, 0x00, 0x00, 0x00, // Reserved
    // Third interface:
    0x02,                               // interface interface index
    0x01,                               // Reserved
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // CompatibleID Null filled
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // SubcompatibleID Null filled
    0x00, 0x00, 0x00, 0x00, 0x00, // Reserved
};

```

Commands and Responses

The interface between the host computer and the IR emulation device is a command-response interface. The host sends a series of command bytes, with the first byte determining the type of command and the length. The IR emulator device responds with a response specific to the command received. Not all commands elicit a response.

Commands and responses have a consistent format. All communication packets begin with a lead byte. This byte is divided into a 3-bit port value and a 5-bit length value.

There are three valid port values, as shown in the following table.

Number	Name	Description
100	PORT_IR	Used for IR commands and responses.
111	PORT_SYS	Used for "system" commands and responses (non-IR device commands).
110	PORT_SER	A legacy port which used to be for a serial port but is now used only for loopback (flush) messages.

The 5-bit length field is overloaded as follows:

- If the length value is 11111, the following byte is a command (or response byte). The value of the command byte determines the length of the message. So, if the first byte of the packet is 0x9F (10011111), then this is a command byte (CMD_PORT_IR). The device then looks at the second byte. If it is 0x06 (CMD_SETIRCFS), then the device knows this packet is 4 bytes long because all CMD_SETIRCFS packets are 4 bytes long.
- If the length value is not 11111, then it specifies the number of bytes of port data that follow. So, if the first byte of the packet is 0x90 (10010000), the following 16 bytes of data are for the IR port. In that case, the total length of the packet, including the lead byte, is 17 bytes.

This same format is used in both directions—from host to device and from device to host. So IR commands and responses always begin with 0x9F.

If either the lead byte sent by the host or the command byte that follows the lead byte is incorrect, then the device sends a RSP_CMD_ILLEGAL response to the host. The device then waits for a CMD_RESUME command before resuming typical operation.

If the host receives an illegal lead byte or an illegal response byte, it can assume the device is in an error state and send a CMD_RESUME command to the device.

Multiple commands may come from the host in a single packet. The commands will always be sequential and never interleaved.

Commands That Set Device State

All commands that set device state are sent to the device over the EP1 OUT endpoint. Responses to these commands are returned to the host over the EP1 IN endpoint.

The following commands set device state:

- CMD_RESET – Reset the device.
- CMD_RESUME – Resume the device after error.
- CMD_SETIRCFS – Set IR carrier.
- CMD_SETIRTIMEOUT – Set IR time-out.
- CMD_SETIRTXPORTS – Set IR transmit ports.
- CMD_SETIRRXPORTEEN - Set IR receive ports.
- CMD_FLASHLED – Flash the LED

CMD_RESET – Reset the Device

Message ID: CMD_RESET

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_SYS	0xFF	System command
1	CMD_RESET	0xFE	Command ID - reset the device

Description

Resets the device. This command should restart the firmware in a default state.

There is no response to this command.

If the device has a bootloader, the device should enter the bootloader when the CMD_RESET command is received from the host.

CMD_RESUME – Resume the Device After Error

Message ID: CMD_RESUME

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_SYS	0xFF	System command
1	CMD_RESUME	0xAA	Command ID - resume the device after error

Description

This command clears any existing error state. This allows the host to resume sending data in a typical fashion. This is sent by the host after specific errors are returned by the device.

CMD_RESUME is sent by the host when one of the following occurs:

- An RSP_TX_TIMEOUT response is received from the device.
- An RSP_CMD_ILLEGAL response is received from the device.
- The device times out. This can happen, for instance, if the host is expecting a 6-byte response and it receives only 4 bytes from the device.

There is no response to this command.

CMD_SETIRCFS – Set IR Carrier

Message ID: CMD_SETIRCFS

Message length: 4 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_IR	0x9F	IR command
1	CMD_SETIRCFS	0x06	Command ID - set IR carrier frequency
2	CP	Number	Carrier prescalar
3	CC	Number	Carrier period

Description

This command sets the IR carrier frequency to use for transmitting IR. The frequency is sent as a carrier period and a carrier prescalar. The need to use a period value and a prescalar value is based on the PIC18F4320 timer architecture and maps directly to timer registers in this chip.

CP contains a prescalar value, and CC contains the carrier period in 1/10 μ sec steps. The actual carrier period will be:

$$\text{Period} = (2 ^ { (\text{CP} * 2) }) * (\text{CC} + 1) * 0.1 \mu\text{s}$$

where

$$\text{frequency} = 1 / \text{period}$$

Setting CP and CC to 0 will cause the device to use no carrier at all (that is, no light modulation, just constant on and off periods). The period count value CC can be any number from 0 to 255.

The following table describes CP and CC values for periods from 16 μ sec to 34 μ sec. This covers the required range of 30 to 60 KHz. Initial values of CP and CC should be 1 and 66 (37037 Hz), respectively.

Note This table calculates CP and CC based on periods in whole μ sec increments. Because the table starts with carrier period, which is an integer, and produces CP and CC, which are also integers, values must be rounded. However, the rounding errors introduced into this table are not significant enough to affect the accuracy of IR blasting.

Period (μ sec)	Carrier (Hz)	CP (μ sec)	CC (μ sec)
2	500000	0	19
3	333333	0	29
4	250000	0	39
5	200000	0	49
6	166666	0	59
7	142857	0	69
8	125000	0	79
9	111111	0	89
10	100000	0	99
11	90909	0	109
12	83333	0	119
13	76923	0	129
14	71428	0	139
15	66666	0	149

Period (μsec)	Carrier (Hz)	CP (μsec)	CC (μsec)
16	62500	0	159
17	58823	0	169
18	55555	0	179
19	52631	0	189
20	50000	0	199
21	47619	0	209
22	45454	0	219
23	43478	0	229
24	41666	0	239
25	40000	0	249
26	38461	1	64
27	37037	1	66
28	35714	1	69
29	34482	1	71
30	33333	1	74
31	32258	1	76
32	31250	1	79
33	30303	1	81
34	29411	1	84

DC Mode

If CP and CC are both zero, the transmitter operates in DC mode, and there is no carrier. This means that any RLC transmitted represents an unmodulated signal. There are no DC code sets in the current database.

CMD_SETIRTIMEOUT – Set IR Time-Out

Message ID: CMD_SETIRTIMEOUT

Message length: 4 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_IR	0x9F	IR command
1	CMD_SETIRTIMEOUT	0x0C	Command ID - set IR time-out
2	TOH	Number	High byte of time-out value
3	TOL	Number	Low byte of time-out value

Description

This command sets the IR time-out. This is the period of silence necessary before the firmware will determine that a signal has ended and will stop sending silence to the host.

TOH and TOL are the high and low bytes of the time-out period as a count of IR sample periods.

Response	Description
RSP_EQIRTIMEOUT	Successfully changed the IR time-out period.

No errors are returned to the host for this command.

CMD_SETIRTXPORTS – Set IR Transmit Ports

Message ID: CMD_SETIRTXPORTS

Message length: 3 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_IR	0x9F	IR command
1	CMD_SETIRTXPORTS	0x08	Command ID - set IR transmit ports
2	P	Number	Bitmask of ports to set

Description

This command sets the ports that will be used for IR transmissions. The bits of P represent the 8 IR ports, IR0 being the LSB, IR7 the MSB. A "1" bit indicates that the port will be used. This command is generally used before each IR transmission to specify which ports it will go to. The IR emulator device will not send a response to verify this command.

No response is returned to the host for this command.

No errors are returned to the host for this command.

CMD_SETIRRXPORTEEN - Set IR Receive Ports

Message ID: CMD_SETIRRXPORTEEN

Message length: 3 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_IR	0x9F	IR command
1	CMD_SETIRRXPORTEEN	0x14	Command ID - set IR receive ports
2	P	Number	Port number to receive on

Description

This command sets the IR ports that are enabled for reception. If P == 1, then the long-range receiver is used. If P == 2, then the wide-band receiver is used.

Response	Description
RSP_EQIRRXPORTEEN	Returns the receiver number used for IR reception.

No errors are returned to the host for this command.

CMD_FLASHLED – Flash the LED

EMVER_EMULATOR_V2 only

Message ID: CMD_FLASHLED

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_SYS	0xFF	System command
1	CMD_FLASHLED	0x23	Command ID – flash the LED

Description

When the device receives this command, it should flash its LED for two seconds.

Response	Description
RSP_FLASHLED	Successfully flashed the LED

No errors are returned to the host for this command.

Commands That Query Device State

All commands that query device state are sent to the device over the EP1 OUT endpoint. Responses to these commands are returned to the host over the EP1 IN endpoint.

The following commands query device state:

- CMD_GETIRCFS - Get IR carrier.
- CMD_GETIRTIMEOUT – Get IR time-out.
- CMD_GETIRTXPORTS – Get IR transmit ports.
- CMD_GETIRRXPORTEIN – Get IR receive ports.
- CMD_GETPORTSTATUS – Get transmit port status.
- CMD_GETIRNUMPORTS – Get number of ports.
- CMD_GETWAKESOURCE – Get wake source.
- CMD_GETEMVER – Get interface version used by emulator.
- CMD_GETDEVDETAILS – Get details about device capabilities.
- CMD_GETWAKESUPPORT – Get details about device wake support.
- CMD_GETWAKEVERSION – Get information about current wake pattern.

CMD_GETIRCFS – Get IR Carrier

Message ID: CMD_GETIRCFS

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_IR	0x9F	IR command
1	CMD_GETIRCFS	0x07	Command IR - get IR carrier frequency

Description

This command queries the Snowball for its current IR carrier setting.

Response	Description
RSP_EQIRCFS	Returns the carrier frequency.

CMD_GETIRTIMEOUT – Get IR Time-Out

Message ID: CMD_GETIRTIMEOUT

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_IR	0x9F	IR command
1	CMD_GETIRTIMEOUT	0x0d	Command IR - get IR time-out

Description

This command queries the device for its current IR time-out setting.

Response	Description
RSP_EQIRTIMEOUT	Returns the IR time-out period.

CMD_GETIRTXPORTS – Get IR Transmit Ports

Message ID: CMD_GETIRTXPORTS

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_IR	0x9F	IR command
1	CMD_GETIRTXPORTS	0x13	Command ID - get IR transmit ports

Description

This command queries the device to get the bitmask of currently selected IR transmit ports.

Response	Description
RSP_EQIRTXPORTS	Returns the bitmask of selected transmit ports.

CMD_GETIRRXPORTEN – Get IR Receive Ports

Message ID: CMD_GETIRRXPORTEN

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_IR	0x9F	IR command

Offset	Name	Value	Description
1	CMD_GETIRRXPORTEN	0x15	Command ID - get IR receive ports

Description

This command returns which IR ports are being used for reception.

Response	Description
RSP_EQIRRXPORTEN	Returns the receiver number used for IR reception.

CMD_GETPORTSTATUS – Get Transmit Port Status

Message ID: CMD_GETPORTSTATUS

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_SYS	0xFF	System command
1	CMD_GETPORTSTATUS	0x11	Command ID - get transmit port status
2	P	Number	Port number to get status for

Description

Sends a command to have the device identify what is connected to the transmit port (for example, nothing or an emitter).

Response	Description
RSP_GETPORTSTATUS	Respond with transmit port status.

No errors are returned to the host for this command.

CMD_GETIRNUMPORTS – Get Number of Ports

Message ID: CMD_GETIRNUMPORTS

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_IR	0x9F	IR command

Offset	Name	Value	Description
1	CMD_GETIRNUMPORTS	0x16	Command ID - get the number of ports

Description

This command queries the device for the number transmit and receive ports that it has.

Response	Description
RSP_EQIRNUMPORTS	Returns the number of ports.

CMD_GETWAKESOURCE – Get Wake Source

Message ID: CMD_GETWAKESOURCE

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_SYS	0xFF	System command
1	CMD_GETWAKESOURCE	0x17	Command ID - get wake source

Description

Requests the source of a device wake. For example, the wake might be due to the Sleep button being pressed.

This command lets host know whether the user is present. If the user presses the Sleep button and this causes the USB device to wake the system, then the firmware needs to internally store a Boolean value that specifies the Sleep button as the source. When the CMD_GETWAKESOURCE command is received from the host, the firmware returns the value of that Boolean and resets it to false.

Response	Description
RSP_GETWAKESOURCE	Returns true if the Sleep button woke the system.

CMD_GETEMVER – Get the Interface Version Used by the Emulator

Message ID: CMD_GETEMVER

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_SYS	0xFF	System command
1	CMD_GETEMVER	0x22	Command ID - Get interface version used by emulator

Description

This command is sent by the host to query which version of the emulator interface this device is using. There are two ways to respond to this command:

- You can error out and return RSP_CMD_ILLEGAL. A properly-implemented EMVER_EMULATOR_V1 device should do this. When the host receives the RSP_CMD_ILLEGAL response, it will assume that the device is EMVER_EMULATOR_V1 and treat it accordingly.
- You can respond with an RSP_EQEMVER response indicating the version of the emulator interface that your device is using.

Response	Description
RSP_EQEMVER	Returns the interface version used by the emulator
RSP_CMD_ILLEGAL	Request is not supported because the emulator uses EMVER_EMULATOR_V1

CMD_GETDEVDETAILS – Get Details about Device Capabilities

EMVER_EMULATOR_V2 only

Message ID: CMD_GETDEVDETAILS

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_SYS	0xFF	System command
1	CMD_GETDEVDETAILS	0x21	Command ID - Get details about device capabilities

Description

This command is sent by the host when the host wants to query the device about its capabilities. Details about the capabilities that a device can return are in the RSP_EQDEVDETAILS section.

Response	Description
RSP_EQDEVDETAILS	Returns details about device capabilities

CMD_GETWAKESUPPORT – Get Details about Device Wake Support

EMVER_EMULATOR_V2 only

Message ID: CMD_GETWAKESUPPORT

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_SYS	0xFF	System command
1	CMD_GETWAKESUPPORT	0x20	Command ID - get details about device wake support

Description

This command is sent by the host when the host wants to query the device about its wake capabilities. Details about the wake capabilities that a device can return are in the RSP_EQWAKESUPPORT section.

Response	Description
RSP_EQWAKESUPPORT	Returns details about device wake capabilities

CMD_GETWAKEVERSION – Get Information about the Current Wake Pattern

EMVER_EMULATOR_V2 only

Message ID: CMD_GETWAKEVERSION

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_SYS	0xFF	System command
1	CMD_GETWAKEVERSION	0x18	Command ID - get information about the current wake pattern

Description

This command is sent by the host when the host wants to query the device about its current wake version. Details about the returned values can be found in the RSP_EQWAKEVERSION section.

Response	Description
RSP_EQWAKEVERSION	Respond with details about current wake pattern

Miscellaneous Commands

The miscellaneous commands are sent to the device over the EP1 OUT endpoint. Responses to these commands are returned to the host over the EP1 IN endpoint.

The following miscellaneous commands are available:

- CMD_NOP – No operation.
- Flush

CMD_NOP – No Operation

Message ID: CMD_NOP

Message length: 2 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_PORT_SYS	0xFF	System command
1	CMD_NOP	0xFF	Command ID - no operation

Description

Does nothing. This operation is a No-Op. When the firmware receives this command, it should ignore it and immediately re-enter its receive loop to receive the next command.

There is no response to this command.

Flush

Message ID: Flush

Message length: 1-31 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	PORT_SER + Length	0xc0 - 0xDF	PORT_SER (0x6) is in the high 3 bits. The length of the data is in the low 5 bits
1-31	Data	Numbers	Data to send. May be empty.

Description

The Flush command is used to synchronize the host with the device. The device should loop back the same data to the host after the device is done processing all data. So, for instance, if the device has an outgoing FIFO with IR data to transmit, and then it receives the flush command, it should wait until the FIFO is empty before responding to the Flush command.

The first byte of this command has PORT_SER (0x6) in the upper three bits and the length of the data in the lower five bits. So, if the host is sending zero bytes (a valid case), the command would just be one byte long, as shown in the following table.

Offset	Name	Value	Description
0	PORT_SER + Length	0xC0	Upper three bits = 0x6. Lower five bits = 0x00. Zero bytes of data.

If the host is sending three bytes, the command would be four bytes long, as shown in this table.

Offset	Name	Value	Description
0	PORT_SER + Length	0xC3	Upper three bits = 0x6. Lower five bits = 0x03. Three bytes of data.
1	Data	Number	First byte of data
2	Data	Number	Second byte of data
3	Data	Number	Third byte of data

Response

The data, excluding the prefix, will be sent back to the host upon completion of the command.

Example

In a typical example, the host will send the sequence “0xC1 0xC5 0xC0”.

0xC1 means one byte of data following.

0xC5 is the one byte of data that you need to echo back.

0xC0 means zero bytes of data following; this is basically an EOF marker.

In response to this, the device will echo the data back. In this example, the host would return a single byte: 0xC5.

Errors

No specific errors are returned.

Responses to Commands: Non-Error Cases

All responses in this section are returned to the host over the EP1 IN endpoint.

The following responses are available when there is no error:

- RSP_EQIRCFS – Respond with IR carrier for transmission.
- RSP_EQIRTIMEOUT – Respond with current IR time-out.

- RSP_GETWAKESOURCE – Respond with wake source.
- RSP_EQIRTXPORTS – Respond with current transmit port mask.
- RSP_EQIRRXPORTEM – Respond with current IR receive port mask.
- RSP_GETPORTSTATUS – Respond with transmit port status.
- RSP_EQIRRXCFCNT – Respond with received carrier count information.
- RSP_EQIRNUMPORTS – Respond with number of ports.
- RSP_EQWAKESUPPORT – Respond with details about device wake capabilities.
- RSP_EQWAKEVERSION – Respond with details about the current wake pattern.
- RSP_EQDEVDETAILS – Respond with details about device capabilities.
- RSP_EQEMVER – Respond with the interface version used by the emulator.

RSP_EQIRCFS – Respond with IR Carrier for Transmission

Message ID: RSP_EQIRCFS

Message length: 4 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_IR	0x9F	IR response
1	RSP_EQIRCFS	0x06	Response ID – respond with IR carrier for transmission
2	CP	Number	Carrier prescalar
3	CC	Number	Carrier period

Description

This is the response used to return the carrier frequency to the host. Specifically, this sends the carrier frequency used to transmit IR. The RSP_EQIRRXCFCNT response is used to send carrier information to the host for received IR.

For a description of the prescalar and period values, see the topic CMD_SETIRCFS earlier in this document.

RSP_EQIRTIMEOUT – Respond with Current IR Time-Out

Message ID: RSP_EQIRTIMEOUT

Message length: 4 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_IR	0x9F	IR response

Offset	Name	Value	Description
1	RSP_EQIRTIMEOUT	0x0C	Response ID – respond with current IR time-out
2	TOH	Number	High byte of time-out value
3	TOL	Number	Low byte of time-out value

Description

This is the response used to return the IR time-out period to the host. For definitions of TOH and TOL, see CMD_SETIRTIMEOUT earlier in this document.

RSP_GETWAKESOURCE – Respond with Wake Source

Message ID: RSP_GETWAKESOURCE

Message length: 3 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_SYS	0xFF	System response
1	RSP_GETWAKESOURCE	0x17	Response ID – respond with wake source
2	WAKE	Number	1 if the device was responsible for waking the system; 0 otherwise

Description

This is the response used to indicate whether the device woke the system.

RSP_EQIRTXPORTS – Respond with Current Transmit Port Mask

Message ID: RSP_EQIRTXPORTS

Message length: 3 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_IR	0x9F	IR response
1	RSP_EQIRTXPORTS	0x08	Response ID – respond with current transmit port mask
2	P	Number	Bitmask with current transmit port mask

Description

This is the response used to return the current transmit port mask to the host.

RSP_EQIRRXPORTEN – Respond with Current IR Receive Port Mask

Message ID: RSP_EQIRRXPORTEN

Message length: 3 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_IR	0x9F	IR response
1	RSP_EQIRRXPORT EN	0x14	Response ID – respond with current IR receive port mask
2	P	Number	Current port used for reception

Description

This is the response used to return the current receiver to the host. For a definition of P, see CMD_SETIRRXPORTEN earlier in this document.

RSP_GETPORTSTATUS – Respond with Transmit Port Status

Message ID: RSP_EQPORTSTATUS

Message length: 7 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_SYS	0xFF	System Response
1	RSP_GETPORTSTA TUS	0x11	Response ID – respond with transmit port status
2	P	Number	Port to return status for
3	VRH	Number	Volts-at-rest high
4	VRL	Number	Volts-at-rest low
5	VDH	Number	Volts-when-driven high
6	VDL	Number	Volts-when-driven low

Description

This is the response used to indicate whether something is plugged into a specific transmit port. In the Microsoft-produced IR device, this was done by measuring voltage drop across the port. By measuring volts-at-rest and volts-when-driven, the software could differentiate between IR emitters and S-Link devices. For emulation devices, IR emitters are the only option. As a result, physically measuring the presence of a plug in the jack is sufficient.

To simplify matters, the following values should be returned.

Status	VRH	VRL	VDH	VDL
Emitter connected	0x00	0x00	0x00	0x00
Emitter not connected	0x00	0x00	0xFF	0x00

RSP_EQIRRXFCNT – Respond with Received Carrier Count Information

Message ID: RSP_EQIRRXFCNT

Message length: 4 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_IR	0x9F	IR response
1	RSP_EQIRRXFCNT	0x15	Response ID – respond with received carrier count information
2	CH	Number	Carrier count high byte
3	CL	Number	Carrier count low byte

Description

After a time-out of reception on the learning receiver, this response is sent to tell the host the carrier frequency of the previous sample. The CH and CL values form a 16-bit value that specifies the count of cycles of the carrier. Carrier count can also be thought of as the number of leading edges in the previous sample.

This is used by the host to calculate carrier frequency as follows:

