



Windows® Embedded Automotive 7



Windows Embedded Automotive 7 Deep Dive: Phone and Media Cores

Abstract

Windows Embedded Automotive 7—based on the newest generation of embedded operating systems from Microsoft, and combining the award-winning Windows Automotive and Microsoft Auto platforms—is designed specifically for developing state-of-the-art, in-vehicle infotainment systems. It offers a standardized, industry-proven platform for building communication, entertainment, and service-enabled location-based solutions.

The Phone Core and Media Core components, part of Windows Embedded Automotive 7, are robust, integrated technologies that allow easy and safe access to phone and media features. Phone Core and Media Core establish a solid foundation for automotive original equipment manufacturers (OEMs) by providing standardized tools and application programming interfaces (APIs) compatible with a wide range of mobile devices. Automotive OEMs can build solutions with the assurance that Microsoft's extensive testing of devices and frequent software updates ensure interoperability with the latest mobile handset and media player technologies.

This white paper discusses the Phone Core and Media Core components of Windows Embedded Automotive 7 in detail and provides specific industry examples of how these technologies are currently being used.

Table of Contents

INTRODUCTION	4
WINDOWS EMBEDDED AUTOMOTIVE IN USE TODAY.....	4
SIMPLIFIED UI DEVELOPMENT WITH SILVERLIGHT FOR WINDOWS EMBEDDED.....	5
DISPLAY	8
<i>Display Driver</i>	8
<i>Native Display API</i>	9
PROJEKT 2 SAMPLE APPLICATION.....	10
PHONE CORE DEEP DIVE.....	11
PHONE CORE ARCHITECTURE OVERVIEW	11
PHONE CONNECTION MANAGEMENT	13
<i>Automatic Connection</i>	14
<i>Manual Connection</i>	14
<i>Disconnection</i>	14
<i>Phone Availability</i>	14
PHONE CORE REGISTRY SETTINGS	14
PHONE CORE WINDOWS MESSAGES	16
BLUETOOTH HARDWARE AND STACK	18
ACOUSTIC ECHO CANCELLATION/NOISE SUPPRESSION.....	20
<i>AEC/NS Configuration Registry Keys</i>	20
SUPPORTED BLUETOOTH PROFILES	21
BLUETOOTH QUALIFICATION	22
BLUETOOTH PAIRING CORE AND SERVICE	22
<i>Bluetooth Pairing Core</i>	22
<i>Bluetooth Pairing Service</i>	24
<i>Discovery Mode and Discoverable Mode</i>	26
<i>Special Pairing Features</i>	26
HANDS-FREE PHONE	26
<i>Hands-free Profile Service</i>	27
<i>HFP Core Service</i>	27
<i>Call Handling</i>	29
<i>Dialing Feature Support</i>	32
<i>Phone Feature Support</i>	32
<i>Phone Core Versus HFP API</i>	33
HIGH-QUALITY RINGTONE	33
PHONE GETLASTERROR.....	34
SYNC MANAGER AND PHONEBOOK	36
<i>Phonebook Text Storage</i>	37
<i>Phonebook Storage Using the Pocket Outlook Object Model</i>	37
<i>Phone Contact Image Auto Compression</i>	39
<i>Phonebook Implementations</i>	40
<i>OBEX Phonebook Stores</i>	40
<i>Sync Manager Architecture</i>	41
SMS SUPPORT AND EMAIL	42

<i>GSM SMS AT Command Support</i>	44
<i>SMS Stack and MAP Service</i>	44
BLUETOOTH AUDIO/VIDEO (BTAV) SERVICE	45
CALENDAR.....	48
INDIVIDUAL PHONE CONFIGURABILITY	49
CONNECTION MANAGER	52
DATA CONNECTIVITY.....	53
<i>Bluetooth Dial-up Networking</i>	53
<i>Bluetooth Gateway Services</i>	54
<i>Personal Area Network</i>	54
SIM ACCESS PROFILE	56
MEDIA CORE DEEP DIVE	57
MEDIA CORE ARCHITECTURE OVERVIEW	57
<i>Media Source Plug-ins</i>	59
SUPPORTED TECHNOLOGIES	60
MEDIA FEATURE ENHANCEMENTS.....	61
BUILDING A MEDIA APPLICATION	62
<i>Sysgen Variables</i>	62
<i>Browsing the Content</i>	62
<i>Playback</i>	63
<i>Album Art</i>	63
<i>Device Tips</i>	63
MEDIA CORE CONFIGURATION	64
MEDIA CORE WINDOWS MESSAGES.....	65
CUSTOM MEDIA DEVICE CLASS.....	68
CUSTOM FILE AND PLAYLIST PARSER.....	68
<i>File Parser</i>	68
<i>Playlist Parser</i>	69
ALBUM ART.....	69
METADATA PLUG-IN API.....	69
IPOD APPLICATION TO ACCESSORY COMMUNICATION	70
IAP AND MTP PASSTHROUGH.....	71
<i>Considerations for iPod</i>	71
<i>Considerations for MTP</i>	72
DEVICE LAB.....	72
CONCLUSION	73
APPENDIX 1: GLOBALIZATION FEATURES	74
APPENDIX 2: BLUETOOTH PAIRING SERVICE REGISTRY KEY VALUES.....	76
APPENDIX 3: MEDIA CORE REGISTRY SETTINGS.....	79
MTP-RELATED REGISTRY KEYS.....	88
<i>Zune</i>	91
APPENDIX 4: COMPATIBLE DEVICES.....	93
GLOSSARY	101



Introduction

With Windows Embedded Automotive 7 application software developers and automotive electronics engineers gain a rich environment from which they can add their own functionality to create a broad range of advanced, in-vehicle solutions that meet the growing needs of consumers while setting the products apart from the rest of the field. The flexible Windows Embedded Automotive 7 platform targets a wide range of devices, including connectivity gateways, connected radios, and multimedia devices.

This white paper takes a closer look at the benefits of developing applications around Phone Core and Media Core, two of the components that form the foundation of the Automotive 7 platform. This paper also explores exciting new user interface (UI) design possibilities using Microsoft Silverlight for Windows Embedded. These components provide standardized tools and APIs, forming a solid foundation for in-car infotainment systems.

Windows Embedded Automotive In Use Today

Several automobile manufacturers and OEMs, including Ford, Kia, and Fiat, have developed and deployed Windows Embedded Automotive applications that take advantage of Phone Core and Media Core. These applications provide in-car services such as hands-free capabilities for Bluetooth-enabled phones, emergency 911 assistance, and voice-activated media playback.

Ford



Ford introduced the Ford SYNC in-car communications and entertainment system in 2007 in Ford, Lincoln, and Mercury vehicles, and has shipped nearly 3 million units. Ford builds upon the Windows Embedded Automotive system—in particular, Phone Core and Media Core—to allow users to operate their Bluetooth-enabled phones via simple voice commands, respond to accidents or emergencies with 911 Assist, and receive text messages, and hear the messages read aloud using a digitized voice.

With Ford SYNC users can also enjoy their music on a wide variety of media devices, and control the devices with voice commands. Ford introduced its second-generation in-car infotainment system, the MyFord Touch, in 2010. MyFord Touch features improved voice recognition, touch-sensitive buttons, touch screens, and thumb-wheel controls to replace the usual knobs and switches. MyFord Touch also boosts Wi-Fi and connectivity with the help of a secure digital (SD) card slot.

Kia



Kia recently introduced Kia UVO, an in-car infotainment system that employs the Windows Embedded Automotive system. Kia UVO builds upon Phone Core and Media Core with advanced features such as voice-activated and touch-activated phone and media commands. Kia UVO includes a full-color in-dash screen, and uses voice commands to help drivers and passengers access music files, operate a rear-view camera, change radio

stations, make or answer phone calls, and more. By supporting complex grammar, UVO needs only short voice commands to connect drivers and passengers with their desired functions. An interactive system, UVO responds to inquiries such as “What’s playing?” and provides audible responses and related functions, allowing drivers to stay safely focused on the road.

Fiat

Fiat Blue&Me empowers customers to connect their personal mobile devices with the integrated solution found in many vehicle models from Fiat, Alfa Romeo, Lancia, Iveco, and Fiat Light Commercial Vehicles. First presented in 2006, Blue&Me was originally developed as an infotainment system capable of allowing mobile phones and MP3 players to be used in the car safely with voice-recognition commands and steering wheel controls.



In 2008, the system evolved further with the introduction of eco:Drive, a free application exclusive to Fiat that helps drivers understand how their driving style can affect consumption and CO2 emissions. Fiat continues to press forward with development of the Blue&Me system, from Blue&Me Nav to TomTom integration, from Nokia Ovi integration to eco:Drive—numerous applications continue to make Fiat Blue&Me an international success.

And More!