```
int lastCarrierCount = ch*256+cl;
double carrier = ((double)lastCarrierCount) / irPacketOnDuration;
```

In this example, *lastCarrierCount* is computed by the host based on the values returned in this response. *irPacketOnDuration* value is the total amount of time that the envelope for the signal was high. This value is computed by the host and is implied by the shape of the RLC envelope returned from the device since the last RSP_EQIRRXFCNT response.

This response is unsolicited. It is returned by the receiver when IR arrives but is never explicitly requested.

The carrier count is a count of pulses that occurred since the last time-out.

RSP_EQIRNUMPORTS – Respond with Number of Ports

Message ID: RSP_EQIRNUMPORTS

Message length: 3 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_IR	0x9F	IR response
1	RSP_EQIRNUMPORTS	0x16	Response ID – respond with number of ports
2	TXC	Number	Count of transmit ports on device
3	RXC	Number	Count of receive ports on device

Description

This is the response that tells the host how many transmit and receive ports your device has.

The numbers returned in this response are fixed. They should not change based on the state of the device.

RSP_EQWAKESUPPORT – Respond with Details about Device Wake Capabilities

EMVER_EMULATOR_V2 ONLY

Message ID: RSP_EQWAKESUPPORT

Message length: 3 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_SYS	0xFF	System response
1	RSP_EQWAKESUPPORT	0x20	Respond with details about device wake capabilities
2	WAKECAPS	Number	Byte with bitmask of WakeSupportBits values

Description

This is the response in which the device tells the host how it supports wake, including the programmability of the device, the protocols that the device supports, and the method of programming.

The following bit values can be OR'ed together and returned in the WAKECAPS field.

Name	Value	Description
------	-------	-------------

Name	Value	Description
WAKE_SUPPORTED	0x01	The device supports wake from remote
WAKE_PROGRAMMABLE	0x02	The device wake algorithm is programmable
WAKE_MULTIPLE	0x04	The device supports wake from all required protocol/key combinations without programming.
WAKE_RC6	0x08	The device supports wake on RC6 key
WAKE_QP	0x10	The device supports wake on Quatro Pulse key
WAKE_DONTCARE	0x20	The device theoretically supports wake on all protocols
WAKE_VOLATILE_PATTERN	0x40	The device wake pattern is stored in volatile memory and must be refreshed on every device init

RSP_EQWAKEVERSION – Respond with Details about the Current Wake Pattern

EMVER_EMULATOR_V2 ONLY
 Message ID: RSP_EQWAKEVERSION
 Message length: 6 bytes
 Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_SYS	0xFF	System response
1	RSP_EQWAKEVERSION	0x18	Respond with details about the current wake pattern
2	Protocol	Number	The protocol for the current wake pattern
3	Payload	Number	The key code for the current wake pattern
4	Address	Number	Address for the current wake pattern
5	Version	Number	Version of the firmware for the current wake pattern

Description

This is the response in which the device tells the host what wake pattern it is currently listening for. This response could be based on factory programming or it could be based on run-time programming of the device by the host using the CMD_BOOT_SETWAKEPATTERN or CMD_BOOT_WRITEBLOCK bootloader commands.

The following values are valid for the Protocol field:

Name	Value	Description
V2_WAKE_PROTOCOL_RC6	0x01	Wake key uses the RC6 protocol.
V2_WAKE_PROTOCOL_QP	0x02	Wake key uses the Quatro Pulse protocol.

The following values are valid for the Payload field:

Name	Value	Description
WAKE_KEY_POWER_TOGGLE	0x0c	Button code for the Sleep toggle button
WAKE_KEY_DISCRETE_ON	0x29	Button code for the discrete on button

The Address field should contain the address field in the wake pattern that it is listening for. Valid values are 0-7 for RC6 and 0-15 for Quatro Pulse.

The Version field is only used when the wake pattern was programmed using the CMD_BOOT_WRITEBLOCK method. When using this method, the Version number is extracted from the firmware that was sent by the host. When not using this method, the Version number should be returned as zero.

RSP_EQDEVDETAILS – Respond with Details about Device Capabilities

EMVER_EMULATOR_V2 ONLY

Message ID: RSP_EQDEVDETAILS

Message length: 3 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_SYS	0xFF	System response
1	RSP_EQDEVDETAILS	0x21	Respond with details about device capabilities
2	DEVDETAILS	Number	Byte with bitmask of DeviceDetailsBits values

Description

This is the response that the device uses to communicate its capabilities to the host.

The following bit values can be OR'ed together and returned in the WAKECAPS field.

Name	Value	Description
DEVDETAILS_TIEDTOTUNER	0x0 1	The device is tied to a tuner (for instance, the device is part of USB tuner/IR receiver combination device)
DEVDETAILS_LEARNINGONLY	0x0 2	The device supports IR learning, but not long-range IR reception
DEVDETAILS_NARROWBPF	0x0 4	Long-range receiver on the device has narrow Band Pass Filter (BPF). Parse-and-match remote identification is not possible with the long-range receiver on this device.
DEVDETAILS_NOINPUT	0x0 8	The device does not support IR input. Long range receiver, if any, is only used for parse-and-match remote identification.
DEVDETAILS_CANFLASH	0x1 0	The device supports the CMD_FLASHLED command
DEVDETAILS_HASBOOTLOADER	0x2 0	The device has bootloader mode.

RSP_EQEMVER – Respond with the Interface Version used by the Emulator

EMVER_EMULATOR_V2 ONLY

Message ID: RSP_EQEMVER

Message length: 3 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_SYS	0xFF	System response
1	RSP_EQEMVER	0x22	Respond with the interface version used by the emulator
2	EMVER	Number	Emulator version number

Description

This response allows the device to tell the host which version of the emulator interface the device implements. If the device responds with EMVER_EMULATOR_V1, or if the device responds to CMD_GETEMVER with RSP_CMD_ILLEGAL, the host assumes that the device uses the version

1 emulator interface. In that case, the host does not send any of the EMVER_EMULATOR_V2-only commands to the device.

The following values are accepted in the EMVER field:

Name	Value	Description
EMVER_EMULATOR_V1	0x01	The device is using the old (Version 1) emulator interface
EMVER_EMULATOR_V2	0x02	The device is using the newer (Version 2) emulator interface.

RSP_FLASHLED – Respond Indicating that the Device Successfully Flashed the LED

EMVER_EMULATOR_V2 ONLY

Message ID: RSP_FLASHLED

Message length: 2 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_SYS	0xFF	System response
1	RSP_FLASHLED	0x23	Respond indicating that device successfully flashed the LED

Description

The device returns this response to the host after successfully flashing the LED in response to a CMD_FLASHLED command.

Responses to Commands: Error Cases

All responses in this section are returned to the host over the EP1 IN endpoint.

The following responses are available when there is an error:

- RSP_CMD_ILLEGAL – Illegal command.
- RSP_TX_TIMEOUT – Error for transmit time-out.

RSP_CMD_ILLEGAL – Illegal Command

Message ID: RSP_CMD_ILLEGAL

Message length: 2 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_SYS or RSP_PORT_IR	0xFF or 0x9F	System or IR response
1	RSP_CMD_ILLEGAL	0xFE	Response ID – illegal command

Description

This response is sent when the command received does not exist for the given port.

This is a critical error. A CMD_RESUME command from the host is required to recover from this error.

RSP_TX_TIMEOUT – Error for Transmit Time-Out

Message ID: RSP_TX_TIMEOUT

Message length: 2 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_PORT_IR	0x9F	IR response
1	RSP_TX_TIMEOUT	0x81	Response ID – error for transmit time-out

Description

This response is sent when the device runs out of data to send to a port before a Data End command is received. The device must receive all of the data for a given transmission in a timely fashion to send the data out as one contiguous signal.

This is a critical error. A CMD_RESUME command from the host is required to recover from this error.

Illegal Command Handling

To ensure backwards and forwards compatibility, all emulator devices must properly respond to illegal commands.

The "Commands and Responses" section briefly describes this behavior. This section contains more detail.

If a command has an illegal lead byte or an illegal following byte, that command is considered illegal. When a device receives an illegal command, it should return RSP_CMD_ILLEGAL and wait for a CMD_RESUME before continuing normal operation.

The expected behavior is outlined in the following example:

```
// main firmware loop for normal operating mode
void mainLoop() {
    while (true) {
```

Remote Control and Receiver-Transceiver Specifications and Requirements
for Windows Media Center in Windows Operating Systems

```
// get next byte from host
byte b = getNextByteFromHost();
if ( errorState ) {
    // if we're in an error state, see if we can exit the error state
    // don't actually remove the next byte from the incoming buffer yet.
    if ( peekNextByteFromHost() == CMD_RESUME ) errorState = false;
}
// only do processing if we're not in error state
if ( !errorState )
{
    // switch on command byte
    switch (b) {
        case CMD_PORT_IR: handleIrCommand(); break;
        case CMD_PORT_SYS: handleSysCommand(); break;
        case CMD_PORT_SER: handleFlush(); break;
        default:
            if ( (b & 0xe0) == 0x90 ) {
                // If the high 3 bits are 100, this is an IR packet.
                // The length of the IR packet is in the low 5 bits
                blastIr(b & 0x1f);
            } else {
                // otherwise, it's an illegal command
                illegalCommand();
            }

            break();
        }
    }
}
}
}
}
void handleIrCommand() {
    // get the next byte from the host.
    byte b = getNextByteFromHost();
    switch (b) {
        case CMD_SETIRCFs: HandleSetIrcfs(); break;
        case CMD_GETIRCFs: HandleGetIrcfs(); break;
        // add code here to handle all the other PORT_IR commands.
        default:
            // any other commands are errors
            illegalCommand();
    }
}
}
void illegalCommand() {
    // report the illegal command to the host
    sendRspCmdIllegalToHost();
    // set the device into an error state
    errorState = true;
}
```

This behavior will be tested as part of our test suites. The expected interaction is as follows:

1. Test sends an illegal command to the device.
2. Test validates the RSP_CMD_ILLEGAL response.
3. Test sends a flush command to the device.
4. Test validates that the device does not respond.
5. Test sends a CMD_RESUME command to the device.
6. Test sends a flush command to the device.

7. Test validates that the device does respond.

Bootloader Implementation

To support wake programming, Version 2 emulator devices must have a bootloader. While in bootloader mode, all normal device operation is suspended. The bootloader is entered when a `CMD_RESET` command is received and exited when a `CMD_BOOT_EXIT` is received. The set of commands that work in the bootloader are separate and distinct from the set of commands that work in normal operating mode. There is no overlap between bootloader commands and normal operation commands.

Enabling Bootloader Functionality

To enable the bootloader, the device must set the `DEVDETAILS_HASBOOTLOADER` bit in its `RSP_EQDEVDETAILS` response. If the device does not set this bit, the host assumes that the device does not have a bootloader and does not attempt to enter the bootloader.

The purpose of the bootloader is to support wake programming. All devices must set the `WAKE_SUPPORTED` and `WAKE_PROGRAMMABLE` bits to communicate to the host that they support wake programming. Supporting for wake programming is required.

Entering the Bootloader

The device should enter the bootloader when it receives the `CMD_RESET` command from the host.

Exiting the Bootloader

The device should exit the bootloader when it receives the `CMD_BOOT_EXIT` command. After exiting the bootloader, the device should re-initialize itself. It is possible that the host will issue a `CMD_RESET` command followed immediately by a `CMD_BOOT_EXIT` command to reset the device.

Wake Programming: `CMD_BOOT_SETWAKEPATTERN`

All devices that have a bootloader and support wake programming will receive `CMD_BOOT_SETWAKEPATTERN` commands from the host. This command sends the protocol, the payload, and the remote control address to the device. The device should use this information as necessary to program its wake algorithm.

When designing your device, you have the following options for wake programming:

- Single pattern. In this case, the device does not set `WAKE_MULTIPLE`. When the host sets the wake pattern using the `CMD_BOOT_SETWAKEPATTERN` command, the device wakes on that protocol, payload, and address only. This option is for devices that need to conserve power in a low power state.

For example, a single-pattern device that receives `CMD_BOOT_SETWAKEPATTERN` with `protocol=RC6`, `Payload=0x0c`, and `Address=0x02` wakes on the RC6 Sleep toggle button from a remote with address 2, but does not wake on any other Sleep signature.

- Multiple pattern. In this case, the device sets `WAKE_MULTIPLE`. When the host sets the wake pattern using the `CMD_BOOT_SETWAKEPATTERN`, the device wakes on all Sleep buttons with that address.

For example, a multiple-pattern device that receives `CMD_BOOT_SETWAKEPATTERN` with `protocol=RC6`, `Payload=0x0c`, and `Address=0x02` wakes on four different keys: RC6 Sleep toggle, RC6 discrete on, Quatro Pulse Sleep toggle, and Quatro Pulse discrete on. However, the device only wakes on these keys if they have the address set to 0x02. Remotes with other addresses must not wake the host.

In both cases, the device must examine the values passed with the `CMD_BOOT_SETWAKEPATTERN` command and adjust its behavior based on those values.

Wake Programming: `CMD_BOOT_WRITEBLOCK`

If your firmware space and clock are limited, you can use custom wake firmware for each wake pattern. This firmware can be installed in the registry on the host, and the device driver can use the `CMD_BOOT_WRITEBLOCK` command to send the appropriate wake firmware from the host to the device. The format of the registry data is entirely opaque to the driver. It sends a specific block of data to the device when a given wake key is needed by the user.

With two protocols, two possible wake keys, and eight addresses, a given device will need 32 ($2 \times 2 \times 8$) separate firmware blocks in the registry.

No mechanism is provided to install the firmware into the registry on the host device. If you choose to use this option, you must provide an installation mechanism to your customers.

If no registry value exists with the correct key, the host will skip the `CMD_BOOT_WRITEBLOCK` write sequence. In this case, the device will probably fall back to default behavior and the user may be left with a Sleep button on their remote control that does not wake the PC.

Firmware Write Sequence

This section describes the entire sequence from beginning to end for programming a new wake pattern in an emulator device.

User Presses Sleep (formally Power) Button

The host driver tracks the type of Sleep button the user has (protocol, button code, and address). When the user presses the Sleep button on the remote, the driver inspects the payload and notes the type of Sleep button the user has. For example, the user has an RC6 remote with discrete Sleep buttons set to address 3. When the user presses the “discrete off” button, the host driver notes that it must program the device to wake when it receives the RC6 “discrete on” button with address 3. The driver doesn’t do anything with this information until later.

Host Enters Low-Power State

Because the user has pressed the Sleep button, the system enters a low-power state. The device watches for whichever wake pattern it happens to be watching for. This wake pattern may or may not be correct. The pattern may be the default wake pattern as chosen by the hardware manufacturer.

Host Wakes, PNP Event Received by Host

The user wakes the host system. The device driver for the emulation device receives a PNP event indicating that the device is once again available.

Host Enters Bootloader

The host uses `CMD_GETWAKEVERSION` to request the wake pattern that the device is currently using. If the device is using a wake pattern that does not match the pattern that the host noted earlier, the host must reprogram the device with a different pattern. At this point, the host sends a `CMD_RESET` to enter the device's bootloader. The host sends a `CMD_BOOT_GETVERSION` to make sure that the device is in the bootloader.

Host Sends `CMD_BOOT_SETWAKEPATTERN`

Once in the bootloader, the host sends a `CMD_BOOT_SETWAKEPATTERN` command to the device.

Host Sends `CMD_BOOT_WRITEBLOCK`

The host scans the registry to determine whether a firmware block is available for this device and this specific wake pattern.

If a firmware block is not in the registry, the host continues to the next step.

If there is a firmware block in the registry, the host begins the `CMD_BOOT_WRITEBLOCK` sequence:

- The host "authorizes writing" using a `CMD_BOOT_WRITEAUTH` command. This sequence of bytes is used to protect the firmware and to prevent random jumps from overwriting flash memory. This process is explained below in more detail in the `CMD_BOOT_WRITEAUTH` section.
- The host uses the `CMD_BOOT_WRITEBLOCK` to send multiple blocks of firmware to the device. The registry data with the firmware contains a size parameter that instructs the host how to break up the firmware into blocks. The host sends one `CMD_BOOT_WRITEBLOCK` command for each block of firmware.

Host Exits Bootloader

The host sends a `CMD_BOOT_EXIT` command to exit the bootloader. The device resets itself and the new wake firmware is applied.

User Presses Sleep Button Again

When the user presses the Sleep button a second time, the host determines from the Sleep button that the device's wake pattern is correct. The host enters a low-sleep state. The next time the user presses the Wake button on the remote, the device will recognize it and signal the host to wake.

Registry Format For Firmware Blocks

Registry Key Location

Wake firmware is stored in the registry at the following location:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\usbcir\PowerKey`

Registry Value Name

The firmware is stored in a REG_BINARY value using the following naming convention:

PowerKey-VIDPID-Protocol-KeyCode-Address

- *VIDPID* is your device's VID and PID concatenated as a hexadecimal value.
- *Protocol* is the protocol, in text. Valid values are RC6 or QP.
- *KeyCode* is the button code for the Wake key. Valid values are 0c or 29.
- *Address* is the remote address for the Wake key.

Registry Value Format

The registry value is a REG_BINARY value of arbitrary length.

The first eight bytes have specific meaning. The rest of the data is the firmware to download.

Offset	Name	Description
0	POWER_KEY_PROTOCOL	Protocol for this Sleep key. One of the V2_WAKE_PROTOCOL values.
1	POWER_KEY_PAYLOAD	Payload for this Sleep key. Either WAKE_KEY_POWER_TOGGLE or WAKE_KEY_DISCRETE_ON.
2	POWER_KEY_ADDRESS	Address for this Sleep key.
3	POWER_KEY_VERSION	Version number of this Sleep key firmware*.
4	POWER_KEY_BLOCKSIZE1	Block size MSB.
5	POWER_KEY_BLOCKSIZE2	Block size byte #2.
6	POWER_KEY_BLOCKSIZE3	Block size byte #3.
7	POWER_KEY_BLOCKSIZE4	Block size LSB.
8+	POWER_KEY_DATA	Sleep key data. Must be an even number of blocks defined by the BLOCKSIZE parameter.

* The version number can be used to update the Sleep key firmware. If you need to fix wake firmware in the field, you can increment the version number in the registry, which forces the host driver to download the new firmware to the device.

The first 4 bytes of this response directly corresponds to the return value from CMD_GETWAKEVERSION. The host performs a 32-bit comparison operation between the CMD_GETWAKEVERSION response and the first four bytes in this registry value. If the values are different, the host initiates a wake firmware download.

Wake Firmware Registry Example

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\usbcir\PowerKey]
```

```
"PowerKey-045e00fe-RC6-0c-03"=hex:01,0C,03,01, \
```

```
00,00,00,0C, \
```

```
1F,00,12,34,56,78,91,23,45,67,00,A9, \
```

```
1F,08,12,34,56,78,91,23,45,67,00,A9, \
```

```
1F,10,12,34,56,78,91,23,45,67,00,A9
```

In this example, the VID is 0x045e, the PID is 0x00fe. This is the firmware for RC6, Sleep toggle (button code = 0x0c), address = 3. Each download block is 12 bytes long. There are three 12-byte download blocks in the firmware. The POWER_KEY_DATA information must be an even number of blocks or the host will ignore the registry entry and does not attempt to program the device.

The block format in this example is not the required format. You can choose any block format and any length you want.

Bootloader Example Pseudo-Code

This example assumes the same registry format used by the Wake Firmware Registry Example section.

```
void handleSysCommand() {
    byte b = getNextByteFromHost();
    switch (b) {
        case CMD_GETPORTSTATUS: handleGetPortStaus(); break;
        case CMD_GETWAKESOURCE: handleGetWakeSource(); break;
        case CMD_RESET: handleReset(); break;
        // add code here to handle all the other PORT_SYS commands.
        default:
            // any other commands are errors
            illegalCommand();
    }
}

void handleReset() {
    // Reset write authorization before entering the bootloader
    writeAuth = 0;
    // on reset, run the bootloader
    runBootLoaderLoop();
    // Reset write authorization after exiting the bootloader
    writeAuth = 0;
    // then reset the device once we exit the bootloader
    resetDevice();
}

void runBootLoaderLoop() {
    while (true) {
        byte b = getNextByteFromHost();
        switch (b) {
            case CMD_BOOT_EXIT: return;
            case CMD_BOOT_GETVERSION: handleBootGetVersion(); break;
            case CMD_BOOT_WRITEAUTH:handleBootWriteAuth(); break;
            case CMD_BOOT_WRITEBLOCK: handleBootWriteBlock(); break;
            case CMD_BOOT_SETWAKEPATTERN:handleBootSetWakePattern(); break;
            default: break;
        }
    }
}

void handleBootGetVersion() {
    // send the wake version to the host
    sendToHost(RSP_BOOT_VERSION,1);
}

void handleBootWriteAuth() {
    // get the write authorization from the host and save it for later.
    writeAuth = (getNextByteFromSource() << 24) +
        (getNextByteFromSource() << 16) +
        (getNextByteFromSource() << 8) +
        getNextByteFromSource();
}

void handleBootWriteBlock() {
    byte checksum = 0;
    byte data[8];
    // In our scheme, first 16 bytes are the destination address.
    uint address = (getNextByteFromSource() << 16) +
        getNextByteFromSource();
    // Next 8 bytes are firmware
```

```
for (int i = 0; i < 8; i++) {
    // save the bytes
    data[i] = getNextByteFromSource();
    // compute a simple checksum
    checksum = checksum << 1;
    checksum ^= data[i];
}
// One byte of padding
getNextByteFromSource();
// Last byte is expected checksum
byte expectedChecksum = getNextByteFromSource();
// Make sure the checksum is correct
if ( checksum != expectedChecksum ) {
    sendToHost(RSP_BOOT_BADSERIALCHECKSUM);
    return;
}
// Make sure we've been authorized to write.
// (protects against random jumps)
if ( writeAuth != EMULATOR_WRITEAUTHSEQ ) {
    sendToHost(RSP_BOOT_BADWRITEAUTH);
    return;
}
// Finally, program the hardware
writeBlock(address,data);
}
```

It is recommended that you put writeAuth checks throughout your firmware writing code, which helps to prevent a random jump from writing random values to your flash memory.

Wake Programming Example Sequence

This section contains an example programming sequence to further explain the communication that initiates wake programming. One example is for a device that relies solely on CMD_BOOT_SETWAKEPATTERN for wake programming, and does not have any wake firmware in the registry. The second example is for a device that relies on CMD_BOOT_WRITEBLOCK with wake firmware stored in the host registry.

Example #1 – No Wake Firmware in Registry

1. Host Sends: 0xff 0xfe (CMD_PORT_SYS, CMD_RESET)
2. (Device enters bootloader)
3. Host Sends: 0xf5 (CMD_BOOT_GETVERSION)
4. Device Responds: 0x04, 0x01 (RSP_BOOT_VERSION,1)
5. Host Sends: 0xef, 0x01,0x0c,0x03 (CMD_BOOT_SETWAKEPATTERN, RC6, Sleep Toggle, Address 3)
6. Device Responds: 0xef (RSP_BOOT_SETWAKEPATTERN)
7. Host Sends: 0xf4 (CMD_BOOT_EXIT)
8. (Device resets itself)

Example #2 – Wake Firmware in Registry

This example assumes the same registry format used by the Wake Firmware Registry Example section.

1. Host Sends: 0xff 0xfe (CMD_PORT_SYS, CMD_RESET)
2. (Device enters bootloader)
3. Host Sends: 0xf5 (CMD_BOOT_GETVERSION)
4. Device Responds: 0x04, 0x01 (RSP_BOOT_VERSION,1)
5. Host Sends: 0xef, 0x01,0x0c,0x03 (CMD_BOOT_SETWAKEPATTERN, RC6, (Sleep Toggle, Address 3)
6. Device Responds: 0xef (RSP_BOOT_SETWAKEPATTERN)
7. Host Sends: 0xf6,0x23,0xca,0x67,0xd0 (CMD_BOOT_WRITEAUTH with EMULATOR_WRAUTHSEQ values)
8. (Device saves authorization code, but does not respond)
9. Host Sends: 0xf0,0x1f,0x00,0x12,0x34,0x56,0x78,0x91,0x23,0x45,0x67,0x00, 0xa9 (CMD_BOOT_WRITEBLOCK followed by first block of firmware from registry example)
10. (device programs block at 0x1f00)
11. Device Responds: 0x01 (RSP_BOOT_BLOCKWRITTEN)
12. Host Sends: 0xf0,0x1f,0x08,0x12,0x34,0x56,0x78,0x91,0x23,0x45,0x67,0x00, 0xa9 (CMD_BOOT_WRITEBLOCK followed by second block of firmware from registry example)
13. (device programs block at 0x1f08)
14. Device Responds: 0x01 (RSP_BOOT_BLOCKWRITTEN)
15. Host Sends: 0xf0,0x1f,0x10,0x12,0x34,0x56,0x78,0x91,0x23,0x45,0x67,0x00, 0xa9 (CMD_BOOT_WRITEBLOCK followed by third block of firmware from registry example)
16. (device programs block at 0x1f10)
17. Device Responds: 0x01 (RSP_BOOT_BLOCKWRITTEN)
18. Host Sends: 0xf4 (CMD_BOOT_EXIT)
19. (Device resets itself)

Bootloader Commands

CMD_BOOT_EXIT – Exit the Bootloader

EMVER_EMULATOR_V2 ONLY

Message ID: CMD_BOOT_EXIT

Message length: 1 byte

Message direction: Host to device

Offset	Name	Value	Description
--------	------	-------	-------------

Offset	Name	Value	Description
0	CMD_BOOT_EXIT	0xF4	Command: Exit the bootloader

Description

When the device receives this command, it should exit the bootloader, reset the device, and resume normal operation.

CMD_BOOT_GETVERSION – Get Bootloader Version

EMVER_EMULATOR_V2 ONLY

Message ID: CMD_BOOT_GETVERSION

Message length: 1 byte

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_BOOT_GETVERSION	0xF5	Command: Return the firmware version of the bootloader code

Description

Return the firmware version of the bootloader code. The actual version that is returned is inconsequential. The host uses the RSP_BOOT_VERSION response as an indication that the device has successfully entered the bootloader.

Response	Description
RSP_BOOT_VERSION	Return the bootloader version

CMD_BOOT_SETWAKEPATTERN – Set Wake Pattern

EMVER_EMULATOR_V2 ONLY

Message ID: CMD_BOOT_SETWAKEPATTERN

Message length: 4 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_BOOT_SETWAKEPATTERN	0xef	Command: Set the wake pattern
1	Protocol	Number	Protocol for the Wake button

Offset	Name	Value	Description
2	Payload	Number	Button code for the Wake button
3	Address	Number	Address for the Wake button

Description

This command tells the device what protocol, payload, and address to wake on.

The following values are valid for the Protocol field:

Name	Value	Description
V2_WAKE_PROTOCOL_RC6	0x01	Wake key uses the RC6 protocol
V2_WAKE_PROTOCOL_QP	0x02	Wake key uses the Quatro Pulse protocol

The following values are valid for the Payload field:

Name	Value	Description
WAKE_KEY_POWER_TOGGLE	0x0c	Button code for the Sleep-toggle button
WAKE_KEY_DISCRETE_ON	0x29	Button code for the discrete-on button

The Address field should contain the address field in the wake pattern that it is listening for. Valid values are 0-7 for RC6 and 0-15 for Quatro Pulse.

Response	Description
RSP_BOOT_SETWAKEPATTERN	Wake pattern set successfully
RSP_BOOT_BADPATTERN	Bad wake pattern sent

CMD_BOOT_WRITEAUTH – Authorize Writing

EMVER_EMULATOR_V2 ONLY

Message ID: CMD_BOOT_WRITEAUTH

Message length: 5 bytes

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_BOOT_WRITEAUTH	0xF6	Command: Authorize writing
1	EMULATOR_WRAUTHSEQ1	0x23	First authorization byte
2	EMULATOR_WRAUTHSEQ2	0xCA	First authorization byte

Offset	Name	Value	Description
3	EMULATOR_WRAUTHSEQ 3	0x67	Second authorization byte
4	EMULATOR_WRAUTHSEQ 4	0xD 0	Third authorization byte

Description

Authorize firmware writes. The device should save the four auth bytes to a known location. Later, when the device is about to commit changes to flash memory, the device should check the known location to validate that the auth bytes match. This is intended to prevent random jumps from overwriting flash memory.

The device does not respond to this command.

CMD_BOOT_WRITEBLOCK – Write Firmware Block

EMVER_EMULATOR_V2 ONLY

Message ID: CMD_BOOT_WRITEBLOCK

Message length: arbitrary

Message direction: Host to device

Offset	Name	Value	Description
0	CMD_BOOT_WRITEBLOCK	0xF0	Command: write a block to firmware memory
1+	Firmware data	Number	Data to write to firmware memory

Description

The device should write the given data to firmware memory. The data comes from the registry on the host. The device manufacturer should have placed this data into the registry. The device manufacturer defines the block size and format of this data. It is expected that a destination address will be encoded somewhere in the block and a checksum will be encoded somewhere in the block. It is also possible that the device manufacturer will apply encryption to the data that it places in the registry. Then, it is the responsibility of the device to decrypt the firmware before storing it in flash memory.

Response	Description
RSP_BOOT_BLOCKWRITTEN	Block written successfully
RSP_BOOT_BADSERIALCHECKSUM	Block not written: checksum was incorrect
RSP_BOOT_BADWRITEAUTH	Block not written: write was not authorized

Bootloader Responses

RSP_BOOT_VERSION – Return Bootloader Version

EMVER_EMULATOR_V2 ONLY

Message ID: RSP_BOOT_VERSION

Message length: 2 bytes

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_BOOT_VERSION	0x04	Response: return bootloader version
1	Version	Number	Bootloader version

Description

The actual version returned is inconsequential. The host uses RSP_BOOT_VERSION response as an indication that the device has successfully entered the bootloader.

RSP_BOOT_SETWAKEPATTERN – Wake Pattern Set Successfully

EMVER_EMULATOR_V2 ONLY

Message ID: RSP_BOOT_SETWAKEPATTERN

Message length: 1 byte

Message direction: Device to host

Off set	Name	Value	Description
0	RSP_BOOT_SETWAKEPATTERN	0xEF	Response: Wake pattern set successfully

Description

The device should return this response to the host after receiving the CMD_BOOT_SETWAKEPATTERN request.

RSP_BOOT_BADPATTERN – Bad Wake Pattern Sent

EMVER_EMULATOR_V2 ONLY

Message ID: RSP_BOOT_BADPATTERN

Message length: 1 byte

Message direction: Device to host

Off set	Name	Value	Description
0	RSP_BOOT_BADPATTERN	0x05	Response: bad wake pattern sent

Description

The device should return this response to the host if it was unable to process the CMD_BOOT_SETWAKEPATTERN request.

RSP_BOOT_BADWRITEAUTH – Bad Write Authorization Sent

EMVER_EMULATOR_V2 ONLY

Message ID: RSP_BOOT_BADWRITEAUTH

Message length: 1 byte

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_BOOT_BADWRITEA UTH	0xF2	Response: Bad write authorization sent

Description

The device should return this response to the host if a write operation was attempted without being preceded by a proper CMD_WRITEAUTH request.

RSP_BOOT_BLOCKWRITTEN – Firmware Block Written Successfully

EMVER_EMULATOR_V2 ONLY

Message ID: RSP_BOOT_BLOCKWRITTEN

Message length: 1 byte

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_BOOT_BLOCKWRITTEN	0x01	Response: block was successfully written

Description

The device should return this response to the host when a block was successfully written to flash.

RSP_BOOT_BADSERIALCHKSUM – Bad Checksum in Firmware Block

EMVER_EMULATOR_V2 ONLY

Message ID: RSP_BOOT_BADSERIALCHECKSUM

Message length: 1 byte

Message direction: Device to host

Offset	Name	Value	Description
0	RSP_BOOT_BADSERIALCHECKSUM	0xF0	Response: Bad checksum in firmware block

Description

The device should return this response to the host if the checksum in the sent block was incorrect.

Format for Transmitting and Receiving IR

This section contains information about formatting data for IR ports and for the Data End message.

Data Format

Data for IR ports is encoded using a prefix byte and data bytes.

The prefix byte contains the value 100 in the upper three bits and the data length in the lower five bits. There can be as many as 30 bytes of data.

For the data bytes, the following format is used.

[h 16 15 14 13 12 11 10]

IR data is coded in a series of run-length coded bytes. The h bit indicates whether the signal is high (1, on - light produced/received) or low (0, off). Bits 16 through 10 form the number L which is the duration, in IR sample periods, during which the signal is high or low. L can range from 1 to 127. If a signal is high or low for more than 127 samples, multiple run-length coding bytes with the same h bit may be used. The IR sample period may be hard-coded in the firmware to 50 microseconds.

The IR LED is turned off when the Data End command is received. If no Data End command is received, the device returns an RSP_TX_TIMEOUT error.

When sending IR, the transmitting IR ports are set using the Set IR Transmit Ports command.

When receiving IR, a Last Received Port message is sent before the Data End byte to identify which IR port received the data.

IR Data End Message

For the data bytes, the following format is used.

[1 0 0 0 0 0 0 0]

The Data End command indicates the end of a set of IR data.