Microsoft is also partnering with other automobile manufacturers and original equipment manufacturers (OEMs), such as Chrysler, PACCAR, and Alpine, to bring new Windows Embedded Automotive–powered products to market.

With a proven track record of deployed products, Automotive 7 provides a flexible, mature platform for OEMs, centered on Phone Core and Media Core components.

Simplified UI Development with Silverlight for Windows Embedded

Today’s consumers expect a compelling user experience (UX), but creating these user experiences can consume a lot of time and resources.

Typically, Human-Machine Interface (HMI) design follows these steps:

1. Create the design using Adobe Photoshop and Microsoft PowerPoint.
2. Define the HMI behavior in Microsoft Excel worksheets and Microsoft Word documents.
3. Roll the design and behavior into a rapid prototype (in Adobe Flash, for example).
4. Build the prototypes into an actual HMI.

Essentially, the work is being done twice: once by the designer and again by the developer. This system also adds delays and imposes additional risk. After the developer re-creates the UX code, the back-end code must be created as well (or at least enough of the code must be created to get the HMI running). It is only at this point that the HMI can be reviewed. If there is a problem, the HMI goes back to the developer to be corrected; if the UX designer created an interface that is not practical, the cycle may need to start over from the beginning.

With Microsoft Silverlight for Windows Embedded, a new design/develop paradigm is possible:



1. Design the user experience in Microsoft Expression Blend.
2. Develop the business logic in Microsoft Visual Studio with Silverlight for Windows Embedded.
3. Run the HMI on the embedded device.

This is a significantly easier process. Figure 1 compares current HMI design with that possible with Silverlight for Windows Embedded.



Figure 1: Comparing typical HMI design with Silverlight for Windows Embedded Automotive

A design process based on Microsoft Silverlight for Windows Embedded reduces delays and last-minute design changes. The developer can take the UX that the designer created as is. No changes are needed—the UX can be imported directly into the developer’s Visual Studio development environment. The review process is instant as well—as soon as an HMI is developed, developers can see exactly what the HMI will look like on the device before any additional code is created. This saves time and reduces the risk of identifying UX issues late in the design cycle.

Figure 2 shows the Silverlight for Windows Embedded workflow.

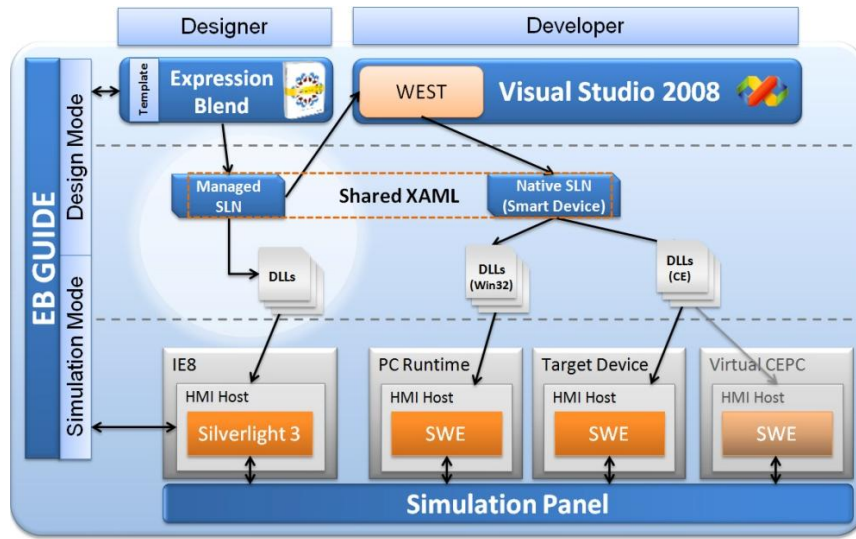


Figure 2: Silverlight for Windows Embedded workflow

Silverlight for Windows Embedded provides a subset of the overall Microsoft Silverlight functionality for embedded devices. Unlike other versions of Silverlight, there is no managed application programming interface (API) and no browser plug-in.

The version of Silverlight for Windows Embedded included with Windows Embedded Automotive 7 extends the version of Silverlight that is available with Windows Embedded Compact with the following key features:

- **Additional behaviors**, or self-contained, reusable snippets of interactivity, that can be applied to any Silverlight UI object. These behaviors can support configuration options that are accessible through the property inspector.
- **Automotive button**, a specialized version of the standard UI.
- **OpenVG support**, support for graphics acceleration using the OpenVG capabilities of the graphics card.

Extensible Application Markup Language (XAML), a markup language for declarative application programming, makes it possible for designers and developers to work simultaneously. As with Silverlight on the desktop, the designer creates the visible UI elements in XAML, and the developer creates the separate code-behind files to respond to events, manipulate the elements declared in the XAML, and control the underlying business logic of the application. Unlike Silverlight on the desktop (which uses managed code like C# or Microsoft Visual Basic), Silverlight for Windows Embedded uses native code (C++) and does not run in a security sandbox. This means that Silverlight has access to any API and resource on the device. However, this also means that desktop Silverlight applications will not run on Windows Embedded CE devices.

To achieve smooth, responsive animations, devices often take advantage of Graphics Processing Unit (GPU) acceleration from their hardware. The automotive version of Silverlight for Windows Embedded provides for hardware-accelerated vector drawing through cached composition. An

OpenVG1.1-based sample render plug-in is provided, and customers can modify it to use any vector graphic APIs (see Figure 3).

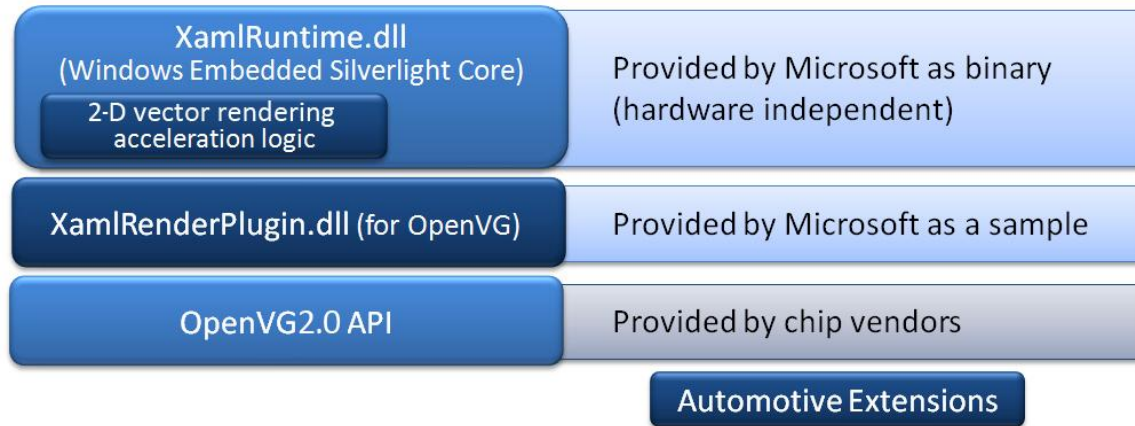


Figure 3: Detail of Silverlight for Windows Embedded

The automotive version of Microsoft Silverlight for Windows Embedded has extra elements that use XAML to compose additional, out-of-process graphics, such as maps and browsers, into an HMI.

Microsoft also provides a default template for an automotive solution, with a reference to the module host binary (shipped with Windows Embedded Automotive 7 or the HMI Toolkit), a bezel proxy and the outline for a media-pseudo application (a managed application that mimics a small subset of the functionality of a real application, used for design purposes and meant to be replaced by a real application when moving to native code).

For further information about the version of Silverlight for Windows Embedded, see the [Microsoft Silverlight for Windows Embedded web page](#).

Display

The ability to display text, buttons, and graphics on the display screen complements speech-based interaction with the user—provided that the display is accomplished with consideration for avoiding driver distraction. The display screen is especially useful for scenarios in which the persistence of visual input has a higher value than auditory input (for example, if a turn announcement is missed, the driver can still look at the display to find out what to do next).

Windows Embedded Automotive 7 can use a display component to work with a remote display, which is typically accessed through the vehicle network. The display service manages the use of the shared display screen by Automotive 7. It provides mechanisms to help ensure that applications can be written to easily accommodate variations in the display type and layouts. This benefits application developers because it lets them write code that is display agnostic, and can be easily adapted to multiple display types.

Display Driver

The display driver abstracts the display-specific communication that is required on the controller area network (CAN) bus from the higher layers. It also provides character-set mapping.

When the system starts up, the display driver determines what type of vehicle display it needs to adapt to on the basis of a CAN message broadcast by the display head unit. Until that message is received, any requests for display write operations fail. Based on the received information the display driver chooses the appropriate character-set translation map, the supported display layouts, and a display-specific CAN signal assembly.

The display driver implements two sets of input/output controls (IOCTLs)—one for determining the supported layouts and another for writing a message or for clearing the display. Applications communicate directly with the display driver for streaming information, requesting display capabilities, and more. The display driver also needs to maintain state information with the audio subsystem because of dependencies between the audio source and a particular display control. The display driver does not distinguish between the different display sections on the display screen. It does not perform any queuing of requests. It also does not perform any verification of whether the application writing to the display has permission to do so.

The display driver relies on the character-set map to perform a mapping of Unicode characters to codes that the display understands. The character codes that are sent to the display are 6-bits long and are chosen from a set of 64 characters, which includes a default character used to map any unknown characters. The display driver also truncates the number of lines in a display request to actual lines that the layout supports (it does not truncate characters on a line—that is the responsibility of the display head unit or of the application itself).

Native Display API

The native display API layer uses the services of the display driver after opening a handle to the display driver through a call to `CreateFile` on “ICD1” (the device name for the display head unit). The handle is stored for subsequent `DeviceIoControl` calls. Upon success, the native display API layer requests the supported display layouts by calling `IOCTL_DISPLAY_GET_LAYOUTS` and then caches them.

The native display API layer is a pass-through to the display driver. The native display API layer performs focus and parameter validation and packages arguments to send to the display driver. All calls into the native display API are synchronous, and there are no callback or event mechanisms.

This API layer helps ensure that the application that is making a display request has obtained the display focus. Display focus is determined by whether the application has the current graphics, windowing, and events subsystem (GWES) foreground window. If the requesting application does have display focus, an appropriate IOCTL call is issued to the display driver. Applications can also query supported capabilities of the vehicle display from this API layer.

Note that a Windows Embedded Automotive HMI is typically defined using XAML and rendered through the XAML renderer. But some applications, such as map and video, have their own graphical outputs from their processes. These graphical outputs are composited into a single HMI and presented as one UI screen from a user viewpoint. Typically, this composition is done using a hardware overlay. Microsoft Silverlight for Windows Embedded provides a feature which blends the graphical outputs of the applications into the Windows Embedded Automotive HMI through the `CompositorCore`.







Projekt 2 Sample Application



Microsoft provides the Projekt 2 sample UX application to help jumpstart OEM UX development. This application provides a sample framework that designers and application developers can use to design their own UIs. This speeds up the design and development process while providing an example of a safe and robust UI.

Projekt 2 demonstrates the types of interactions that are supported by Microsoft Silverlight for Windows Embedded. The UI encompasses three key areas of automotive UI: phone, radio, and media playback. These UIs are tied into the Phone, Radio, and Media Cores, which help users receive and interact with phone calls, listen to the radio and manage radio station lists, and browse and play songs in the media library, respectively.

Table 1 shows a sampling of the Projekt 2 UIs.

Table 1: Projekt 2 UI elements

Projekt 2 UI	Description
	<p>The menu UI demonstrates user workflows for accessing the various Windows Embedded Automotive 7 components, such as navigation, phone, radio, media, and setup.</p>
	<p>The media UI provides the user access to the media library and playback control of supported audio file types. The UI also demonstrates the display of imported images associated with albums.</p>
	<p>The radio UI provides the user controls for managing radio functions. These controls include station lists, frequency tuning, and frequency band selection. The UI also displays the currently selected radio station, and any information that is sent over-the-air by the radio station.</p> <p>Note that the UI provides information about the phone, including Bluetooth connection status, signal strength, and battery level.</p>
	<p>The phone UI provides access to the most commonly accessed phone features. These include outbound dialing, contact management, messaging, and call history. The UI also displays message and call status information, such as the number of calls missed and the number of new received messages, including email and SMS.</p>

Projekt 2 UI	Description
	<p>The incoming call UI displays as a pop-up dialog over the currently selected UI. When a call is received, the application matches the caller ID information against contacts in the phonebook. If a match is found, the contact information is displayed, including name, phone number, and image. The UI provides options for controlling the call, including accept, ignore, and decline.</p>
	<p>The contacts UI provides the user access to the phonebook synchronized with the phone. The UI provides advanced features such as sort and search, as well as the capability to edit individual contacts.</p>

Phone Core Deep Dive

Telephone and data communications are a key component of an in-vehicle infotainment system, helping consumers access their phones and data in a safe, hands-free manner. Phone Core supplies a robust set of services and APIs that allow software engineers to design powerful phone and data applications that integrate with other Windows Embedded Automotive 7 applications to provide consumers a safe and accessible solution.

Phone Core Architecture Overview

Phone Core provides a uniform interface with services for hands-free telephony, managing Bluetooth profiles, phone management, data connectivity, and Short Message Service (SMS). These services integrate with the vehicle's audio system and provide a voice-based and touch-based interface.

Note that Microsoft recommends using the Phone Core API for all application development, unless otherwise noted.

The foundation of Phone Core resides in three components:

- Bluetooth profile:** Applications that rely on telephony and data communications—such as a phone application—reside at the topmost layer of the telephony and data communications architecture and use Windows Embedded Automotive middleware services and/or Bluetooth profiles. For applications that use Bluetooth wireless technology for data communications, a profile is applied to describe how to exchange specific types of data over Bluetooth. For example, a hands-free phone application must apply the hands-free profile (HFP) to define how to place and receive phone calls, as well as perform other phone-related functionality on the Automotive 7–based device using a Bluetooth audio gateway (a paired mobile phone).

- **SMS:** The SMS supports access to SMS messages that are received by a connected Bluetooth phone and supports sending of SMS messages by a connected Bluetooth phone. SMS messages can be retrieved via AT commands or the Message Access Profile (MAP) service. Automotive 7 also supports MAP email from any Message Access Service (MAS) instance. Email can be retrieved from the Bluetooth-connected device and then handled by an appropriate application or service.
- **Connection Manager:** Data connections established with other data sources—through the external mobile phone—are managed by the Connection Manager. The Connection Manager centralizes and automates the establishment and management of these connections for Windows Embedded Automotive applications, handling the details of each connection. The Connection Manager can establish and manage network connections and route data to the network using the Bluetooth stack. For connections to data sources established over Bluetooth, the Bluetooth stack is used to transport data to a Bluetooth host. Figure 4 shows the overall architecture of Phone Core. See the Glossary for definitions of unfamiliar terms and acronyms.