When returning data received to the host, you must send this value at the end of the IR data. The end of the IR data happens after there is a period of IR silence equal to the IR timeout value. The Data End message must be sent after the RSP_EQIRRXPORTEN and RSP_EQIRRXCFcnt messages.

When receiving data to transmit from the host, a Data End message will always indicate the end of data. If you do not receive the Data End message, you should return RSP_TX_TIMEOUT to the host.

Example: transmission

Because our sample period is 50 microseconds, 1 millisecond (ms) is 20 sample periods. So, our IR signal would be on for 10 ms, then off for 20 ms, then on again for 10 ms. To send this IR to port #1 with a 36 kHz carrier, the host would send the following sequence of bytes to the device:

```
// Set the output port.
[10011111] 0x9F CMD_PORT_IR - IR command
[00010100] 0x08 CMD_SETIRTXPORTS - set output ports
[00000001] 0x01 use the first port

// Set the carrier frequency - 36 kHz.
[10011111] 0x9F CMD_PORT_IR -IR command
[00000110] 0x06 CMD_SETIRCFS
[00000001] CP - 1 - carrier prescalar
[01000010] CF - 66 - carrier period
// Send the RLC.
[10001000] prefix byte - 8 bytes of IR data follows
[11111111] on for 127 sample (6.35 ms)
[11001001] on for 73 sample (3.65ms - bringing the total to 10 ms)
[01111111] off for 127 samples (6.35 ms)
[01111111] off for 127 samples (6.35 more ms - bringing the total to 12.7 ms)
[01111111] off for 127 samples (6.35 more ms - bringing the total to 19.05 ms)
```

Remote Control and Receiver-Transceiver Specifications and Requirements
for Windows Media Center in Windows Operating Systems

```
[00010011] off for 19 samples (.95 more ms - bringing the total to 20 ms)  
[11111111] on for 127 sample (6.35 ms)  
[11001001] on for 73 sample (3.65ms - bringing the total to 10 ms)  
// Send the data end.  
[10000000] IR Port Data End
```

Example: reception

Assume, as in the previous example, the sample period is set to 50 microseconds. If the same waveform is received, the device would return the following sequence to the host:

```
// IR Data:
[10001000] prefix byte - 8 bytes of IR data follows
[11111111] on for 127 sample (6.35 ms)
[11001001] on for 73 sample (3.65ms - bringing the total to 10 ms)
[01111111] off for 127 samples (6.35 ms)
[01111111] off for 127 samples (6.35 more ms - bringing the total to 12.7 ms)
[01111111] off for 127 samples (6.35 more ms - bringing the total to 19.05 ms)
[00010011] off for 19 samples (.95 more ms - bringing the total to 20 ms)
[11111111] on for 127 sample (6.35 ms)
[11001001] on for 73 sample (3.65ms - bringing the total to 10 ms)
// What port was used for reception:
[10011111] - 0x9F - RSP_PORT_IR
[00010100] - 0x14 - RSP_EQIRRXPORTEN
[00000010] - 0x02 - Wide-band receiver used.
// Carrier frequency:
[10011111] - 0x9F - RSP_PORT_IR
[00010101] - 0x15 - RSP_EQIRRXFCNT
[00000010] - 0x02 - CH - Carrier count high
[11100100] - 0xE4 - CL - Carrier count low
// EOM
[10000000] IR Port Data End
```

The host then knows the envelope of the IR signal. It knows it was received in the learning receiver. (This is redundant information because it was the host that told the device to listen with the learning receiver.)

The host also has enough information to calculate the carrier frequency. It knows that the envelope was high for 20 ms (from the RLC data), and it knows that there were 740 leading edges in the signal (from the RSP_EQIRXFCNT response). Because $740 / .02 = 37037$, this IR signal had a carrier frequency of about 37000 kHz.

As these examples illustrate, the IR data is broken into packets with a prefix byte indicating the length of the packet. The maximum packet size is 31 bytes (1 prefix byte plus 30 bytes of data). The minimum packet size is 2 bytes (1 prefix plus 1 byte of data). The firmware must decide how to break the data into packets. In the preceding example, the total RLC was 6 bytes long and this was in one packet. It is acceptable to break this into any number of packets. For example, if you wanted to break the same data into 3 packets of 2 bytes each, you would have the following:

```
[10000010] prefix byte - 2 bytes of IR data follows
[11101000] on for 200 samples (10 ms)
[01111111] off for 127 samples (6.35 ms)
[10000010] prefix byte - 2 bytes of IR data follows
[01111111] off for 127 samples (6.35 more ms - bringing the total to 12.7 ms)
[01111111] off for 127 samples (6.35 more ms - bringing the total to 19.05 ms)
[10000010] prefix byte - 2 bytes of IR data follows
[00010011] off for 19 samples (.95 more ms - bringing the total to 20 ms)
[11101000] on for 200 samples (10 ms)
[10000000] IR Port Data End
```

Packet size does not need to be consistent, so, for example, 6 bytes could be broken into one packet with 4 bytes and one packet with 2 bytes.

Suggested Firmware Memory Organization

It is recommended that you divide your firmware into the following four sections. This is not required, but this is how the Microsoft-produced IR Transceiver Version 2 or IR Receiver Version 3 memory is divided.

Bootloader

This section contains startup code, comm code, and the bootloader command switch. It has a checksum as the last 4 bytes which is checked at startup.

Main

This section contains IR reception code, as well as system and IR command switches. The main code is flash-upgradeable by the bootloader. It has a checksum as the last 4 bytes which is checked at startup.

Config

This section contains the unique serial number for the device, all strings, and all USB descriptors. It can be stored either in eeprom or in flash memory.

Sleep Key

This section contains firmware that runs under low power and watches for the Sleep key IR pattern. It should be flash upgradeable.

Port Driver Requirements

Windows Media Center uses infrared (IR) both for basic control of the computer and for controlling other IR-based devices used in conjunction with Windows Media Center. It's important that you have a basic understanding of the IR functionality in Windows Media Center and how it is used in conjunction with the software so that when you develop your hardware and port drivers you understand how the hardware and software will work together.

Following are some important terms to know when reading this section:

Main IR Receiver

Wide-band IR receiver used to receive IR commands from the Windows Media Center remote control and to translate those signals in order to interact with Windows Media Center.

Used in first run when the user is configuring a set-top box. The main IR receiver is used with the parse-and-match functionality to recognize a set-top box remote control.

Learning IR Receiver

Used in first run when the user is configuring a set-top box. The Learning IR Receiver is a close-range IR receiver that is used when Windows Media Center is unable to correctly identify the set-top box remote control using parse-and-match. The user is then guided through a step-by-step process so that Windows Media Center "learns" the numeric keys on the remote control so it can control the set-top box.

IR Output

Used to send IR signals from the computer to control a set-top box. The IR signals sent through the IR output port are sent either from a licensed IR database or from a user IR database, if the user has gone through the IR learning process.

IR output is important for any customer trying to control a set-top box. There are two primary scenarios that are important to Windows Media Center:

- The user wants to record a show when they are not at home. Windows Media Center needs to be capable of changing the channel for the user.

- The user uses a Media Center Extender in another room to watch television and they want to change channels. The Windows Media Center computer needs to relay the command to the set-top box.

Device Configuration Information

Used in first run to identify the type of receiver that is connected to the system and display appropriate error messages to the user so they can set up their computer.

Basic CIR Architecture

Support for consumer infrared (CIR) remote controls is implemented in the Windows operating systems by using a stack of drivers. Starting with Windows Media Center in Windows Vista, the architecture of this driver stack has been both extended and simplified to facilitate support of non-Microsoft CIR remote controls. The overall architecture of the CIR driver stack is shown in the following diagram.

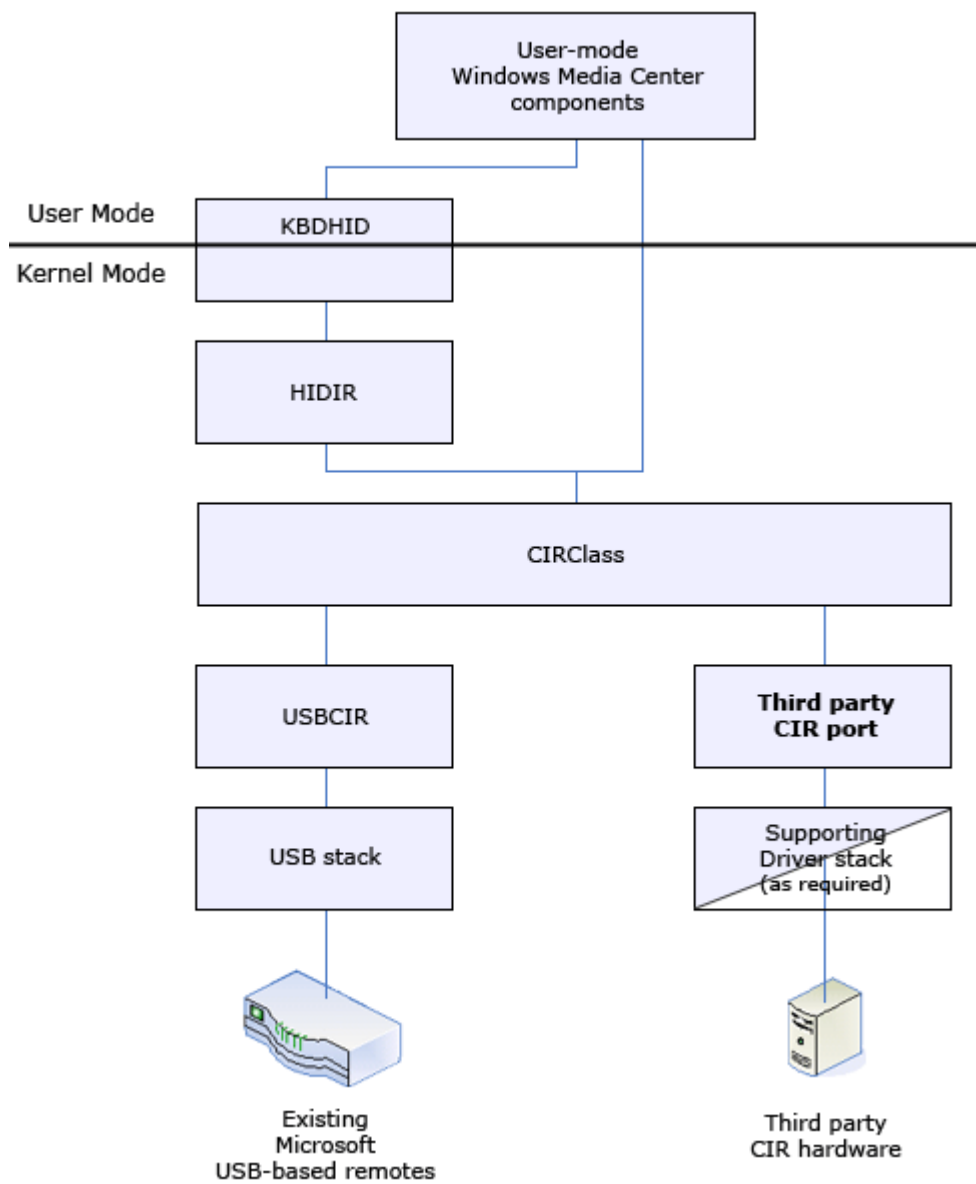


Figure 27: The CIR driver stack

The preceding diagram shows the components that are used to support remote control operations in Windows. Components provided by Microsoft are shown as blue-shaded blocks; Windows Media Center partner-developed components are shown as white blocks. Both user-mode and kernel-mode components are illustrated.

Windows support for CIR remote controls is anchored by the **CIRClass** driver. As shown in the preceding diagram, the upper edge of this driver provides the interface to the rest of the Windows system. As **CIRClass** receives data from underlying CIR Port drivers (such as **USBCIR** in the diagram), it routes that data (according to its own algorithms) to user-mode Windows Media Center components, to the human interface device (HID) stack, or to both of these destinations. During "IR blasting" operations, **CIRClass** sends data to one or more specific CIR Port drivers.

The lower edge of **CIRClass** provides an interface to one or more CIR Port drivers, including the Microsoft-supplied **USBCIR** driver. CIR Port drivers are responsible for controlling their CIR remote control hardware, along with translating data for that hardware between the standard format used by **CIRClass** (described in detail later in this document) and their hardware's proprietary format.

The **USBCIR** driver supports standard Microsoft-defined CIR devices that connect to the computer through USB. Continuing with our examination of the preceding diagram, **USBCIR** interfaces with **CIRClass** at its upper edge, and with the standard Microsoft USB driver stack at its lower edge.

Also shown in the preceding diagram is how an arbitrary, non-Microsoft-developed CIR remote control fits into the Windows system of CIR support. Non-Microsoft CIR Port drivers interface with the standard **CIRClass** driver at their upper edge, and with their hardware at their lower edge. Note that the CIR Port driver may interact with its hardware either directly or indirectly, through an additional set of drivers which may or may not be supplied by Microsoft. For example, a non-Microsoft CIR Port driver would directly interface with a PCI-based non-Microsoft CIR remote control (with the support of the standard Microsoft-supplied PCI bus driver). Alternatively, a non-Microsoft USB-based CIR remote control would interact with its device indirectly through the standard Microsoft-supplied USB driver stack.

Introduction to the CIRClass Framework

As described previously, the **CIRClass** driver is the interface between drivers that support one or more Windows CIR devices and the rest of the Windows operating system. The **CIRClass** driver provides a framework that centralizes common processing and simplifies the creation of CIR Port drivers. To help describe that framework, this section will describe how **CIRClass** supports and interfaces with the **USBCIR** driver supplied by Microsoft. The relationship between the device objects created by these two drivers is shown in the following diagram.

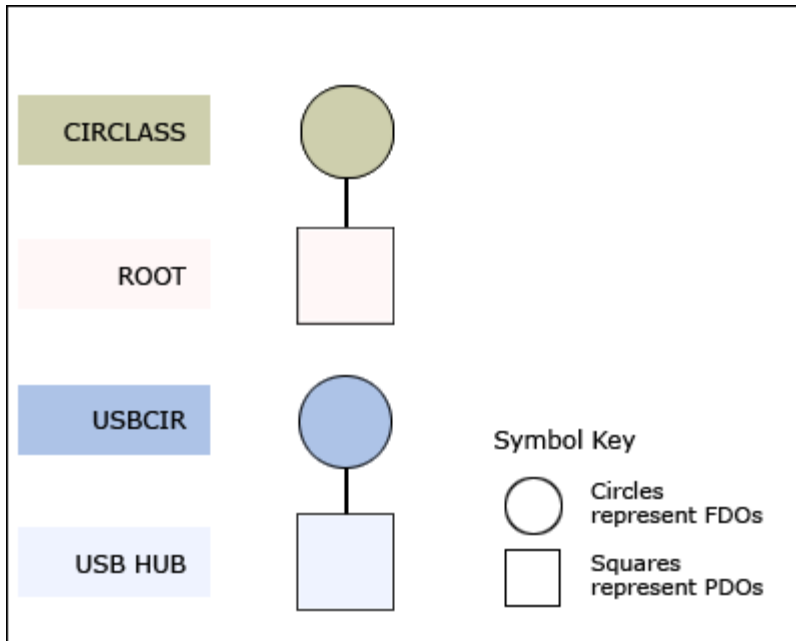


Figure 28: The device object relationship

In the preceding diagram, each device object is represented by either a circle or a square. Circles represent Functional Device Objects (FDOs) and squares represent Physical Device Objects (PDOs). The driver that creates each device object is shown on the left of the diagram, on the same line as the device object that it creates. Attachment between device objects is illustrated by a solid line between the attached device objects. Note that only the device objects that are directly relevant to **CIRClass** and **USBCIR** are shown.

In the preceding diagram, note that **CIRClass** and **USBCIR** are separate drivers, each with its own FDO. While not specifically shown in the preceding diagram, it's also important to note that there is only one instance of the **CIRClass** driver (and one **CIRClass** FDO) irrespective of the number of CIR Port drivers that are installed on a system. Thus, **CIRClass** does not provide a "mini-driver"-based framework such as that provided by, for example, the HIDClass or StorPort drivers. Rather, **CIRClass** supports CIR Port drivers that are independent function drivers. Thus, CIR Port drivers are entirely responsible for controlling their hardware and act as the power policy owner for their hardware devices.

CIR Port drivers are instantiated by the Windows Plug and Play Manager once the hardware device that they support is detected. For example, the USBCIR driver is instantiated as a result of the (standard Microsoft-supplied) USBHUB driver detecting that a supported **USBCIR** remote control device has been plugged in. Thus, when the USBHUB driver detects a **USBCIR** remote control device, it creates a PDO that describes this device and informs the Plug and Play Manager of the device's existence. The Plug and Play Manager then loads the appropriate function driver for the CIR device. In the preceding diagram, the driver that is loaded as a result of this action is the **USBCIR** driver. When **USBCIR** starts, it creates an FDO and attaches that FDO to the underlying device stack, according to the standard Windows practice.

Even though the **CIRClass** driver is available on all Windows systems, it is not installed until a CIR Port driver is installed. Whenever a CIR Port driver is installed, a device-specific coinstaller is invoked to install and start the **CIRClass** (if it has not already been installed and started). This device-specific coinstaller, which is provided by Microsoft, must be invoked by all CIR Port drivers as part of their installation procedure.

Note that there is no direct connection (that is, no attachment) between the **CIRClass** and **USBCIR** drivers. As a result, **CIRClass** engages in a handshaking exchange (described later in this document) with each CIR Port driver as that port driver is started.

CIR Version 1 DDI and Version 2 DDI

A newer version of the Consumer IR Device Driver Interface (CIR DDI), called the version 2 DDI, is being introduced with the release of Windows 7. The version 2 DDI is meant to replace and augment the first CIR DDI, which was distributed with Windows Media Center in Windows Vista.

The version 2 DDI is designed as a superset of the version 1 DDI with backwards and forwards compatibility, which was designed for from the beginning. This compatibility will be discussed in future sections.

Note on Documentation Conventions

Throughout this section, the version of the CIR DDI that was distributed with Windows Media Center in Windows Vista will be referred to as the version 1 DDI. The newer DDI will be referred to as the version 2 DDI.

If a section does not explicitly specify which version of the DDI it refers to, it can be assumed that it applies to both the version 1 DDI and the version 2 DDI.

New sections that only apply to the version 2 DDI will be labeled version 2 DDI only.