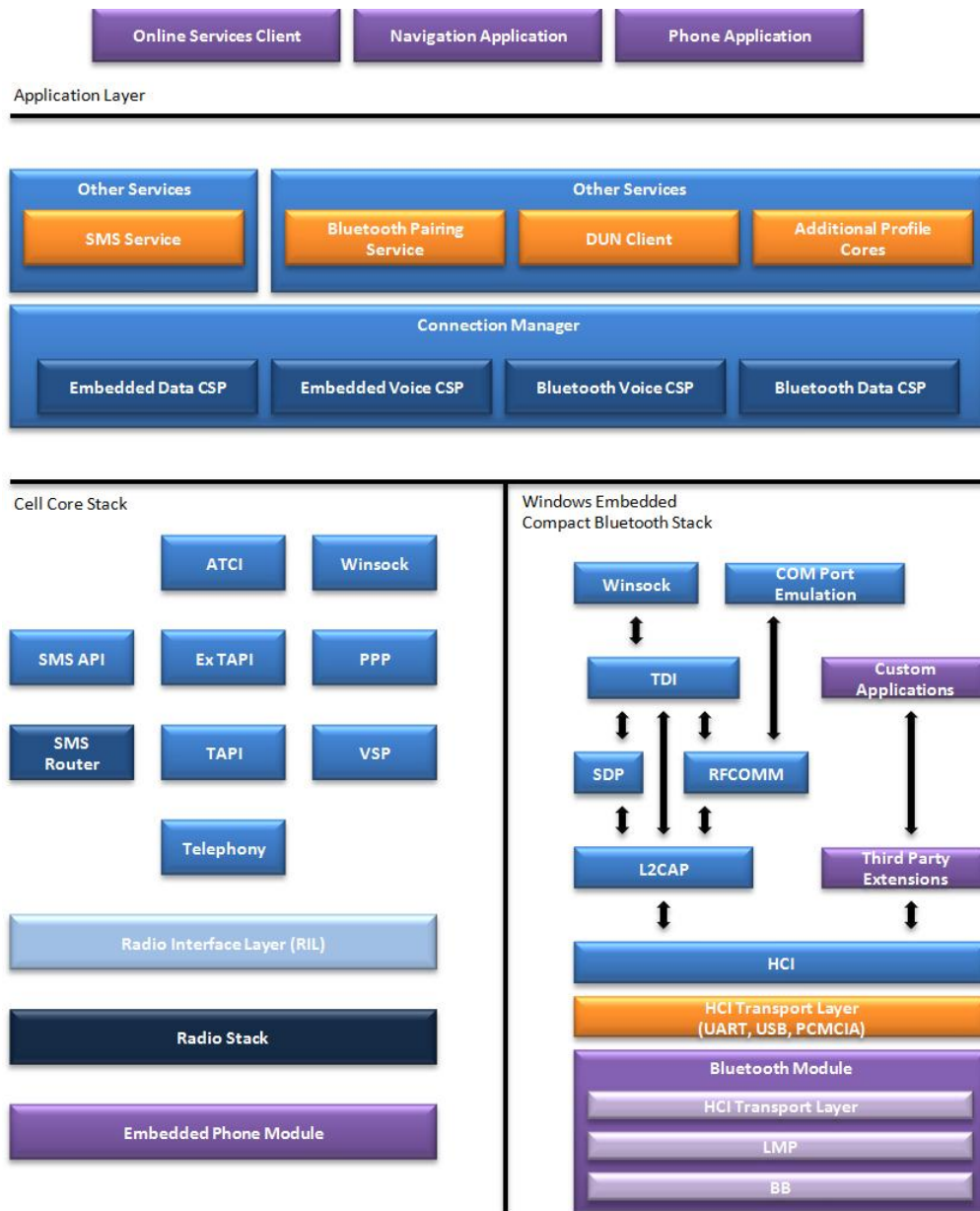


Figure 4: Phone Core architecture

Phone Connection Management

Windows Embedded Automotive 7 provides features to connect to a previously paired Bluetooth-enabled phone, as well as features to handle phone disconnects and phone availability. Only one Bluetooth-enabled phone can be connected at a time, and any additional attempts from other phones to connect to the Windows Embedded Automotive device are rejected.

Automatic Connection

Windows Embedded Automotive 7 can automatically connect to a previously-paired phone, based on events initiated from applications or phone-related functions initiated by the user, such as pressing the Send button on the phone.

Automotive 7 supports connection to a phone with an active call. If a call is in-progress at connection time, Windows Embedded Automotive queries the phone for the phone number and contact information if this feature is supported by the Bluetooth phone.

Manual Connection

Automotive 7 supports manual connection to a previously-paired Bluetooth device. A user can select a paired device from the UI and manually initiate a connection. If the connection is successful, a notification appears indicating that the device is connected. If the selected device is not found or fails to connect after one minute, the connection times out and the user is returned to the paired device selection UI.

Automotive 7 also supports manual connection events while automatic connection sequences are underway. If a user selects a manual connection routine for a paired device during an automatic connection sequence, Automotive terminates the automatic sequence. If a phone fails to connect during the manual connect routine, the automatic connection routine does not automatically resume unless the user either initiates a key-on event or presses a phone-related button.

Automotive 7 includes a feature that returns a notification of why a particular connection failed—either that the device is out of range, or that the link key is no longer valid. If the link key is no longer valid, a UI can be displayed that automatically begins a pairing sequence.

Disconnection

When a connected phone disconnects, Automotive 7 does not automatically trigger the automatic connection sequence to search for a device. Windows Embedded Automotive typically does not have any indication as to what caused the disconnect, such as whether or not a user powered off the phone, selected a privacy option, or manually disconnected the phone. Therefore, Automotive will not try to reconnect to the disconnected device.

Phone Availability

When a user attempts to perform an interaction that requires a Bluetooth-enabled phone be connected while no phone is paired, Automotive 7 notifies the user that no phone is connected. If a paired phone is available, Automotive notifies the user that it is trying to connect to the phone.

Phone Core Registry Settings

Phone Core is managed through registry key values. These values can be programmatically modified by applications.



The Phone Core registry keys are:

- **HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\PhoneCore**
- **HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\PhonebookOptions**

Table 2 details the Phone Core registry values. Table 3 details the **PhonebookOptions** registry key values.

Table 2: Phone Core registry values

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\PhoneCore		
Key Value	Type	Description
AutoConnectAttempt	DWORD	Specifies the number of loops through the paired phone list to attempt auto connect. The device attempts to connect to each of the paired phones in the list and repeats this process for a total of three times until a connection attempt is successful or until the third repetition of the process completes. The default value is 3.
AppConnectAttempt	DWORD	Specifies the number of times that the Windows Embedded Automotive 7–based device attempts to connect to the paired phone when the user initiates the connection. The default value is 1.
ConnectTimeoutSec	DWORD	Contains the timeout value, in seconds, for making a manual connection attempt. A manual connection attempt occurs when the user attempts to make a connection either from the phone handset or by making a selection from the Windows Embedded Automotive 7–based device display menu. Note that this timeout does not apply to auto connect. The default value is 30.
RingCountToPlayLocalRingTone	DWORD	Specifies the number of times to ring before cancelling the attempt to play a synchronous connection-oriented (SCO) ringtone. The default value is 3.
PhonebookDownload	DWORD	Specifies the phonebook automatic download flag. This value can be 1 to specify automatic download upon connection or 0 to specify no automatic download upon connection. The default value is 0.
PhonebookFull	DWORD	Specifies the value of the phonebook full flag. A 1 indicates the phonebook is full. The default value is 0.
ContactAdded	DWORD	Specifies the value of the contact added flag. A 0 indicates none were added to the phonebook. A 1 indicates a contact was added to the phonebook. The default value is 0.
SCORingTone	DWORD	Specifies whether the phone has SCO ringtone capability. The default value is 0.

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\PhoneCore		
Key Value	Type	Description
RingtoneOption	DWORD	Specifies the ringtone configuration options: <ul style="list-style-type: none"> • 0—no ringtone • 1—ringtone temporarily silent • 2—in-band ringtone (SCO audio connection) • 3—default local ringtone • 4—local ringtone number 1 • 5—local ringtone number 2 • 6—local ringtone number 3 The default value is 0.
HQRTOption	DWORD	Enables or disables high-quality ringtones (HQRT). A value of 0 disables HQRT, a value of 1 enables HQRT. The default value is 1.

Table 3: PhonebookOptions registry key values

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\PhonebookOptions	
Key Value	Description
ProgressInterval	Specifies the phonebook download progress bar notification interval. This value can be set to 0 to disable the progress bar. If set to any other value, the application is notified of download progress for ProgressInterval items. The default value is 20.

Phone Core Windows Messages

Applications can receive phone-related event messages, such as when a phone call is received or answered, and then act upon that message. For example, if an application receives a **WM_PHONE_CALLERID** message, the application can display the incoming call’s caller ID on an in-car display device.

Table 4 lists the Phone Core Windows messages and their descriptions.

Table 4: Phone Core messages

Message	Description
WM_PHONE_ANSWERCOMPLETE	Posted when a phone call is answered.
WM_PHONE_AUDIOTRANSFERRED	Posted when an audio connection is transferred between the Bluetooth phone and the Windows Embedded Automotive–based device.
WM_PHONE_BATTERYLEVEL	Posted to indicate the battery level of the Bluetooth phone.

Message	Description
WM_PHONE_CALL_CONNECTED	Posted when a phone call is connected.
WM_PHONE_CALLACTIVATED	Posted when a connected call becomes the active call.
WM_PHONE_CALLERID	Posted to indicate the caller ID.
WM_PHONE_CALLINFO_UPDATE	Posted when the connected call information is updated.
WM_PHONE_CALLONHOLD	Posted when a single call is on hold.
WM_PHONE_CALLSTATE_CHANGE	Posted when a call changes state.
WM_PHONE_CALLWAITING	Posted when an incoming call is waiting.
WM_PHONE_CARRIERCHANGED	Posted when the phone connects to a different carrier.
WM_PHONE_CONFACALLCONNECTED	Posted when a conference call is connected.
WM_PHONE_DIALCOMPLETE	Posted when the phone has dialed a requested phone number.
WM_PHONE_HANGUPCOMPLETE	Posted when a call is terminated.
WM_PHONE_HFPOBEXVCARDCOUNT	Posted to indicate the number of virtual business cards (vCards) received.
WM_PHONE_HFPPORT_CONNREQUEST	Posted when the Windows Embedded Automotive–based device attempts to connect to a paired phone.
WM_PHONE_HFPPORTCONNECTED	Posted when a Bluetooth phone is connected.
WM_PHONE_HFPPORTCONNECTFAILURE	Posted when an application-requested connection fails.
WM_PHONE_HFPPORTDISCONNECTED	Posted when a Bluetooth phone is disconnected.
WM_PHONE_PHONEBOOK_STATUSCHANGED	Posted when the phonebook status changes.
WM_PHONE_PHONEBOOKSYNCCOMPLETE	Posted when the phonebook on the connected phone has been downloaded.

Message	Description
WM_PHONE_RING	Posted when the phone is ringing.
WM_PHONE_SENDDTMFCOMplete	Posted when the user dials a digit, which generates a dual-tone multi-frequency (DTMF) audio signal, during an active phone call.
WM_PHONE_SERVICESTATE	Posted to indicate the carrier and service state of the phone.
WM_PHONE_SIGNALSTRENGTH	Posted to indicate the signal strength.
WM_PHONECORE_START	Posted when the Phone Core component is initialized.

Bluetooth Hardware and Stack

The Microsoft Bluetooth wireless technology stack implementation is a modular, general-purpose Bluetooth 2.1+EDR-compatible software stack. This stack makes up the core portion of the Bluetooth wireless technology implementation. Through a Bluetooth wireless technology connection, devices can exchange data and interact with one another. The host controller interface (HCI) software module supports various connections (universal asynchronous receiver/transmitter [UART] and Universal Serial Bus [USB]) to the Bluetooth wireless technology chip.

Figure 5 shows the components and relationships in the Phone Core Bluetooth stack.

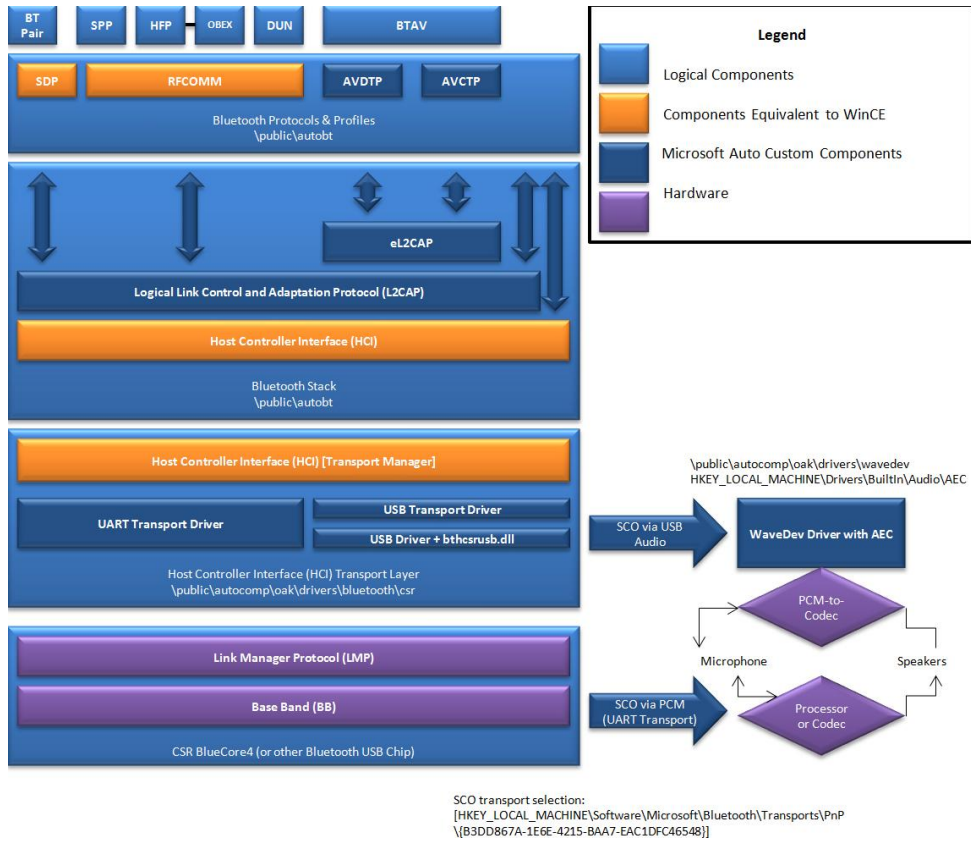


Figure 5: The Phone Core Bluetooth stack

Table 5 lists the additional layers that provide the core of the Bluetooth stack.

Table 5: Bluetooth stack layers

Layer	Function
Baseband	The physical radio layer.
LMP	Handles Bluetooth link establishment, authentication, and encryption.
HCI	Provides a uniform method of accessing the Bluetooth baseband capabilities by exposing a command interface to the baseband controller and link manager, and access to hardware status and control registers.
L2CAP	Logical Link Control and Adaptation Layer Protocol (L2CAP) provides the following functions: <ul style="list-style-type: none"> Multiplexing data between the different higher-layer protocols Segmentation and reassembly of data packets Providing one-way transmission management of multicast data to a group of other Bluetooth devices Quality of service (QOS) management for higher-layer protocols

Layer	Function
eL2CAP	Provides enhanced error detection and flow control.
SDP	Gives devices the ability to discover what services other Bluetooth devices support and what parameters to use to connect to them.
RFCOMM	Provides RS-232 serial port emulation, up to 60 simultaneous connections.
AVDTP	Applies point-to-point signaling over a connection-oriented L2CAP channel, which enables Advanced Audio Distribution Profile (A2DP) streaming.
AVCTP	Used by Audio/Video Remote Control Profile (AVRCP) to transfer Audio/Video Control (AV/C) commands via L2CAP.

The Windows Embedded Automotive 7 reference platform uses the CSR BlueCore4 Bluetooth chipset, a single-chip Bluetooth v2.1+EDR system. Microsoft provides the following key elements where the Bluetooth chip, HCI, and audio meet in the Bluetooth hardware implementation:

- Bluetooth chips route asynchronous connection-oriented (ACO) traffic via USB or a UART, which allows SCO audio to be routed via USB or PCM. The Automotive 7 hardware reference platform demonstrates BlueCore4 connections via both pathways.
- Registry configuration for BlueCore4 can be found in the registry at **HKEY_LOCAL_MACHINE\Software\Microsoft\Bluetooth\Transports\PnP\{B3DD867A-1E6E-4215-8AA7-EAC1DFC46548}**
- BlueCore4 can also be configured by setting the PSKEYs `\public\autocomp\oak\drivers\Bluetooth\csr_pskeys.h`. The power settings may require adjustment for controller subsystem qualification.
- The Bluetooth Class of Device (COD) for any device developed for Automotive 7 can be set in the registry, but the recommended default is **HKEY_LOCAL_MACHINE\Software\Microsoft\Bluetooth\sys, "COD"=dword:340408**

Acoustic Echo Cancellation/Noise Suppression

Windows Embedded Automotive 7 includes support for acoustic echo cancellation/noise suppression (AEC/NS), which removes noise and echo from voice communications. This improves voice quality on phone calls. Automotive 7 meets Verband der Automobilindustrie (VDA) standards for AEC/NS performance with the following attributes:

- Sending delay of 67.5 milliseconds (ms).
- Receiving delay of 57.62 ms (VDA standard is less than 120 ms).
- Echo delay of 125.12 ms (VDA standard is less than 260 ms).

Individual phones can have custom AEC/NS settings.

AEC/NS Configuration Registry Keys

AEC/NS is configured via three registry keys. The gain and audio processing registry key is **HKEY_LOCAL_MACHINE\Drivers\BuiltIn\Audio\AEC**. This registry key provides two values that are a

bit mask for 16 tunable settings. Microsoft does not recommend changing the settings, as this typically results in degraded audio quality.

Two additional registry keys hold AEC/NS configuration settings:

- **HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\HFP\SendNRECSetting**
- **HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\HFP>NoiseSuppressSetting**

Table 6 lists the key values that can be configured for each registry key.

Note: The first three values in the following table define whether or not the AT+NREC=0 command is sent to the phone’s audio gateway (AG). This HFP command turns off noise reduction and echo cancellation on a phone.

Table 6: AEC/NS registry key values

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\HFP\SendNRECSetting HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\HFP>NoiseSuppressSetting	
Key Value	Description
ALLOW_NREC_REGISTRY_OVERRIDE	The Hands-Free Profile Service (HFPSvc) sends the AT+NREC=0 command to the phone’s AG unless the PHONESPECIFIC_SKIP_AT registry setting is active for this phone manufacturer/model. This is the default setting.
ALWAYS_SEND_NREC	The HFPSvc always sends the AT+NREC=0 command to any AG.
NEVER_SEND_NREC	The HFPSvc never sends the AT+NREC=0 command to any AG.
USE_NS_UNLESS_NO_NREC	The HFPSvc enables AEC/NS unless the AT+NREC=0 command fails or is blocked from being sent by the HFP\SendNRECSetting command. This is a default setting.
ALWAYS_USE_NS	The HFPSvc always enables AEC/NS, independent of the HFP\SendNRECSetting command.
NEVER_USE_NS	The HFPSvc always disables AEC/NS, independent of the HFP\SendNRECSetting command.

Supported Bluetooth Profiles

Phone Core supports the Bluetooth Core 2.1+EDR specification with Secure Simple Pairing (SSP). You can find the complete specification at <http://www.bluetooth.com/English/Technology/Building/Pages/Specification.aspx>.