Backwards and Forwards Compatibility

The version 2 DDI is designed to be both backwards and forwards compatible. This means that the DDI that the port driver implements and the DDI that the class driver implements may be two different versions, but if the port driver is written with backwards compatibility in mind, the drivers should be able to adapt gracefully to these differences in versions.

Specifically, this means:

- A port driver that is written for the version 1 DDI will work with the version 2 DDI class driver with absolutely no modification.
- A port driver that is written for the version 2 DDI is required to be aware that it may run on a system with the version 1 DDI class driver and adjust its behavior accordingly.

Because version 2 DDI is a superset of the version 1 DDI, the first assertion is virtually guaranteed. This works because the version 2 class driver inside of Windows Media Center is aware of both the version 1 DDI and the version 2 DDI. If the version 2 class driver sees a version 1 port driver, it is able to treat it as a port driver with limited capabilities. See the Proper Implementation of Version 1 and Version 2 Devcaps section for more information about how DDI versioning is accomplished.

On the other hand, a version 2 port driver may have capabilities that are unknown to the version 1 class driver. Specifically, a version 2 port driver may support features such as programmable wake, blast-only, or narrow band pass filter (BPF). Because the version 1 class driver does not implement these features, the driver writer needs to be aware that their port driver may run with a version 1 class driver, but their device may not be able to function correctly in all circumstances.

Features Added to Version 2 DDI

The version 2 DDI was created to support additional device capabilities, allowing OEM partners to create a wider variety of CIR devices with a wider variety of hardware capabilities.

Note that you will be bound by the requirements in the Windows Logo Program when building your device.

Additional protocol support

The version 2 class driver adds support for a newer remote control protocol. The SMK Quatro Pulse protocol is now available as an option for remote controls manufacturers. Contact remotemc@microsoft.com for details about licensing this protocol. In June 2009, all receiver partners must be able to handle both the MC RC 6 and the MC SMK QP protocol per the Windows Logo Program.

Dynamic wake programming

Hardware devices need to operate in extremely low-power situations while waiting for the Sleep toggle or Wake button to turn the PC back on. In this low-power state, the hardware is often only capable of watching for a single IR signature to wake the system. Because there are a wide variety of options available for protocol and Sleep-button combinations (Sleep toggle versus discrete Wake/Sleep), the system must be able to program the hardware dynamically to wake on the correct Sleep button. The version 2 DDI adds a set of IOCTLs and capabilities flags to control dynamic wake programming.

Dynamic active input device selection

The previous implementation of the IR class driver relied on the concept of *an active input device*. This concept was used to prevent the host from receiving multiple key presses if multiple IR receivers were present on the system. The heuristic used to determine which IR receiver was the active IR receiver relied on the order of PNP events and was difficult for the end user to comprehend. The version 2 class driver relies on a more dynamic mechanism to filter out duplicate input if multiple receivers are present. With the version 2 class driver, there is no need to switch the active input device because all IR receivers are active with the class driver filtering out duplicate input.

Receiver capabilities: input only on long-range receiver

A partner may want to build a receiver device that uses a long-range receiver part with a narrow band pass filter (BPF). This long-range receiver would allow IR input to function correctly, but would not allow parse-and-match to function correctly.

Receiver capabilities: no input on long-range receiver

An OEM partner may want to build a receiver device that uses the long-range receiver part for parse-and-match, but not for IR input. This might be desirable because the partner may be distributing this receiver with a non-IR remote. In this case, the long-range receiver would be used for parse-and-match operation, but not for IR input operation.

Receiver capabilities: no long-range receiver

An OEM partner may want to build a receiver device that does IR blasting and IR learning, but they don't want to put a long-range receiver in it because remote control input is done using a different device.

Receiver capabilities: no receiver at all (blasting only).

An OEM partner may want to build an IR device that doesn't do any IR receiving. This device might only do IR blasting. This would be desirable for partners whose hardware might be with the TV tuner hardware in a position where IR reception would be impossible (such as in a closet or in the back of the PC).

Notes on the Updated Emulator Interface

In addition to the version 2 DDI, a newer IR emulator interface is also being published. This newer emulator interface is related to version 2 DDI in that it defines the emulator protocol

necessary to support the version 2 DDI. However, it is not strongly tied to this DDI, meaning that old emulators will still work with the newer version 2 DDI port driver.

CIRClass and CIR Port Interface Details

There are three mechanisms by which CIR Port drivers and the **CIRClass** driver interact. All three mechanisms are required and must be implemented by every CIR Port driver.

The three mechanisms are:

Device-Specific Class Installer (used during installation): Whenever a CIR Port driver is installed, it must invoke the Microsoft-supplied device-specific class installer for the **CIRClass** driver. This installer installs and starts the **CIRClass** driver if it has not already been installed and started (as a result of another CIR Port driver being installed).

Device Interface Registration and Enabling (used during initialization and teardown): Whenever a CIR Port device is enumerated, the driver for that device must register and enable a device interface for the CIR Port device, using device interface GUID GUID_DEVINTERFACE_IRPORT. The **CIRClass** driver "listens" for devices to be enabled with this interface and takes specific actions (described later) as a result. Typically, the CIR Port driver registers the device interface within its Add Device routine and will enable the interface within its IRP_MN_START_DEVICE processing code. Note that, if a CIR Port device is removed from the system, the CIR Port driver must disable its device interface. (However, the device interface need not be deleted.)

IOCTLs (used during initialization and normal operations): The **CIRClass** and CIR Port drivers exchange data using a defined set of IOCTLs (that is, IRP_MJ_DEVICE_CONTROL requests), that are described later in this document. Note that IOCTLs are the only type of I/O request that is exchanged between **CIRClass** and CIR Port drivers. Standard read and write (IRP_MJ_READ and IRP_MJ_WRITE) requests are not used.

CIR Port Driver Installation

The **CIRClass** driver and its associated INF file are present on all supported Windows systems. However, the driver is neither installed nor started until a CIR Port device is installed. The **CIRClass** driver is installed and started as a result of the CIR Port driver's INF file invoking the **CIRClass** device-specific class installer. This device-specific coin installer is named CIRCoInst.dll. The installation procedure for all CIR Port drivers must invoke CIRCoInst.dll as a device-specific coin installer during their installation process. The INF command to invoke the coin installer will be similar to the following:

```
;
;--- usbcir_Device Coinstaller installation -----
;
[DestinationDirs]
IR_CoInstaller_CopyFiles = 11
[usbcir_Device.NT.CoInstallers]
AddReg=IR_CoInstaller_AddReg
CopyFiles=IR_CoInstaller_CopyFiles
;
; IR CoInstaller
;
[IR_CoInstaller_AddReg]
HKR,,CoInstallers32,0x00010008, "CIRCoInst.dll,IRCoInstaller"
[IR_CoInstaller_CopyFiles]
CIRCoInst.dll
```

CIR Port Device Initialization

As previously described, each CIR Port device will be loaded as a result of its device being enumerated by the underlying bus driver. For example, the Microsoft-supplied **USBCIR** devices are enumerated by the USBHUB driver.

Device Interface Registration and Enabling

As each CIR Port driver identifies an instance of a device it supports, it creates whatever FDOs it requires. For each instance of an IR interface, each CIR Port driver must register and enable a device interface using the GUID `GUID_DEVINTERFACE_IRPORT`.

It is important to note that a CIR Port driver should not enable this interface until its interface is ready for use and it is ready and willing to receive requests from **CIRClass**. If a CIR Port driver needs to delay between device discovery and enumeration and the device's ready state (to POST the device, download microcode, interrogate or calibrate the device, and so on), it should delay enabling its device interface until its device is ready to receive requests.

`GUID_DEVICEINTERFACE_IRPORT` is defined as follows:

```
// {064F8C82-77B2-445e-B85D-C4E20F942FE1}  
DEFINE_GUID(GUID_DEVINTERFACE_IRPORT,  
            0x64f8c82, 0x77b2, 0x445e, 0xb8, 0x5d, 0xc4, 0xe2, 0xf, 0x94, 0x2f, 0xe1);
```

CIRClass Handshaking with CIR Port Drivers

When **CIRClass** discovers an instance of a CIR Port device as a result of its device interface being enabled, **CIRClass** will send a handshake IOCTL (`IOCTL_IR_HANDSHAKE`) to the newly-created port device object instance. This IOCTL informs the CIR Port driver that its device has been detected by **CIRClass**.

On receiving the `IOCTL_IR_HANDSHAKE`, the CIR Port driver must complete the request. When this IOCTL is completed, **CIRClass** may start immediately sending IR-related requests to the new CIR Port device instance. *Note that a CIR Port driver may not delay completion of the `IOCTL_IR_HANDSHAKE` IRP. This IRP must be completed immediately and synchronously when received by IRPORT.* If a CIR Port driver needs to delay before it is ready to receive requests from **CIRClass**, it should delay enabling its device interface.

If the handshake operation fails, **CIRClass** will log an error and disregard that CIR Port device instance.

Successful completion of the handshake signals a completed binding between **CIRClass** and a given CIR Port device instance.

Device Capabilities for Version2 DDI

When **CIRClass** or other upper-level software requires the capabilities of the IR receiver, it sends an `IOCTL_IR_GET_DEV_CAPS` request to the port driver. On receipt of this IRP, the port driver must fill in the specific hardware capabilities and complete the IRP.

Because backward compatibility is required, a version 2 port driver may need to fill in a version 1 capabilities structure. Likewise, a version 1 port driver is required to (partially) fill in a version 2 capabilities structure.

On receiving the **IOCTL_IR_GET_DEV_CAPS** request, the port driver must check the size of the out buffer. If the out buffer is big enough to hold an **IR_DEV_CAPS_V2** structure, the port driver should assume that it is working with a version 2 class driver. It should fill in the entire **IR_DEV_CAPS_V2** structure, and set the **IR_DEV_CAPS_V2.ProtocolVersion** member to **DEV_CAPS_PROTOCOL_VERSION_V2** (0x200). If, however, the buffer is only big enough to hold an **IR_DEV_CAPS_V1** structure, the port driver should assume that it is working with a version 1 class driver. It should fill in the **IR_DEV_CAPS_V1** structure and set **IR_DEV_CAPS_V2.ProtocolVersion** member to **DEV_CAPS_PROTOCOL_VERSION_V1** (0x100). If the buffer is not big enough to hold an **IR_DEV_CAPS_V1** structure, the port driver should fail the IRP.

Proper Implementation of Version 1 and Version 2 Devcaps

The port driver is able to infer the DDI version of the class driver by looking at the size of the output buffer in the **IOCTL_IR_GET_DEV_CAPS** request. By adjusting its behavior based on the version of the DDI, the version 2 port driver is able to work with version 1 class drivers.

Likewise, a version 1 port driver must be able to work with a version 2 class driver. This is only possible if the version 1 port driver has properly implemented the **IOCTL_IR_GET_DEV_CAPS** handler correctly. Namely, the version 1 port driver must accept buffers that are larger than **sizeof(IR_DEV_CAPS_V1)** and it must properly set the **IR_DEV_CAPS_V1.ProtocolVersion** member to indicate that it subscribes to the (limited) version 1 DDI.

Wake Pattern Programming

When **CIRClass** wants to direct the hardware to wake on a specific signature, it sends an **IOCTL_IR_SET_WAKE_PATTERN** IRP to the port driver. This IRP includes an **IR_SET_WAKE_PATTERN_PARAMS** structure, which contains the details of the wake pattern that the hardware needs to watch for. The port driver is responsible for programming the hardware to watch for this wake pattern and it should not complete the IRP until the hardware has been programmed. Because the wake programming may require a moderate amount of I/O to the device, the port driver may need to return **STATUS_PENDING** for (to "pend") this IRP while the programming is taking place.

Alternatively, the hardware may be designed to watch for all valid and required wake patterns. If so, the port driver can ignore any **IOCTL_IR_SET_WAKE_PATTERN** IRPS as it already watches for all required wake patterns.

There are two reasons to change the wake pattern:

- **Choice of protocol.** Remote controls are available using both the RC6 protocol and the Quatro Pulse protocol. The wake hardware needs to be programmed to listen for the specific protocol that is being used by the user's remote control.
- **Choice of Sleep button.** Most current Windows Media Center remote controls offer a single Sleep button, which can either turn Windows Media Center on or off. This button is called a *Sleep toggle* because it toggles the power state. However, the option exists to implement a pair of discrete Sleep buttons, called *discrete Sleep* and *discrete Wake*. The discrete Sleep button will put the PC into the sleep state, but it won't wake it up. The discrete Wake button will wake the PC, but it won't put it to sleep. This is a very desirable feature for power users who want to program remote control macros that are guaranteed to put the PC into a known state. If the user has a remote control with a discrete Sleep/Wake button pair, the hardware must be programmed to wake when the user presses the discrete Wake button.

CIRClass will start with a default wake pattern using the RC6 protocol with a Sleep toggle button, since this is the most common remote control configuration. During the course of normal

operation, **CIRClass** will receive additional information in the form of remote control button presses that may indicate that a different pattern should be used. When this happens, it will use the **IOCTL_IR_SET_WAKE_PATTERN** IRP to reprogram the hardware.

CIRClass is most likely to send an **IOCTL_IR_SET_WAKE_PATTERN** request in two circumstances:

- When the drivers are being initialized, **CIRClass** tracks the current wake pattern across reboots and sends the wake pattern to the port driver when the port driver loads.
- When the user presses a key that causes **CIRClass** to decide that a different wake pattern should be used. This is likely to happen in two specific cases:

Change of protocol. If the wake pattern is currently set to RC6 and the user presses a button on a Quatro Pulse remote control, **CIRClass** will reprogram the device to wake on the Quatro Pulse Sleep toggle button.

Change of button code. If the user presses the discrete off button, **CIRClass** will assume that the remote control has a discrete on button and program the hardware to respond to this button.

Notes

- Although unlikely, this can happen while the user is putting the PC into a low-power state. Blocking the **IOCTL_IR_SET_WAKE_PATTERN** IRP for very long may cause the PC to delay going into the low-power state.
- If the port driver is unable to program the hardware to wake on the specific button code before the PC goes to sleep, the Sleep button on the remote control will appear to be broken, but only that one time. The next time the driver loads, **CIRClass** will again attempt to program the hardware to wake on the correct pattern.

The port driver is responsible for reporting its wake abilities using the appropriate bits in the **DevCapsFlags** and **WakeProtocols** members of the **IR_DEV_CAPS_V2** structure. For more information, see the documentation for **IR_DEV_CAPS_V2**.

IR Port Driver and CIRClass Data Exchange

CIRClass and CIR Port drivers communicate and exchange data using IOCTLs and data structures defined later in this document. Note that the only I/O requests that **CIRClass** sends to a CIR Port driver are IOCTLs; read, write, and other I/O function codes are never used.

Data Format

The **CIRClass** driver exchanges IR data with CIR Port drivers in a standard format. This format is referred to as "Run Length Coded" (RLC) format.

In run length coding, CIR data is encoded according to duration of high or low signal. These durations are recording the demodulated signal (the envelope) and not the modulated signal.

Each LONG (32 bits) of the IR represents either a period of time that the signal is on or a period of time that the signal is off. If the LONG is positive, the signal is on. If the LONG is negative, the signal is off. The absolute value of the LONG is the duration of time that the signal is either high or low.

Therefore, if the IR stream is high for 500 microseconds, low for 200 microseconds, and high for 150 microseconds (typically represented as "500 -200 150"), then the IR buffer contains the following:

0x01F4 (500 microseconds high)

0xFF38 (200 microseconds low)

0x0096 (150 microseconds high)

Data Flow

This section provides an overview of the data exchange process that takes place between **CIRClass** and a CIR Port driver. This overview will serve as an introduction to the topic, in preparation for reading the sample CIR Port driver code, the IOCTL definitions, and the data structure definitions.

While reading the overview in this section, keep in mind that the code for the example driver, which is described in a later section of this document, provides the most detailed example of how data is exchanged between a CIR Port driver and the **CIRClass** driver. That code is also the ultimate authority on the rules of those exchanges, such that if any information in this document differs from the sample implementation, the sample implementation should be considered correct.

Receive Data

A CIR Port driver reads data from its IR device as a result of receiving an IOCTL_IR_RECEIVE request from **CIRClass**. This IOCTL utilizes direct I/O, and thus its data buffer is described using a Memory Descriptor List (MDL). The data buffer for the IOCTL_IR_RECEIVE request is formatted as an IR_RECEIVE_PARAMS structure:

```
typedef struct _IR_RECEIVE_PARAMS {  
    OUT ULONG_PTR DataEnd;  
    IN  ULONG_PTR ByteCount;  
    OUT LONG     Data[1];  
}IR_RECEIVE_PARAMS, *PIR_RECEIVE_PARAMS;
```

Before sending the IOCTL_IR_RECEIVE request to the CIR Port driver, the **CIRClass** driver initializes the fields in the IR_RECEIVE_PARAMS structure as follows:

- **ByteCount** – This field is set to the maximum number of data bytes that can be accommodated in the receive data buffer.

When a CIR Port driver receives a packet of IR data from its hardware, it converts that data into the RLC format previously described. It then returns the RLC data packet in the Data field of the IR_RECEIVE_PARAMS structure.

Note that only two events can cause a CIR Port driver to consider an IR data packet "complete" and therefore complete a pending IOCTL_IR_RECEIVE:

- The CIR Port driver completely fills the data buffer with IR data. In this case, the CIR Port driver sets the DataEnd field of the IR_RECEIVE_PARAMS structure to FALSE.
- The IR sample period elapses, indicating the end of a stream of key presses. This sample period is the time period that must elapse without IR data being received, after receiving one or more IR key presses, before a "packet" of IR data is considered complete. In this case, the CIR Port driver sets the DataEnd field of the IR_RECEIVE_PARAMS structure to TRUE. The default timeout is 100 milliseconds.

In both cases, before completing the IOCTL_IR_RECEIVE request, the CIR Port driver sets the ByteCount field of the IR_RECEIVE_PARAMS structure to the number of bytes of RLC-coded data being returned in the data buffer. To complete the request, the CIR Port driver sets the request's completion status to STATUS_SUCCESS and the request's information field to the number of bytes returned in the IOCTL data buffer. Note that the byte count in the information field includes both the RLC data bytes returned and the overhead of the IR_RECEIVE_PARAMS structure.

Priority Receives

In most cases, during normal operations, the **CIRClass** driver will keep an `IOCTL_IR_RECEIVE` in progress with the CIR Port driver. In some cases, Windows will need to start a new receive operation that bypasses and leaves pending any `IOCTL_IR_RECEIVE` requests that might already be in progress. This operation is referred to as a "Priority Receive."

The start of a Priority Receive operation is always indicated to a CIR Port driver by an `IOCTL_IR_ENTER_PRIORITY_RECEIVE` request. `IOCTL_IR_PRIORITY_RECEIVE` passes the CIR Port driver an `IR_ENTER_PRIORITY_RECEIVE` structure, with the following format:

```
typedef struct _IOCTL_IR_ENTER_PRIORITY_RECEIVE_PARAMS {
    IN ULONG_PTR Receiver;
    IN ULONG_PTR Timeout;
}IOCTL_IR_ENTER_PRIORITY_RECEIVE_PARAMS, *PIOCTL_IR_ENTER_PRIORITY_RECEIVE_PARAMS;
```

The Receiver field of the `IR_ENTER_PRIORITY_RECEIVE` structure indicates on which IR port of the indicated IR device the receive should be performed. The Timeout field of this structure indicates the number of milliseconds that the CIR Port driver should use to determine IR data packet completion for subsequent `IOCTL_IR_PRIORITY_RECEIVE` requests.

On receipt of an `IOCTL_IR_ENTER_PRIORITY_RECEIVE` request, a CIR Port driver enters Priority Receive mode and proceeds as follows:

- Immediately stops processing any pending `IOCTL_IR_RECEIVE` requests for the indicated device. Note that any pending `IOCTL_IR_RECEIVE` requests remain pending in the CIR Port driver and are not completed by the CIR Port driver in response to this `IOCTL`.
- Does whatever processing on its device that may be necessary to set the new IR timeout value.
- Enables the device to use the proper receiver part for the priority receive.
- Queues any newly-arriving `IOCTL_IR_RECEIVE` requests for processing after leaving Priority Receive mode.
- Completes the `IOCTL_IR_ENTER_PRIORITY_RECEIVE` as soon as its IR device is ready to receive IR data with the new timeout value.
- Awaits receipt of an `IOCTL_IR_PRIORITY_RECEIVE` request, which should be immediately forthcoming from the **CIRClass** driver, following completion of the `IOCTL_IR_ENTER_PRIORITY_RECEIVE` request.

While in Priority Receive mode, the CIR Port driver processes `IOCTL_IR_PRIORITY_RECEIVE` requests similarly to a standard `IOCTL_IR_RECEIVE` request. That is, the `IOCTL_IR_PRIORITY_RECEIVE` request is completed when the specified timeout period elapses or the supplied data buffer is full. Note that the `IOCTL_IR_PRIORITY_RECEIVE` data buffer returned by a CIR Port driver contains an `IR_PRIORITY_RECEIVE_PARAMS` structure. This structure has the following format:

```
typedef struct _IR_PRIORITY_RECEIVE_PARAMS {
    OUT ULONG_PTR DataEnd;
    IN ULONG_PTR ByteCount;
    OUT ULONG_PTR CarrierFrequency;
    IN LONG Data[1];
}IR_PRIORITY_RECEIVE_PARAMS, *PIR_PRIORITY_RECEIVE_PARAMS;
```

While in Priority Receive mode, any IOCTL_IR_RECEIVE requests that the CIR Port driver receives must be queued for later processing. While in Priority Receive mode, any non-receive-related requests (such as IOCTL_IR_TRANSMIT or other IOCTLs) are processed as normal; Priority Receive mode only affects processing of receive-related packets. The only two receive-related requests that are processed by the CIR Port driver while in Priority Receive mode are the following:

- IOCTL_IR_PRIORITY_RECEIVE – Receiving this request indicates that the CIR Port driver should continue with the current Priority Receive operation, using the timeout value that is specified in the most recently received IOCTL_IR_ENTER_PRIORITY_RECEIVE request.
- IOCTL_IR_EXIT_PRIORITY_RECEIVE – Receiving this request indicates that the CIR Port driver should exit Priority Receive mode and return to its normal mode receive processing. In this case, the CIR Port driver does whatever processing is necessary to restore its previously-used receive timeout period to its hardware. Following receipt of an IOCTL_IR_EXIT_PRIORITY_RECEIVE request, a CIR Port driver returns to using the IOCTL_IR_RECEIVE request that was pending before receipt of the IOCTL_IR_ENTER_PRIORITY_RECEIVE (if there was one) for newly-arriving IR data.

Transmit Data

When Windows wants to send CIR data, it uses the IOCTL_IR_TRANSMIT request. Transmit request processing is only a bit more complicated than processing a receive request. IOCTL_IR_TRANSMIT uses both IOCTL data buffers. The input buffer (which uses buffered I/O) contains an IR_TRANSMIT_PARAMS structure that describes the parameters for the transmit request.

The format of this structure is as follows:

```
typedef struct _IR_TRANSMIT_PARAMS {
    IN ULONG_PTR TransmitPortMask;
    IN ULONG_PTR CarrierPeriod;
    IN ULONG_PTR Flags;
    IN ULONG_PTR PulseSize;
} IR_TRANSMIT_PARAMS, *PIR_TRANSMIT_PARAMS;
```

Note that the transmit port mask indicates on which ports of the indicated IR device the specified data should be transmitted.

The IOCTL_IR_TRANSMIT data buffer, described by an MDL, is used to supply the data in RLC format to be sent by the device controlled by the CIR Port driver. The data buffer comprises a series of transmit "chunks," each of which is described by an IR_TRANSMIT_CHUNK structure with the following format:

```
typedef struct _IR_TRANSMIT_CHUNK {
    ULONG_PTR    OffsetToNextChunk;
    ULONG_PTR    RepeatCount;
    ULONG_PTR    ByteCount;
    LONG        Data[1];
} IR_TRANSMIT_CHUNK, *PIR_TRANSMIT_CHUNK;
```

The repeat count indicates the number of consecutive times that the RLC data in the Data field of the structure is to be transmitted. The number of bytes of RLC data in the Data field is indicated by the ByteCount field of the structure.

The last chunk to be transmitted is identified by an OffsetToNextChunk field value of zero.

CIR Port drivers are required to process IOCTL_IR_TRANSMIT requests synchronously. That is, a CIR Port driver must not complete an IOCTL_IR_TRANSMIT request until all the data described by that request has been transmitted.

Example CIR Port Driver – Hardware Design Requirements and Considerations

This section describes an example of requirements and considerations for designing a consumer infrared (CIR) port driver.

Software Decoding of Infrared

Implicit to this driver model is that the infrared signal is decoded in software. Hardware that decodes the infrared signal into a payload or keystroke is not supported in this model. This is done for several reasons, including the following:

- Decoding the infrared signal in software allows us to decouple the receiver implementation from the remote implementation. This way, any Windows Media Center-compatible remote will work with any Windows Media Center-compatible receiver. There is no reason to modify the receiver hardware to support a new remote protocol.
- Multiple infrared remote (IR) protocols can be supported simultaneously. There is no need to put the receiver into "Protocol #1 mode" or "Protocol #2 mode". The software that decodes the protocol can decode numerous protocols and doesn't need to be put into a specific protocol mode.
- Returning remote line controller (RLC) data allows us to do learning and parse-and-match in a hardware-independent way. The learning algorithm is implemented in software, as is the IR database, thus ensuring a consistent learning experience across hardware implementations.
- Filtering of input can happen in software. This is useful, for instance, when a single computer has multiple IR receivers (all the more likely now, considering current MPEG encoder cards are already being distributed with their own IR receiver.). In this case, the IR driver stack is smart enough to realize that there are multiple receivers and that it can ignore input from one of the receivers, thus preventing the user from seeing multiple responses from a single key press. Another place this is useful is in remote addressing. It is not hard to imagine multiple Windows Media Center computers in a store environment, or in an enthusiast's home. In that case, the Windows Media Center software can be configured to only accept input from a given remote control. This way, remote control #1 can control computer #1 exclusively and remote control #2 can control computer #2 exclusively.

Sampling Resolution

When receiving IR or transmitting IR, your hardware needs to operate with a 50-microsecond resolution. When transmitting, this means that you can effectively round the RLC durations to the nearest 50 microseconds. When receiving, this means that you only need to return RLC that is accurate to 50-microsecond durations.

Differences between Learning (Wide Band) Receiver and Long Range (Narrow Band) Receiver

There are two types of light detectors needed for an IR transceiver:

- **Long Range Receiver.** This is the receiver that is used during normal operation of Windows Media Center. It is designed to receive input from 3 to 30 feet away. It de-multiplexes the signal in hardware. It is optimized to receive a 36 kilohertz (kHz) signal, but it can also receive a (degraded) signal from 30 kHz to 60 kHz (or more).

If your long-range receiver has a narrow band pass filter (BPF), you must set the **V2_DEV_CAPS_NARROW_BPF** bit. Note that this results in a less-than-optimal experience for users while setting up Windows Media Center for set-top box control because they will have to complete the long learning process instead of the shorter parse-and-match process.

- **Learning Receiver.** This is the receiver that is used during IR Learning. It is designed to be used from 2 inches away. It does not de-multiplex the signal in hardware. It is optimized to receive a signal from 30 kHz to 60 kHz (or more).

If your IR transceiver is input only (no blasting and therefore no learning), a long-range receiver is required. If your IR transceiver also does IR output (blasting), both a long-range and learning receiver are required.

Emitter Detection

When responding to the `IOCTL_IR_GET_EMITTERS` request, the hardware only needs to detect if something is plugged into the emitter port. If, for instance, a user plugs a pair of headphones into an emitter port, it is acceptable to return that an emitter is detected in that port.

Emitter Multiplexing

In reading the reference for `IOCTL_IR_TRANSMIT`, you may notice that the transmit port is a bitmask. This means that it's possible to transmit to two different emitter ports with the same `IOCTL`. There is some freedom for design here. If your hardware can only output to one port at the same time, and the `IOCTL` is asking you to transmit to two ports, your driver can first transmit to the first transmit port and then transmit to the second port.

Pulse Mode Remotes

Pulse mode is no longer required.

Wake From Remote

To support the "Wake From Remote" feature, your hardware needs to do several things:

- It must resume from standby mode using the Sleep button for the particular IR protocol for which the hardware is optimized. Resume-from-standby must do hardware decoding of the protocol and operate when the Windows Media Center computer is in a state of lower power consumption.
- It must wake from S1 or S3. Resuming or waking from S4 or S5 is optional.

- Power consumption requirements are defined by the bus and architecture used by the IR receiver. For example, USB allows 2.5 mA during suspend and a variable amount of current depending on whether it is a high-power or lower-power device.
- If using a USB device, it is recommended that the device be able to operate correctly when it is plugged into a passive hub.
- It must properly indicate user presence to the operating system when waking the system. This can be tested by first waking the system with the remote control, and then by running a scheduled task. The monitor should turn on when waking from the remote control, but not when waking from running a scheduled task
- It needs to fire a hardware interrupt to wake the system when it sees the Sleep key's IR signature.
- It needs to be software-configurable. Depending on what IR protocol is being used, the IR signature for the Sleep key can change from computer to computer. This signature is stored in the registry. The IR port driver needs to take this information from the registry and program the hardware to wake on this signature.
- It needs to call `PoSetSystemState(ES_USER_PRESENT)` when the device causes the system to wake up. This call causes the computer to turn on video and audio. It is important to do this only when the user presses the Sleep button on the remote. Because this call causes video and audio to start playing, implementing it incorrectly could cause the computer to turn on and play music in the middle of the night.

Wake Signatures

A wake signature is an IR pattern that represents a remote control key that can wake the PC from a low-power state. There are different wake signatures required for device operation:

- RC6 Protocol, Sleep Toggle Key (most common and currently the default signature)
- RC6 Protocol, Discrete Wake
- Quatro Pulse Protocol, Sleep Toggle Key
- Quatro Pulse Protocol, Discrete Wake

Note that there is no wake signature for the Discrete Sleep keys because these keys are designed to put the PC into a low-power state. They are not designed to wake the PC from a low-power state, so they are not included in the list of wake patterns.

It is worth noting that the payloads for these patterns have a number of bits that aren't significant for wake functioning. The implementation should be aware of these bits and ignore them accordingly.

- The RC6 protocol has a toggle bit. The wake pattern decoding should ignore this toggle bit.
- The Quatro Pulse protocol has a checksum bit. The wake pattern decoding should ignore this checksum bit.

These bits are indicated in the table below.

Note It is important to implement these bits properly—specifically the address bits. Many wake implementations have failed to implement these properly in the past.

Protocol	Button	Button Code	Payload	Don't Care Bits
RC6	Sleep	12	Customer	Toggle Bit

Protocol	Button	Button Code	Payload	Don't Care Bits
	Toggle		Code=32783 System=4 Address=Address* KeyCode=12	
RC6	Discrete Wake	41	Customer Code=32783 System=4 Address=Address* KeyCode=41	Toggle Bit
Quatro Pulse	Sleep Toggle	12	Flag=2** ID = Address* Maker=0x22 Device=0x01 Extension=0x00 KeyCode=12	Checksum
Quatro Pulse	Discrete Wake	41	Flag=2** ID = Address* Maker=0x22 Device=0x01 Extension=0x00 KeyCode=41	Checksum

* The **Address** field is specified by the host through the Address member of the **IR_SET_WAKE_PATTERN_PARAMS** structure. When the port driver receives an **IOCTL_IR_SET_WAKE_PATTERN** IRP, it should program the hardware to listen for the address as specified in this structure. If no **IOCTL_IR_SET_WAKE_PATTERN** has ever been received by the port driver, the hardware needs to default to all addresses. If the hardware stores the address in volatile memory, the port driver needs to persist the address value across reboots and reprogram the hardware as necessary.

** The **Flag** field in a Quatro Pulse payload is specified to change based on the number of high bits in the payload. A complete implementation would require the hardware to adjust the interpretation of mark/space timing according to the value of the **Flag** field. Because of the specific payloads that are used, your hardware does not need to be concerned about the **Flag** value ever being set to "Reversed".

Wake Programmability Options

The following options are available for building your hardware:

- One protocol, one button only, not programmable (version 1 DDI or version 2 DDI). In this option, your hardware is basically hard-wired to respond to a single wake pattern. Note that this option is not allowed in Windows 7. In future Windows Logo program detail, this option will eventually be eliminated in its entirety. If you do this with the version 2 DDI, you should make sure that the **V2_DEV_CAPS_PROGRAMMABLE_WAKE** bit is not set.
- All protocols, all buttons, programmable (version 2 DDI only). In this option, you support a programmable wake pattern, but your hardware can only respond to a single wake pattern. If you do this, be sure to set the **V2_DEV_CAPS_PROGRAMMABLE_WAKE** bit and be sure to indicate which protocols your hardware supports by setting the appropriate bits in the **IR_DEV_CAPS_V2.WakeProtocols** variable.
- This second option is the Microsoft recommended method.
- All protocols, all buttons, simultaneous decoding (version 1 DDI or version 2 DDI). In this option, you must program your hardware to respond to all valid wake patterns simultaneously. If you do this, be sure to set the **V2_DEV_CAPS_MULTIPLE_WAKE** bit. Note that if you choose this option, your device will still need to be programmable because the **Address** field can change and you must filter wake patterns based on the **Address** field.

IOCTL Definitions

This section describes the specific IOCTLs that the **CIRClass** driver uses to communicate with CIR Port drivers. Note that support of all IOCTLs is mandatory, unless a given IOCTL specifically indicates that its implementation is optional.

IOCTL_IR_ENTER_PRIORITY_RECEIVE

User Scenario

This puts the IR receiver into learning mode for parse and match as well as for learning of individual buttons (IR learning short range).

Operation

This request is sent to prepare the port to enter Priority Receive mode. While the device is in Priority Receive mode, all **IOCTL_IR_RECEIVE** requests should be starved and **IOCTL_IR_PRIORITY_RECEIVE** requests should be completed.

Input

Irp->AssociatedIrp.SystemBuffer points to an **IR_ENTER_PRIORITY_RECEIVE_PARAMS** structure containing parameters for the Priority Receive operation.

Output

None.

I/O Status Block

The **Information** field is set to zero.

The **Status** field is set to **STATUS_SUCCESS** or an appropriate error status.

For more information, see IR_ENTER_PRIORITY_RECEIVE_PARAMS,
IOCTL_IR_EXIT_PRIORITY_RECEIVE, and IOCTL_IR_PRIORITY_RECEIVE.

IOCTL_IR_EXIT_PRIORITY_RECEIVE

User Scenario

This is used to transfer the device from learning mode back into basic receive mode.

Operation

This request is sent to end Priority Receive mode. Upon receipt of the request, the port should abort any outstanding **IOCTL_IR_PRIORITY_RECEIVE** requests and fail any future **IOCTL_IR_PRIORITY_RECEIVE** requests (before receiving a new **IOCTL_IR_ENTER_PRIORITY_RECEIVE** request). As a result of receiving this IOCTL, the CIR Port driver is responsible for restoring the device to the state that it was in before receipt of the **IOCTL_IR_ENTER_PRIORITY_RECEIVE**.

Input

None.

Output

None.

I/O Status Block

The **Information** field is set to zero.

The **Status** field is set to **STATUS_SUCCESS** or an appropriate error status.

For more information, see **IOCTL_IR_ENTER_PRIORITY_RECEIVE** and **IOCTL_IR_PRIORITY_RECEIVE**.

IOCTL_IR_FLASH_RECEIVER

User Scenario

This is used to give the user a visible indication of where to point the receiver.

Operation

Flash an LED on the given receiver. Used to tell the user where to point the remote, so a given "receiver box" with multiple receiver parts only needs one LED to flash.

Important This is highly recommended as a key user scenario.

Input

Irp->AssociatedIrp.SystemBuffer contains a pointer to a 32-bit bitmask of receivers to flash.

Output

None.

I/O Status Block

The **Information** field is set to zero.

The **Status** field is set to **STATUS_SUCCESS** if the operation is completed successfully.

IOCTL_IR_GET_DEV_CAPS

User Scenario

Used during driver load and in first run to determine what hardware is connected to the system and if that hardware supports the functionality the user is trying to configure.

Operation

Returns device capabilities in the **IR_DEV_CAPS_V1** or **IR_DEV_CAPS_V2** structure.