Phone Core supports the following Bluetooth profiles:

- Generic Object Exchange Profile (GOEP) 1.1
- Object Push Profile (OPP) 1.1

- Serial Port Profile (SPP) 1.1
- Phone Book Access Profile (PBAP)-Phonebook Client Equipment (PCE) 1.1
- A2DP-SNK 1.2
- AVRCP-Controller 1.4
- HFP-HFP 1.5 (backward compatible to HFP 1.0)
- Dial-Up Networking Profile (DUN)-DT and GW 1.1
- MAP 1.0
- SyncML 1.1.2
- Bluetooth SYNCH IrMC-Client 1.1.
- SIM access profile (SAP) Client 1.1
- Device ID profile 1.3
- Human Interface Device (HID) Profile 1.0
- Personal Area Network (PAN) Profile 1.0
- Simplified extensibility model for new Bluetooth profiles

Bluetooth Qualification

Microsoft has several Bluetooth qualifications that are available to OEMs. A Bluetooth controller subsystem must be qualified by a tier-one manufacturer to create an End Product Listing (EPL). By combining Microsoft's qualifications with their own qualifications, an OEM can create an EPL, which allows them to use the Bluetooth logo on products built using the certified hardware.

Note: The qualification design ID QDID for the Windows Embedded Automotive 7 host subsystem is B017024.

Bluetooth Pairing Core and Service

Windows Embedded Automotive 7 provides the Bluetooth Pairing Service that manages the Bluetooth service discovery process and device pairing process. OEMs do not need to develop the core functionality needed to pair with a Bluetooth-enabled device, as the Bluetooth pairing core provides this functionality. Applications can build upon this functionality, as the Bluetooth Pairing Service provides an interface to control the Bluetooth discovery and pairing process and manages Bluetooth devices once they are paired. The pairing service also maintains a database of Bluetooth profile information for each of the paired devices. Pairing creates a trusted relationship between two devices. As defined by the Bluetooth SIG, the pairing procedure creates a common link key that is used as the basis for a trusted relationship or a single secure connection.

Prior to pairing, a Bluetooth-enabled device attempts to locate the address, clock, class-of-device field, and used page scan mode from discoverable devices within range. Once discovered, the devices initiate the pairing procedure.

Bluetooth Pairing Core

The Windows Embedded Automotive 7 Bluetooth Pairing Core provides pairing functionality and paired device management to applications that require the establishment of pairing relationships and access to paired device information. This capability frees OEMs from having to develop custom code to handle Bluetooth device pairing management.



Several Automotive 7 applications rely on the Bluetooth Pairing Core, including the phone application, the media player, Windows Embedded Automotive shell, and the SMS application. Other applications can also use the Bluetooth Pairing Core API to pair and communicate with Bluetooth-enabled devices via the Bluetooth Pairing Service.

The Bluetooth Pairing Core consists of two parts: the service component, which is loaded by Services.exe, and the in-process dynamic-link library (DLL), which provides the API that abstracts the input/output control (IOCTL) calls to the service. All Bluetooth Pairing Core functions are located in the in-process DLL, and each function has a corresponding IOCTL. The core APIs wrap the task of packing function arguments into IOCTL calls to the service. The IOCTL packages the function's arguments inside of it in the DLL. The arguments are then unpacked in the service and processed by the corresponding function in the service.

The core functionality provided by the service is managing the paired device list. The service maintains three paired device lists for phone, media, and other devices. The service also handles the Bluetooth pairing procedure, but not the Bluetooth connecting procedure. The paired device list is an access ordered list that can contain a configurable maximum of 12 paired devices.



Figure 6 shows the components and data flow of the Bluetooth Pairing Core and Bluetooth Pairing Service architecture.

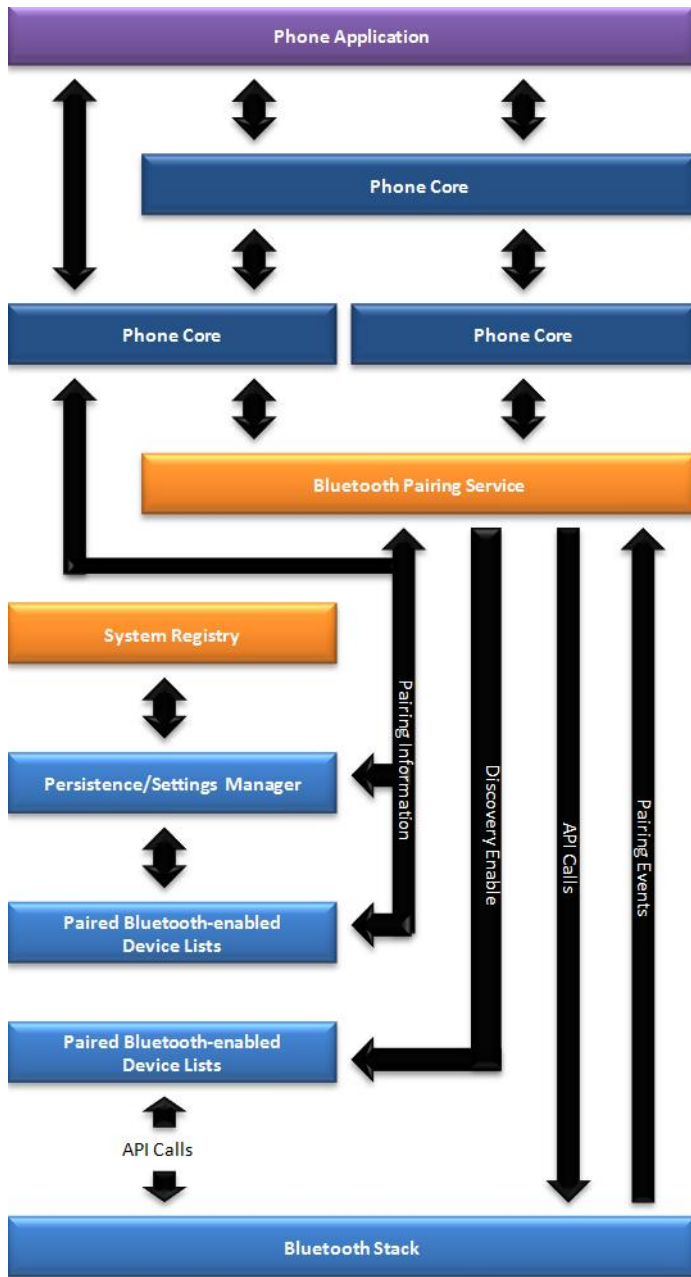


Figure 6: Bluetooth Pairing Core and Bluetooth Pairing Service architecture

Bluetooth Pairing Service

The Bluetooth Pairing Service manages the Bluetooth service discovery process and the device pairing process. It provides an interface to control the discovery and pairing process and to manage the list of paired Bluetooth-enabled devices. The service also maintains a database of Bluetooth profile information for each of the paired devices. Multiple phones can be paired to the Automotive 7 device, but only one may be connected at a time. The Windows Embedded

Automotive 7 phone application currently supports up to five paired phones. Up to 12 configurable media devices can be simultaneously connected through A2DP, in addition to a single phone.

The Bluetooth Pairing Service provides the following specific capabilities:

- Enable or disable the Bluetooth radio
- Start or stop discovery of nearby Bluetooth-enabled devices
- Start or stop pairing with a selected Bluetooth-enabled device
- Enable or disable Bluetooth discovery mode
- Provide a signal to the Phone Core API that a device has been paired
- Provide management capabilities to the Phone Core API to control paired devices' profile information
- Allow the Phone Core API to activate or deactivate a specific paired device by deregistering the communication port
- Provide the ability for the Phone Core API to append data to the paired device profile record as name-value pairs

When Bluetooth discovery is enabled, the Bluetooth Pairing Service waits for a pairing event from the Bluetooth stack. Once it receives an event and successfully authenticates using personal identification number (PIN) negotiation, the connecting device's Bluetooth address (BT_ADDR) is checked against the existing set of known devices. If the service finds an existing record, it obtains the Bluetooth link key again and then updates the device record. If an existing device record is not found, the service queries the device for a set of Bluetooth profiles Windows Embedded Automotive 7 can use. The service stores the list of supported services with the device pairing record in the registry.

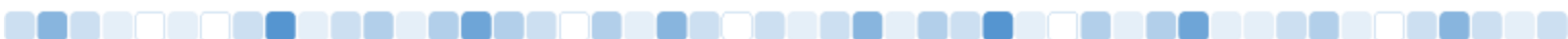
Higher protocol layers can append name/value pair attributes to each pairing record to support storing custom data. As new devices are added to the device list, the higher protocol layers are signaled via shared named events.

The discovery operation may pair only one device in each session. If no pairing event occurs, the operation times out. When a paired device is deleted, the service removes the information related to the device from the registry.

The Bluetooth Pairing Service information is located in the registry at the following location:

- **HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc**

Please see



Appendix 2: Bluetooth Pairing Service Registry Key Values for the key values.

Discovery Mode and Discoverable Mode

Discovery mode allows an Automotive 7 device to search for a nearby Bluetooth-enabled device to pair with. Discoverable mode allows an Automotive 7–based device to be found by a nearby Bluetooth-enabled device so that the Bluetooth-enabled device may establish a pairing relationship to the Automotive 7–based device.

When the device discovery starts, a phone or media application activates Bluetooth discovery mode using the Bluetooth Pairing Core API, which communicates with the Bluetooth Pairing Service to start an inquiry scan for nearby discoverable devices. When a nearby device is discovered, the Bluetooth Pairing Service notifies the Bluetooth Pairing Core, which routes a message to the application for each discovered device. When discovery mode ends, the Bluetooth Pairing Service notifies the Bluetooth Pairing Core, which routes another message to the application to notify it that discovery mode has ended.

To facilitate quick reconnection of a previously paired Bluetooth phone, an “ignition ON” event triggers Windows Embedded Automotive to enter a listening mode so that previously paired handsets can be reconnected as quickly as possible.

Special Pairing Features

Windows Embedded Automotive 7 includes support for the following features:

- Delayed PIN sharing
- SSP
- HID
- PAN
- Device ID

SSP simplifies the pairing procedure for Bluetooth users, while maintaining or improving Bluetooth wireless security. Automotive 7 supports the following SSP association models:

- Numeric compare
- Just works
- Out-of-band

Hands-free Phone

Automotive 7 provides robust hands-free capabilities when paired with Bluetooth-enabled mobile phones. These capabilities include the following:

- Authenticate command and control phones
- Receive service information from phones
- Manage incoming and outgoing calls
- Provide 3-way calling
- Maintain call history
- Receive call information from Bluetooth-enabled mobile phones

Automotive 7 provides the ability to download and access phone book data, in addition to initiating phone calls using the downloaded phone book entries. This provides users easy and



fast access to the phone's phonebook data through the Automotive 7 device's UI. See the Sync Manager and Phonebook sections for more information about this capability.

Hands-free Profile Service

HFP support is provided by the Hands-free Phone Core (HFPCore) service and the Bluetooth Pairing Service, which provide support for the Bluetooth SIG Hands-free Profile v1.5.

The hands-free profile (HFP) service provides users with hands-free access to their mobile phone. To place or receive calls, the service can use a mobile phone paired over Bluetooth, or an embedded phone module. For a paired Bluetooth phone, various features of phone call management—digit dialing, dialing by name, conference calling, call-hold, and so forth—are supported. Only one Bluetooth phone can be connected at a time.

HFPCore Service

The HFPCore service coordinates the interaction between applications with Bluetooth-enabled phones. HFPCore provides a native asynchronous (non-blocking) API (HFPAPI) that allows programmatic access to call handling, phonebook management, phone call audio management, and signals events—such as incoming calls—to its application clients through messages.

The HFPCore service hosts an implementation of the Bluetooth HFP 1.5 and controls the paired Bluetooth-enabled phone using AT commands over the phone's HFP and/or SPP ports. This implementation is backward compatible with HFP 1.0. HFPCore also enables Global Systems for Mobile Communications (GSM)-defined non-HFP AT commands for delivering uniform functionality to its clients.

Figure 7 shows the HFPCore service architecture.



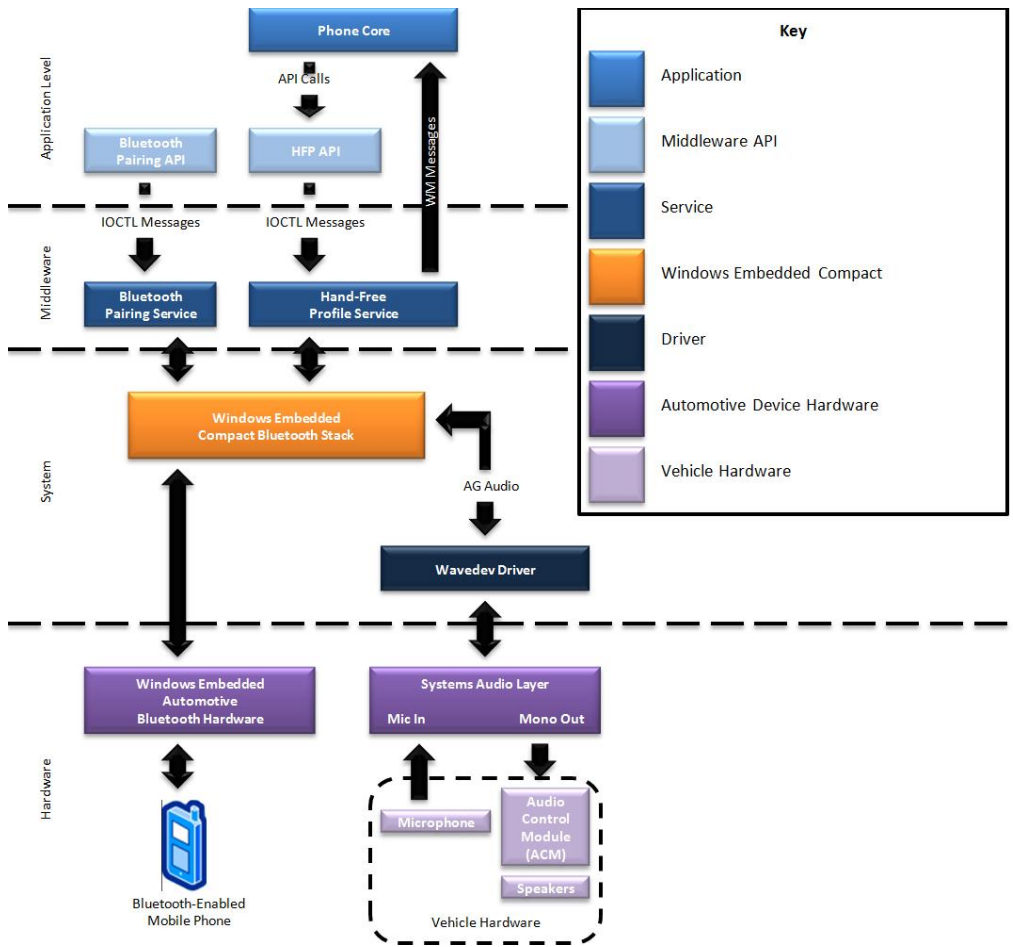


Figure 7: HFPCore service architecture

The HFPCore provides the following capabilities:

- Handle Bluetooth HFP port connected/disconnected status
- Place calls by number
- Report whether a call is connected or disconnected
- Report call privacy status
- Notify an application of an incoming call
- Enable/disable in-band ringing
- Provide ring type, such as local ringtone (WAV), SCO, and A2DP
- Report various phone status items, such as caller ID, signal strength, and battery level
- Notify an application when caller ID is received
- Answer or reject calls
- Receive notification of a call-waiting call with caller ID
- Place a current call on hold and switch to a waiting call
- Switch back and forth between active calls and calls on hold
- Toggle audio to and from the phone and the Windows Embedded Automotive device
- Retrieve the call list from the paired phone
- Report the status on phonebook download
- Receive object exchanger (OBEX) vCards via OPP
- Retrieve phonebook records
- Delete phonebook records
- Provide HFP **GetLastError** functionality

Call Handling

Windows Embedded Automotive 7 provides capabilities for handling a variety of call situations. These include:

- Incoming calls
- Multiple simultaneous calls
- Active call handling
- Conference calls
- Call termination

Incoming Calls

When an incoming call is initiated, Automotive 7 accesses the phonebook to determine whether or not a phonebook entry exists for the incoming number. If the incoming call is successfully matched to a phonebook contact, the contact name is passed back to the application. If the incoming call is not successfully matched to a phonebook contact, but caller ID is available, the caller ID number is passed back to the application. If the number is unknown for any reason—such as when the phone fails to report caller ID, there are two names with the same phone number in the phone book, or caller ID is blocked—HFPCore indicates to the application that the number is unknown.

HFPCore passes caller ID information back to the application using the **WM_PHONE_CALLERID** message.



Multiple Simultaneous Calls

A user can accept or place a second call while on an active call. When the user makes or accepts a second call—for example, a call waiting call—the active call is placed on hold, and the second call becomes the active call. The user can then either switch the calls or join them into a conference call. The application alerts the user through the UI that the first call is on hold.

Table 7 shows the multiple call behaviors supported by Windows Embedded Automotive 7.