The port driver should examine the size of the output buffer. If the output buffer is big enough to hold an **IR_DEV_CAPS_V2** structure, the port driver should fill in the **IR_DEV_CAPS_V2** values, set **ProtocolVersion** to **DEV_CAPS_PROTOCOL_VERSION_V2**, and set the **Information** field for **sizeof(IR_DEV_CAPS_V2)**. If the buffer is only big enough to hold an **IR_DEV_CAPS_V1** structure, the port driver should fill in the **IR_DEV_CAPS_V1** values, set **ProtocolVersion** to **DEV_CAPS_PROTOCOL_VERSION_V1**, and set the **Information** field to **sizeof(IR_DEV_CAPS_V1)**.

Input

None.

Output

Irp->AssociatedIrp.SystemBuffer points to an **IR_DEV_CAPS_V1** or **IR_DEV_CAPS_V2** structure to be filled in by the port driver.

I/O Status Block

The **Information** field is set to **sizeof(IR_DEV_CAPS_V1)** or **IR_DEV_CAPS_V2** if successful. The value of this result depends on the size of the buffer passed as indicated above.

The **Status** field is set to **STATUS_SUCCESS** if the operation is completed successfully. It may also be the following value:

STATUS_BUFFER_TOO_SMALL - The supplied output buffer is too small to be an **IR_DEV_CAPS_V1** or **IR_DEV_CAPS_V2** structure.

For more information, see **IR_DEV_CAPS_V1** and **IR_DEV_CAPS_V2**.

IOCTL_IR_GET_EMITTERS

User Scenario

Used in first run to identify how many emitters are connected to the system and which ports have emitters connected.

Operation

Gets attached emitters and returns the information in a bitmask. This needs to return timely information. It is expected that the user will plug and unplug emitters during the operation of the PC, so it is necessary to query the hardware at the time of the call.

Input

None.

Output

Irp->AssociatedIrp.SystemBuffer. Contains a pointer to a 32-bit value to be filled in with a bitmask representing the attached emitters.

I/O Status Block

The **Information** field is set to **sizeof(ULONG)** if the operation is successful.

The **Status** field is set to **STATUS_SUCCESS** if the operation is completed successfully. It may also be the following value:

STATUS_BUFFER_TOO_SMALL - The supplied output buffer is too small to be a ULONG.

IOCTL_IR_HANDSHAKE

User Scenario

User plugs device into computer.

Operation

This IOCTL is sent from **CIRClass** before creating the HID child device to represent the port. This IOCTL is to be completed synchronously by the port as an indication that it is prepared to return RLC IR data to the class driver.

Input

None.

Output

None.

I/O Status Block

The **Information** field is set to zero.

The **Status** field is set to **STATUS_SUCCESS**.

IOCTL_IR_PRIORITY_RECEIVE

User Scenario

Data sent from learned remote control to the class driver.

Operation

This request is sent from **CIRClass** and receives Run Length Coded (RLC) IR data when the device is running in Priority Receive mode. If the device is not already in Priority Receive mode, initiated by having previously received an **IOCTL_ENTER_PRIORITY_RECEIVE**, the CIR Port driver fails this request immediately. If in Priority Receive mode, the request will remain pending until one of two events occurs:

- The data buffer provided in the request has been completely filled with data.
- An IR timeout occurs. The length of time required for the IR timeout was specified when entering Priority Receive mode.

While in Priority Receive mode and processing **IOCTL_IR_PRIORITY_RECEIVE** requests, **IOCTL_IR_RECEIVE** requests remain pending and are not filled with IR data.

Input

None.

Output

Irp->MdlAddress contains a variable length IR_PRIORITY_RECEIVE_PARAMS structure.

I/O Status Block

The **Information** field is set to the actual number of bytes copied into the supplied data buffer, including the **IR_PRIORITY_RECEIVE_PARAMS** structure.

The **Status** field is set to **STATUS_SUCCESS** if the operation is completed successfully. It may also be one of the following values:

STATUS_INVALID_DEVICE_STATE - The device is not in Priority Receive mode.

STATUS_BUFFER_TOO_SMALL - The supplied output buffer is too small to be an **IR_PRIORITY_RECEIVE_PARAMS** structure.

STATUS_INVALID_BUFFER_SIZE – The ByteCount field of the **IR_PRIORITY_RECEIVE_PARAMS** structure is larger than the output buffer size specified in the request.

For more information, see **IR_PRIORITY_RECEIVE_PARAMS**, **IOCTL_IR_ENTER_PRIORITY_RECEIVE**, **IOCTL_IR_EXIT_PRIORITY_RECEIVE**, and **IOCTL_IR_RECEIVE**.

IOCTL_IR_RECEIVE

User Scenario

Basic remote commands coming in from a remote control.

Operation

This request is sent from **CIRClass** and receives Run Length Coded (RLC) IR data when the device is not running in Priority Receive mode. When running in Priority Receive mode, these requests remain queued but receive no data.

An **IOCTL_IR_RECEIVE** request remains pending until one of two events occurs:

- The data buffer provided in the request has been completely filled with RLC IR data.
- An IR timeout occurs. In the case of an IR timeout, the DataEnd member of the output structure is set to TRUE. The default timeout is 100 milliseconds.

Input

None.

Output

Irp->MdiAddress contains a variable length **IR_RECEIVE_PARAMS** structure.

I/O Status Block

The **Information** field is set to the actual number of bytes copied into the supplied data buffer, including the **IR_RECEIVE_PARAMS** structure.

The **Status** field is set to **STATUS_SUCCESS** if the operation is completed successfully. It may also be one of the following values:

STATUS_BUFFER_TOO_SMALL - The supplied output buffer is too small to be an **IR_RECEIVE_PARAMS** structure.

STATUS_INVALID_BUFFER_SIZE – The ByteCount field of the **IR_RECEIVE_PARAMS** structure is larger than the output buffer size specified in the request.

For more information, see **IR_RECEIVE_PARAMS** and **IOCTL_IR_PRIORITY_RECEIVE**.

IOCTL_IR_RESET_DEVICE

User Scenario

Not documented in this release.

Operation

Resets the given device. When a device is reset, all pending transmit and receive IOCTLs are canceled by the port driver. Additionally, the power driver should re-initialize the hardware to the default state.

Input

None.

Output

None.

I/O Status Block

The **Information** field is set to zero.

The **Status** field is set to **STATUS_SUCCESS** if the operation is successful. May also return an appropriate error if communication with the device fails.

IOCTL_IR_TRANSMIT

User Scenario

Used to send IR data to control a set-top box.

Operation

Transmits the given IR stream on the given ports at the given carrier frequency. This IOCTL is synchronous. It does not return until the IR has actually been transmitted.

Input

Irp->AssociatedIrp.SystemBuffer points to an **IR_TRANSMIT_PARAMS** structure, describing the state the device should be in while the data is transmitted.

Irp->MdlAddress contains a variable length **IR_TRANSMIT_CHUNK** structure.

Output

None.

I/O Status Block

The **Information** field is set to the total number of bytes written.

The **Status** field is set to **STATUS_SUCCESS** if the operation is completed successfully. It may also be the following value:

STATUS_INVALID_PARAMETER - The carrier period in the parameters structure is zero, or some other parameter is incorrect.

For more information, see **IR_TRANSMIT_PARAMS** and **IR_TRANSMIT_CHUNK**.

IOCTL_IR_USER_CLOSE

User Scenario

No user scenario. Part of basic device communication.

Operation

This IOCTL is sent from IRCLASS when a user has indirectly closed the port driver. This IOCTL is informational only, allowing the port to do any cleanup required when closed by a user.

Input

None.

Output

None.

I/O Status Block

The **Information** field is set to zero.

The **Status** field is set to **STATUS_SUCCESS**.

For more information, see **IOCTL_IR_USER_OPEN**.

IOCTL_IR_USER_OPEN

User Scenario

No user scenario. Part of basic device communication.

Operation

This IOCTL is sent from the class driver when a user has indirectly opened the port driver through IRCLASS. This IOCTL is informational only, allowing the port to do any initialization or bookkeeping required to handle requests not directly originating from IRCLASS.

Input

None.

Output

None.

I/O Status Block

The **Information** field is set to zero.

The **Status** field is set to **STATUS_SUCCESS**.

For more information, see **IOCTL_IR_USER_CLOSE**.

IOCTL_IR_SET_WAKE_PATTERN (Version 2 Only)

User Scenario

No user scenario. The device is being configured to wake on a specific IR pattern. This may happen during driver initialization or in response to a specific user key press.

Operation

This IOCTL is sent from IRCLASS to program the hardware to respond to a new wake pattern. The port driver should program the hardware to respond to this wake pattern. The port driver should not complete this IRP until it is done programming the hardware. A delay in completing this IRP may cause a delay in perceived system-shutdown time.

Input

Irp->AssociatedIrp.SystemBuffer points to an **IR_SET_WAKE_PATTERN_PARAMS** structure, describing the wake pattern that the hardware should respond to.

Output

None.

I/O Status Block

The **Status** field is set to **STATUS_SUCCESS** if the operation is completed successfully. It may also be one of the following values:

Data Structure Definitions

The following section defines all of the data structures used by the **CIRClass** and CIR Port drivers in their IOCTL communications.

STATUS_BUFFER_TOO_SMALL: The supplied input buffer is too small to contain an **IR_SET_WAKE_PATTERN_PARAMS** structure.

STATUS_NOT_SUPPORTED: The hardware does not support programmable wake, or the hardware does not support the specified protocol or button code.

IR_ENTER_PRIORITY_RECEIVE_PARAMS

The **IR_ENTER_PRIORITY_RECEIVE_MODE_PARAMS** structure is used in conjunction with the **IOCTL_IR_ENTER_PRIORITY_RECEIVE_IOCTL** to put the device into Priority Receive mode.

Syntax

```
typedef struct _IOCTL_IR_ENTER_PRIORITY_RECEIVE_PARAMS {  
    IN ULONG_PTR Receiver;  
    IN ULONG_PTR Timeout;  
}IOCTL_IR_ENTER_PRIORITY_RECEIVE_PARAMS, *PIOCTL_IR_ENTER_PRIORITY_RECEIVE_PARAMS;
```

Members

Receiver

Index of the receiver to use.

Timeout

Timeout value, in milliseconds. Used to define the end of a sample.

Headers

For more information, see **IOCTL_IR_ENTER_PRIORITY_RECEIVE**.

IR_DEV_CAPS

The **IR_DEV_CAPS** structure is used in conjunction with the **IOCTL_IR_GET_DEV_CAPS_IOCTL** to retrieve capability information of the device. The **IR_DEV_CAPS** structure is used by the version 1 DDI and is also known as the **IR_DEV_CAPS_V1** structure. Port drivers written to the version 2 DDI should use the **IR_DEV_CAPS_V2** structure if enough buffer is presented by the class driver. See **IOCTL_IR_GET_DEV_CAPS** for more information.

Syntax

```
typedef struct _IR_DEV_CAPS {  
    OUT ULONG_PTR ProtocolVersion;  
    OUT ULONG_PTR NumTransmitPorts;  
    OUT ULONG_PTR NumReceivePorts;  
    OUT ULONG_PTR LearningReceiverMask;  
    OUT ULONG_PTR DevCapsFlags;  
}IR_DEV_CAPS, *PIR_DEV_CAPS;
```

Members

ProtocolVersion

Protocol version. If using this structure, this must be **DEV_CAPS_PROTOCOL_VERSION_V1** (0x100).

NumTransmitPorts

Number of transmit ports: 0-32.

NumReceivePorts

Number of receive ports: 0-2 Typically, this number will either be 1 (for a receive only device with a long-range receiver) or 2 (for a transmit/receive device with a long-range receiver and a wide-band receiver).

LearningReceiverMask

Bitmask identifying which receivers are learning receivers.

DevCapsFlags

A combination of one or more of the following values:

- **DEV_CAPS_HAS_UNIQUE_SERIAL**: Unused
- **DEV_CAPS_CAN_FLASH_RECEIVER_LED**: Device supports the **IOCTL_IR_FLASH_RECEIVER** IRP

Headers

For more information, see **IOCTL_IR_GET_DEV_CAPS**.

IR_DEV_CAPS_V1

Alias for **IR_DEV_CAPS**. See **IR_DEV_CAPS**.

IR_DEV_CAPS_V2 (Version 2 Only)

The **IR_DEV_CAPS_V2** structure is used in conjunction with the **IOCTL_IR_GET_DEV_CAPS** IOCTL to retrieve capability information of the device. Port drivers written to the version 2 DDI should use the **IR_DEV_CAPS_V2** structure if enough buffer is presented by the class driver. If the class driver does not provide enough buffer for an **IR_DEV_CAPS_V2** structure, the port driver should treat the buffer as an **IR_DEV_CAPS** structure and fill it in accordingly. See **IOCTL_IR_GET_DEV_CAPS** for more information.

Syntax

```
typedef struct _IR_DEV_CAPS_V2 {  
    OUT ULONG_PTR ProtocolVersion;  
    OUT ULONG_PTR NumTransmitPorts;  
    OUT ULONG_PTR NumReceivePorts;  
    OUT ULONG_PTR LearningReceiverMask;  
    OUT ULONG_PTR DevCapsFlags;  
    OUT ULONG_PTR WakeProtocols;  
    OUT WCHAR TunerPnpId[MAXIMUM_FILENAME_LENGTH] ;  
}IR_DEV_CAPS_V2, *PIR_DEV_CAPS_V2;
```

Members

ProtocolVersion

Protocol version. If using this structure, must be **DEV_CAPS_PROTOCOL_VERSION_V2** (0x200).

NumTransmitPorts

Number of transmit ports: 0-32.

NumReceivePorts

Number of receive ports: 0-2 This number includes long-range receivers and learning receivers. Will typically be 0, 1, or 2, depending on the number of receivers present on the device.

LearningReceiverMask

Bitmask identifying which receivers are learning receivers. Set to zero (0) if no learning receivers are on the device.

Flags

A combination of one or more of the following values:

- **DEV_CAPS_HAS_UNIQUE_SERIAL**: Unused.
- **DEV_CAPS_CAN_FLASH_RECEIVER_LED**: Device supports the **IOCTL_IR_FLASH_RECEIVER** IRP.
- **V2_DEV_CAPS_SUPPORTS_WAKE**: Device supports wake from sleep.
- **V2_DEV_CAPS_MULTIPLE_WAKE**: Device supports all wake from all keys simultaneously.
- **V2_DEV_CAPS_PROGRAMMABLE_WAKE**: Device supports wake pattern programming to set the wake button code.
- **V2_DEV_CAPS_VOLATILE_WAKE_PATTERN**: Wake pattern programming is stored in volatile storage. The device must refresh the wake pattern on each re-initialization. For example, this bit would be set if the wake pattern is stored in RAM and clear if the wake pattern is stored in flash memory.
- **V2_DEV_CAPS_LEARNING_ONLY**: The device supports IR learning, but not IR input, and not parse-and-match (no long-range receiver).
- **V2_DEV_CAPS_NARROW_BPF**: Long-range receiver on the device has a narrow band pass filter (BPF), so parse-and-match operation is not possible with this device.
- **V2_DEV_CAPS_NO_SWDECODE_INPUT**: The device is not meant as an IR input device. IR input into this device should not generate keystrokes for the user application.
- **V2_DEV_CAPS_HWDECODE_INPUT**: The device has additional “hardware decoded” IR-input functionality. This would be set, for instance, if the device was a composite device that also supports input using a hardware-decoded, non-standard IR protocol.
- **V2_DEV_CAPS_ATTACHED_TO_TUNER**: The device is attached to a tuner. This would be set for IR receivers/blasters that are integrated into a USB or PCI tuner module.

WakeProtocols

A list of wake protocols that this hardware supports. A combination of one or more of the following values:

- **V2_WAKE_PROTOCOL_RC6**: The device supports wake from RC6 protocol.
- **V2_WAKE_PROTOCOL_QP**: The device supports wake from Quatro Pulse protocol.

TunerPnpId

If this device is attached to a tuner, and the driver knows the PNP ID of the tuner, this field should contain the PNP ID of the tuner.

Headers

For more information, see **IOCTL_IR_GET_DEV_CAPS**.

IR_PRIORITY_RECEIVE_PARAMS

The **IR_PRIORITY_RECEIVE_PARAMS** structure is used in conjunction with the **IOCTL_IR_PRIORITY_RECEIVE_IOCTL** to read IR data from a device.

Syntax

```
typedef struct _IR_PRIORITY_RECEIVE_PARAMS {  
    OUT ULONG_PTR DataEnd;  
    IN  ULONG_PTR ByteCount;  
    OUT ULONG_PTR CarrierFrequency;  
    IN  LONG      Data[1];  
}IR_PRIORITY_RECEIVE_PARAMS, *PIR_PRIORITY_RECEIVE_PARAMS;
```

Members

DataEnd

Set by port driver to TRUE if a DataEnd (Timeout) event occurred. Otherwise, FALSE.

ByteCount

Set by caller to indicate the number of bytes in the variable length **Data[]** portion. When completing the IRP, the port driver sets this to the number of bytes that were filled in by the port driver.

CarrierFrequency

Set by port driver to indicate carrier frequency of IR sample (if available).

Data

First byte in the **ByteCount** data buffer.

Headers

For more information, see **IOCTL_IR_PRIORITY_RECEIVE**.

IR_RECEIVE_PARAMS

The **IR_RECEIVE_PARAMS** structure is used in conjunction with the **IOCTL_IR_RECEIVE_IOCTL** to read RLC IR data from a device.

Syntax

```
typedef struct _IR_RECEIVE_PARAMS {  
    OUT ULONG_PTR DataEnd;  
    IN  ULONG_PTR ByteCount;  
    OUT LONG      Data[1];  
}IR_RECEIVE_PARAMS, *PIR_RECEIVE_PARAMS;
```

Members

DataEnd

Upon completion of the IOCTL, indicates whether or not this data buffer was completed because of an IR timeout, or "data end", Set to TRUE if it was completed because of a timeout, FALSE otherwise.

ByteCount

Set by caller to indicate the number of bytes in the variable length **Data[]** portion.

Data

First byte in the **ByteCount** data buffer.

Headers

For more information, see **IOCTL_IR_RECEIVE**.

IR_SET_WAKE_PATTERN_PARAMS (Version 2 Only)

The **IR_SET_WAKE_PATTERN_PARAMS** structure is used in conjunction with the **IOCTL_IR_SET_WAKE_PATTERN** IOCTL to program the device to wake on a specific IR pattern.

Syntax

```
typedef struct _IOCTL_IR_SET_WAKE_PATTERN_PARAMS {  
    IN ULONG Protocol;  
    IN ULONG Payload;  
    IN ULONG Address;  
} IR_SET_WAKE_PATTERN_PARAMS, *PIR_SET_WAKE_PATTERN_PARAMS;
```

Members

Protocol

Protocol of the Wake button. Set to one of the following values:

- **V2_WAKE_PROTOCOL_RC6**: The Wake button uses the RC6 protocol.
- **V2_WAKE_PROTOCOL_QP**: The Wake button uses the Quatro Pulse protocol.

Payload

Button code for the Wake button. Set to one of the following values:

- **WAKE_KEY_POWER_TOGGLE**: The wake on the Sleep toggle button.
- **WAKE_KEY_DISCRETE_ON**: The wake on the Discrete On button.

Address

The remote control address to wake on.

Headers

For more information, see **IOCTL_IR_SET_WAKE_PATTERN**.

IR_TRANSMIT_PARAMS

The **IR_TRANSMIT_PARAMS** structure is used in conjunction with the **IOCTL_IR_TRANSMIT_IOCTL** and **IR_TRANSMIT_CHUNK** structure to blast IR data. It describes the device parameters to use for the blasting.

Syntax

```
typedef struct _IR_TRANSMIT_PARAMS {  
    IN ULONG_PTR TransmitPortMask;  
    IN ULONG_PTR CarrierPeriod;  
    IN ULONG_PTR Flags;  
    IN ULONG_PTR PulseSize;  
} IR_TRANSMIT_PARAMS, *PIR_TRANSMIT_PARAMS;
```

Members

TransmitPortMask

Bitmask containing ports to transmit on.

CarrierPeriod

Carrier period to use.

Flags

Currently unused.

PulseSize

Currently unused.

Headers

For more information, see IOCTL_IR_TRANSMIT and IR_TRANSMIT_CHUNK.

IR_TRANSMIT_CHUNK

The **IR_TRANSMIT_CHUNK** structure is used in conjunction with the **IOCTL_IR_TRANSMIT_IOCTL** and **IR_TRANSMIT_PARAMS** structure to blast IR data. It describes the IR data to be blasted by the device:

Syntax

```
typedef struct _IR_TRANSMIT_CHUNK {  
    ULONG_PTR    OffsetToNextChunk;  
    ULONG_PTR    RepeatCount;  
    ULONG_PTR    ByteCount;  
    LONG        Data[1];  
} IR_TRANSMIT_CHUNK, *PIR_TRANSMIT_CHUNK;
```

Members

OffsetToNextChunk

Offset, in bytes, from **Data** member of this buffer to next **IR_TRANSMIT_CHUNK** (or zero if no more chunks in buffer.)

RepeatCount

Number of times to serially repeat **ByteCount** bytes of data.

ByteCount

Number of data bytes contained in **Data[]**.

Data

First byte of **ByteCount** bytes of data.

Note Each chunk is filled to integral ULONG_PTR boundary.

Headers

For more information, see IOCTL_IR_TRANSMIT and IR_TRANSMIT_PARAMS.

HID Device Requirements

The Windows Media Center user interface is designed to be used with a remote control by a user who is sitting up to five meters away from a computer running a Windows operating system. The remote control receiver is used to process commands that control and navigate the Windows Media Center user interface.

In Microsoft Windows® XP Media Center Edition, the only supported remote control implementation is an infrared (IR) device that is based on reference designs provided by Microsoft. In an effort to give IHVs and OEMs more options for building remote control solutions, Microsoft is providing these additional specifications for delivering Human Interface Device (HID)-based remote control receivers (input only) that will work with Windows Media Center. Using these specifications, any medium can be used to create HID-based PC remote control receivers (for example, IR, RF, and so on). The considerations and restrictions for IR devices are described in this section.

If you need to build a transmit/receive device, you must build a legacy device, a port driver device, or an emulator device.

HID Remote Control Receiver Requirements

This section contains requirements for remote control receivers, system-level interaction, and triple-tap input.

Remote Control Receiver (Input Only)

The following list provides remote control and long-range receiver requirements.

- The receiver will decode input from a matching remote control device in the hardware and issue Human Interface Device (HID) usage reports based on the HID usage code table in "HID Table," later in this document.
- The receiver must resume from standby mode using the Sleep button for the particular IR protocol for which the hardware is optimized. Resume-from-standby must do hardware decoding of the protocol and operate when the Windows Media Center computer is in a state of lower power consumption.
- The receiver must wake from S1 or S3. Resuming or waking from S4 or S5 is optional.
- The receiver must properly indicate user presence to the operating system when waking the system. This can be tested by first waking the system with the remote control, and then by running a scheduled task. The monitor should turn on when waking with the remote control, but not when waking from running a scheduled task.
- If using a USB device, it is recommended that the device be able to operate correctly when it is plugged into a passive hub.

Handling Numeric Input from a Remote Control

When using a remote control to enter numbers and text in Windows XP Media Center Edition, or Windows Media Center in Windows Vista, users must be able to "triple-tap" letters. For example, a user can type a song title by pressing the "2" key once to get an A, twice to get a B, and three times to get a C. This functionality is called triple-tap because a user taps a key as many as three times (and sometimes more).

In Windows 7 the "triple tap" is replaced by an onscreen keyboard. In order for numeric entry to work worldwide, the same registry entry below needs to be implemented.

To enable triple-tap input and accurate worldwide numeric input from a remote control in Windows Media Center text fields, the receiver must be registered in the Windows registry as a supported Windows Media Center remote control. When the receiver is properly registered, Windows Media Center processes input correctly.

The key must be set on any consumer system that uses the receiver, either by having OEMs apply the key to their image or by providing an end-user setup on a CD-ROM. The following example illustrates the registry key to be set.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Media Center\Remote Controls]
"HID\VID_045E&PID_006D"=""
```

In this example, the VID value is 045E and the PID value is 006D.

Notes The VID and PID values in this registry key are for your specific receiver. Each different receiver model should have its own VID and PID values.

The registry key string (HID\VID_XXXX&PID_XXXX "=") is the Plug and Play (PNP) ID for your specific device. If your device is not based on the USB bus, then the PNP ID will likely be in a different format.

On 64bit versions of the operating system, you must also add this registry key to [HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Media Center\Remote Controls], otherwise your device may produce multiple input events for 32-bit extensibility apps running on the 64-bit OS.

IR Receiver Considerations

All Windows Media Center IR receivers (input only) must be certified to meet the Windows Hardware Logo Program Hardware Logo Program requirements.

IR receivers (input only) that use hardware decoding must prevent input duplication and conflicts with other Windows Media Center IR receiver devices. To avoid such conflicts, IR receivers that use hardware decoding are prohibited from using Microsoft-reserved IR protocols.

HID Usage Codes

This section contains information about support for Human Interface Device (HID) usage codes.

Future Considerations

Microsoft will continue to evaluate the use of new HID usage code standards as they are made available from the USB Implementers' Forum. Over time, future versions of the Windows Media Center operating system will implement the use of these standardized HID pages in place of the current vendor-specific HID page 0xFFBC. Windows Media Center specifications will be updated to reflect these changes as soon as they are established. Windows Media Center will maintain support for current HID usage codes for backward compatibility, but reserves the right to phase them out in future versions of Windows Media Center.

Blu-ray HID Usage Codes Support

Currently there is no specific remote control support for Blu-ray controls. Future versions of Windows Media Center may contain support and documentation for using the enhanced functionality of these technologies by using a remote control.

HID Descriptor

The following is the Recommended HID Report Descriptor used for producing HID events.

```
\ ; Consumer Controls
  0x05,  0x0c,  \ ; Usage Page (Consumer Controls),
  0x09,  0x01,  \ ; Usage (Consumer Control),
  0xA1,  0x01,  \ ; Collection (Application),
  0x85,  0x01,  \ ; Report Id (1)
  0x19,  0x00,  \ ; Usage Minimum (0),
  0x2a,  0x3c, 0x02, \ ; Usage Maximum (23c)
  0x15,  0x00,  \ ; Logical Minimum (0),
  0x26,  0x3c, 0x02, \ ; Logical Maximum (23c)
```

```

0x95, 0x01, \ ; Report Count (1),
0x75, 0x10, \ ; Report Size (16),
0x81, 0x00, \ ; Input (Data, Array),
0xC0, \ ; End Collection
\ ; MS Vendor controls
0x06, 0xbc, 0xff, \ ; Usage Page (Vendor 0xffbc),
0x09, 0x88, \ ; Usage (88),
0xa1, 0x01, \ ; Collection (Application),
0x85, 0x02, \ ; Report Id (2)
0x19, 0x01, \ ; Usage Minimum (0x01),
0x29, 0xff, \ ; Usage Maximum (0xff),
0x15, 0x00, \ ; Logical Minimum (0),
0x25, 0x01, \ ; Logical Maximum(1),
0x95, 0x01, \ ; Report Count (1),
0x75, 0x08, \ ; Report Size (8),
0x81, 0x00, \ ; Input (Data, Array),
0xc0, \ ; End Collection
\ ; Standby button
0x05, 0x01, \ ; Usage Page (Generic Desktop),
0x09, 0x80, \ ; Usage (System Control),
0xa1, 0x01, \ ; Collection (Application),
0x85, 0x03, \ ; Report Id (3)
0x19, 0x81, \ ; Usage Minimum (0x81),
0x29, 0x83, \ ; Usage Maximum (0x83),
0x25, 0x01, \ ; Logical Maximum(1),
0x75, 0x01, \ ; Report Size (1),
0x95, 0x03, \ ; Report Count (3),
0x81, 0x02, \ ; Input
0x75, 0x01, \ ; Report Size (1),
0x95, 0x05, \ ; Report Count (5),
0x81, 0x01, \ ; Input (Constant),
0xC0, \ ; End Collection
\ ; Keyboard
0x05, 0x01, \ ; Usage Page (Generic Desktop),
0x09, 0x06, \ ; Usage (Keyboard),
0xA1, 0x01, \ ; Collection (Application),
0x85, 0x04, \ ; Report Id (4)
0x05, 0x07, \ ; usage page key codes
0x19, 0xe0, \ ; usage min left control
0x29, 0xe8, \ ; usage max keyboard right gui
0x75, 0x01, \ ; report size 1
0x95, 0x08, \ ; report count 8
0x81, 0x02, \ ; input (Variable)
0x19, 0x00, \ ; usage min 0
0x29, 0x90, \ ; usage max 91
0x26, 0xff, 0x00, \ ; logical max 0xff
0x75, 0x08, \ ; report size 8
0x95, 0x01, \ ; report count 1
0x81, 0x00, \ ; Input (Data, Array),
0xC0 \ ; End Collection

```

HID Table

The following table describes the remote control HID usages that Windows Media Center will respond to. Remote controls that are used to control Windows Media Center must meet the Microsoft remote control specifications.

Remote Control and Receiver-Transceiver Specifications and Requirements
for Windows Media Center in Windows Operating Systems

Button Grouping	Name	HID TLC Page	HID TLC Usage	HID Button Page	HID Button Usage
Navigation	Green Start E	0xFFBC	0x88	0xFFBC	0x0D
Navigation	Back	0x0C	0x01	0x0C	0x0224
Navigation	More Info	0x0C	0x01	0x0C	0x0209
Navigation	Up	0x01	0x06	0x07	0x52
Navigation	Down	0x01	0x06	0x07	0x51
Navigation	Left	0x01	0x06	0x07	0x50
Navigation	Right	0x01	0x06	0x07	0x4F
Navigation	OK	0x01	0x06	0x07	0x28
Transport	Play	0x0C	0x01	0x0C	0xB0
Transport	Pause	0x0C	0x01	0x0C	0xB1
Transport	Play/Pause C	0x0C	0x01	0x0C	0xCD
Transport	Stop	0x0C	0x01	0x0C	0xB7
Transport	Record	0x0C	0x01	0x0C	0xB2
Transport	Fast Forward	0x0C	0x01	0x0C	0xB3
Transport	Rewind	0x0C	0x01	0x0C	0xB4
Transport	Skip Forward	0x0C	0x01	0x0C	0xB5
Transport	Skip Back	0x0C	0x01	0x0C	0xB6
AV Control Powe	Volume Up	0x0C	0x01	0x0C	0xE9
AV Control Powe	Volume Dowr	0x0C	0x01	0x0C	0xEA

Remote Control and Receiver-Transceiver Specifications and Requirements
for Windows Media Center in Windows Operating Systems

Button Grouping	Name	HID TLC Page	HID TLC Usage	HID Button Page	HID Button Usage
AV Control Powe	Mute	0x0C	0x01	0x0C	0xE2
AV Control Powe	Closed Captic	0xFFBC	0x88	0xFFBC	0x2B
AV Control Powe	Chan/Page U	0x0C	0x01	0x0C	0x9C
AV Control Powe	Chan/Page D	0x0C	0x01	0x0C	0c9D
AV Control Powe	Sleep toggle	0x01	0x80	0x01	0x82
AV Control Powe	Wake	0x01	0x80	0x01	0x83
AV Control Powe	Sleep	0x01	0x80	0x01	0x82
Numeric Keypad	0	0x01	0x06	0x07	0x27
Numeric Keypad	1	0x01	0x06	0x07	0x1E
Numeric Keypad	2	0x01	0x06	0x07	0x1F
Numeric Keypad	3	0x01	0x06	0x07	0x20
Numeric Keypad	4	0x01	0x06	0x07	0x21
Numeric Keypad	5	0x01	0x06	0x07	0x22
Numeric Keypad	6	0x01	0x06	0x07	0x23
Numeric Keypad	7	0x01	0x06	0x07	0x24
Numeric Keypad	8	0x01	0x06	0x07	0x25
Numeric Keypad	9	0x01	0x06	0x07	0x26
Numeric Keypad	Clear	0x01	0x06	0x07	0x29
Numeric Keypad	Enter	0x01	0x06	0x07	0x28

Remote Control and Receiver-Transceiver Specifications and Requirements
for Windows Media Center in Windows Operating Systems

Button Grouping	Name	HID TLC Page	HID TLC Usage	HID Button Page	HID Button Usage
Numeric Keypad	'#'	0x01	0x06	0x07	0x20 + 0xE1
Numeric Keypad	'*'	0x01	0x06	0x07	0x25 + 0xE1
Teletext/ISDB-T	Teletext/"d" C	0xFFBC	0x88	0xFFBC	0x5A
Teletext	Teletext Red	0xFFBC	0x88	0xFFBC	0x5B
Teletext	Teletext Gree	0xFFBC	0x88	0xFFBC	0x5C
Teletext	Teletext Yellc	0xFFBC	0x88	0xFFBC	0x5D
Teletext	Teletext Blue	0xFFBC	0x88	0xFFBC	0x5E
Windows Media (Shortcuts	Guide	0x0C	0x01	0x0C	0x8D
Windows Media (Shortcuts	Live TV	0xFFBC	0x88	0xFFBC	0x25
Windows Media (Shortcuts	Music	0xFFBC	0x88	0xFFBC	0x47
Windows Media (Shortcuts	Recorded TV	0xFFBC	0x88	0xFFBC	0x48
Windows Media (Shortcuts	Pictures	0xFFBC	0x88	0xFFBC	0x49
Windows Media (Shortcuts	Videos	0xFFBC	0x88	0xFFBC	0x4A
Windows Media (Shortcuts	FM Radio	0xFFBC	0x88	0xFFBC	0x50
Windows Media (Shortcuts	Extras	0xFFBC	0x88	0xFFBC	0x3C
Windows Media (Shortcuts	Extras App	0xFFBC	0x88	0xFFBC	0x3D

Button Grouping	Name	HID TLC Page	HID TLC Usage	HID Button Page	HID Button Usage
DVD	DVD Menu	0xFFBC	0x88	0xFFBC	0x24
DVD	DVD Angle	0xFFBC	0x88	0xFFBC	0x4B
DVD	DVD Audio	0xFFBC	0x88	0xFFBC	0x4C
DVD	DVD Subtitle	0xFFBC	0x88	0xFFBC	0x4D
DVD	Eject	0xFFBC	0x88	0xFFBC	0x28
DVD	DVD Top Mei	0xFFBC	0x88	0xFFBC	0x43
Extensibility	Ext0	0xFFBC	0x88	0xFFBC	0x32
Extensibility	Ext1	0xFFBC	0x88	0xFFBC	0x33
Extensibility	Ext2	0xFFBC	0x88	0xFFBC	0x34
Extensibility	Ext3	0xFFBC	0x88	0xFFBC	0x35
Extensibility	Ext4	0xFFBC	0x88	0xFFBC	0x36
Extensibility	Ext5	0xFFBC	0x88	0xFFBC	0x37
Extensibility	Ext6	0xFFBC	0x88	0xFFBC	0x38
Extensibility	Ext7	0xFFBC	0x88	0xFFBC	0x39
Extensibility	Ext8	0xFFBC	0x88	0xFFBC	0x3A
Extensibility	Ext9	0xFFBC	0x88	0xFFBC	0x80
Extensibility	Ext10	0xFFBC	0x88	0xFFBC	0x81
Extensibility	Ext11	0xFFBC	0x88	0xFFBC	0x6F
Other	Print	0x0C	0x01	0x0C	0x0208

Button Grouping	Name	HID TLC Page	HID TLC Usage	HID Button Page	HID Button Usage
Other	Zoom	0xFFBC	0x88	0xFFBC	0x27
Other	Channel Input (Input)	0xFFBC	0x88	0xFFBC	0x42
Other	Sub Audio	0xFFBC	0x88	0xFFBC	0x2D
Other	10	0xFFBC	0x88	0xFFBC	0x3E
Other	11	0xFFBC	0x88	0xFFBC	0x3F
Other	12	0xFFBC	0x88	0xFFBC	0x40

Reserved Button Codes

The following table shows the reserved buttons and corresponding button codes that are defined now to allocate space in the button map and IR stack for use in the future. At this time, functionality for the buttons in this class is not implemented in Windows Media Center. These functions might be implemented in future releases.

No specifications are available that describe Microsoft's intended use of these buttons. Future uses of these buttons by Microsoft might be incompatible with any implementations that are generated before the intended uses are specified.

Button Grouping	Name	HID TLC Page	HID TLC Usage	HID Button Page	HID Button Usage
Reserved	Display	0xFFBC	0x88	0xFFBC	0x4F
Reserved	Kiosk	0xFFBC	0x88	0xFFBC	0x6A
Reserved	Network Sele	0xFFBC	0x88	0xFFBC	0x2C
Reserved	BD Tool	0xFFBC	0x88	0xFFBC	0x78
Reserved	Channel Infor	0xFFBC	0x88	0xFFBC	0x41