Table 7: Supported multiple call behaviors

Status	Action	Behavior	+CLCC Support
One call	Place call on-hold	First call placed on hold	N/A
One call	Second incoming call accepted	First call placed on hold Second call is active	N/A
One call	Second incoming call rejected or ignored	First call retains state (on hold or active)	N/A
One call on hold	Resume call	First call active	N/A
One call	Terminate call	No calls	N/A
One call	Initiate second call	First call on hold Second call outgoing	N/A
One call held One call active	Third incoming call	Ignore third call until it is missed. Once “missed,” notify the user with call history of who it was.	Yes – Use +CLCC to achieve behavior No – Use +CLIP if received or simply state “missed call”
First call active Second call on hold	Swap	Second call active First call on hold	N/A
One call active One call held	Join calls	One conference call with both calls active	N/A
Conference call	Place conference call on hold	Not allowed	N/A
Conference call on hold	Change conference call to active	Not allowed	N/A
Conference call	One incoming call	Not allowed	N/A
Conference call One call on hold	Swap calls	Not allowed	N/A
Conference call One call on hold	Remove call from conference to active	Not allowed	N/A
Conference call with 2 calls	Remove call from conference to active	Not allowed	N/A

Status	Action	Behavior	+CLCC Support
Conference call	Regular terminate	No calls	N/A
One call active Second call incoming	Regular terminate	One call active Second call rejected	N/A
One call active Second call on hold	Regular terminate	Second call active	N/A
One call on hold	Regular Terminate	No calls	N/A
Conference call with x and y number of calls	Terminate call y	Call x active Call y terminated	+CLCC required
One call active Second call on hold	Terminate second call	First call active	+CLCC required
Incoming call	Ignore call	Call ringing stops on head unit. Call eventually goes to voice mail	N/A
One call active Second call on hold	Place active call on hold	Two held calls not allowed	N/A
One call active Second call on hold	Mute active call	First call active but muted Second call on hold	N/A

Active Call Handling

When a call is active, Automotive 7 displays the call information, as defined by the HMI specifications, to the user. An active call can have the following options made available through the UI:

- **Privacy:** This mode transfers the SCO audio to the phone handset. The call is still handled by Automotive 7. Turning this mode off transfers the audio back to the Automotive 7 device.
- **Hold:** This function places the active call on hold using the HFP_HOLD command, which is part of the standard Bluetooth command set. If the user places an active call or calls on hold using the handset, Automotive 7 indicates the hold status. An active call can be placed on hold when a new call is placed or a second call is accepted by the user.
- **Join:** This function gives the user the capability to join two calls to create a conference call. This function is allowed only when one call is on hold and the other call is active. The user may only join calls that are connected and cannot join a call that is incoming.

Conference Calling

Users can control two active simultaneous calls using this feature, and join the two calls into a conference call. Windows Embedded Automotive 7 only supports conference calling on phones that support the AT+CLCC and AT+CHLD mode 3.

If one of the calls is terminated or lost while in a conference call, Automotive 7 automatically reverts to a single active call state.

Call Termination

Automotive 7 provides the following call termination functionality:

- Pressing “END” while on a single active call terminates the call.
- Pressing “END” while in a conference call terminates both active calls.
- Pressing “END” during a dual-call state causes the active call to be terminated. The on hold call is then made active.

Dialing Feature Support

Automotive 7 provides the following dialing capabilities:

- Handset keypad dialing
- Speed dial
- Redial
- Digit Dialing
- DTMF

Handset Keypad Dialing

Automotive 7 lets a user dial using a connected handset. For phones that enable SCO at all times while connected, Automotive 7 mutes the SCO line unless a call is active or incoming. This prevents button presses from interrupting the user’s audio experience.

When a call is connected using keypad dialing or any other means that does not immediately report the number to the Automotive 7 device, Automotive 7 attempts to obtain caller ID information after the call is connected.

Speed Dialing

Speed dialing is available when the Bluetooth-enabled phone contains speed dial information. When using this feature, the Automotive 7 device can use the saved speed dial information on the paired phone to dial the phone number.

Redial

Redial functionality is provided via the HFP redial command. Redial uses the last outgoing number stored in the Bluetooth phone. When using redial, Windows Embedded Automotive 7 attempts to obtain caller ID information after a phone call is connected.

Digit Dialing

When a Bluetooth phone is connected, digit dialing is available to the user via a manual method through the UI, a speech method, or a combination of both, depending on an application’s implementation.

On vehicle platforms that include buttons numbering zero through nine, the user is able to enter digits to dial a number. A button combination allows the user to delete the last digit and clear all entered digits.

DTMF

Automotive 7 supports DTMF tones for all numerals, including the asterisk (*) and pound sign (#). A user can enter digits manually or by voice. The digits are sent while a call is connected.

Phone Feature Support

Automotive 7 provides support for the calling and phone features shown in Table 8.



Table 8: Phone feature support

Feature	Description
Signal Strength	On phones which support signal strength reporting over Bluetooth, Windows Embedded Automotive 7 reports this value to the user.
Roaming Report	On phones which have a roaming indicator available via Bluetooth, Windows Embedded Automotive reports this value to the user when paired with a platform having the appropriate display.
Battery Level	On phones which have a battery level indicator available via Bluetooth, Windows Embedded Automotive 7 reports this value to the user.
Power Management	Windows Embedded Automotive 7 will transfer audio to the Bluetooth phone and disconnect Bluetooth devices when power to the Windows Embedded Automotive 7 device is turned off.

Phone Core Versus HFPAPI

Microsoft recommends that OEMs develop applications using the Phone Core API instead of HFPAPI when possible to maintain timing and command order. Table 9 provides details on what features the Phone Core API manages.

Table 9: Phone Core API feature management

API	Features
Phone Core	<ul style="list-style-type: none"> In-process DLL which wraps HFPAPI messages and extends the base service for applications Manages Bluetooth phone objects and API routing Manages Bluetooth phone CONNECT_AUTO and CONNECT_APP Manages SCO audio connections Manages ring types (in-band SCO, HQRT, embedded) Provides interfaces to handle an embedded CellCore module Provides connection failure cause (not paired or out of range)
Host subsystem	<ul style="list-style-type: none"> In-process DLL interface to HFP service which provides call control (call, ring, notifications, caller ID, answer, reject) HFP Service contains specific phone workarounds and fixes

High-Quality Ringtone

Motorola phones can use HQRTs, though they do not notify connected devices that they are using HQRTs. Detection of HQRTs can take up to three rings and is configurable within the registry. The phone application should track the proper ring type (SCO, HQRT, embedded) on a per-connected phone basis.

HQRT can be configured using the following registry key:

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Automotive\PhoneCore**

The **HQRTOption** key value is used to set HQRT. A value of 0 disables HQRT, and a value of 1 enables HQRT. The **RingCountToPlayLocalRingTone** key value controls the number of rings that



occur before defaulting back to the local ringtone. When this registry setting is not present, the default value is 3. See Table 2 for more information about these settings.

Table 10 and Table 11 show the Windows Embedded Automotive 7 settings and the relationship between phone behavior and the resulting ring.

Table 10: Ring behavior

Windows Embedded Automotive 7 Settings	Phone Behavior	Result Ring
Local Ring	A2DP Ring	Local
	SCO Ring	Local
	No Ring	Local
In-band, HQRT Disable	A2DP Ring	Local
	SCO Ring	SCO
	No Ring	Local
In-Band	A2DP Ring	A2DP
	SCO Ring	SCO
	No Ring	Local
In-band, Media Core playing other stereo source	A2DP Ring	A2DP
	SCO Ring	SCO
	No Ring	Local
In-band, Media Core playing other A2DP source	A2DP Ring	Local
	SCO Ring	SCO
	No Ring	Local

Table 11: Ring behavior

Windows Embedded Automotive 7 Multi-Zone Settings	Phone Behavior	Result Ring
In-band, Phone A2DP in front, other source in rear	A2DP Ring	A2DP front
	SCO Ring	SCO
	No Ring	Local
In-band, other source in front, phone A2DP in rear	A2DP Ring	A2DP F/R
	SCO Ring	SCO
	No Ring	Local

Phone GetLastError

The phone **GetLastError** feature is useful in tracking phone HFP errors in deployed Windows Embedded Automotive 7 systems. OEMs can access the errors through a web service or local application, and then report the errors to the phone manufacturer. Alternative implementations include utilizing a USB memory stick to which error messages can be copied, or providing a graphical user interface (GUI) that displays a code that the user can provide to a customer support representative.

The interface for **GetLastError** is located in HFPAPI.h. The **WM_HFPERRORLOG** message is raised when a new error is logged by the HFPCore service.

The buffer that contains the errors is configurable through the registry key **HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\HFP**

The **ErrorLogMaxEntries** key value determines the number of error messages that are stored in the buffer. The default value is 5. The log is cleared when the HFP connection to the phone is closed. **HFPGetLastError** is used to retrieve error logs.

Table 12 lists the **GetLastError** wParam values.

Table 12: GetLastError wParam values

Class	Code in wParam
Outgoing Call Failures	0x00000001
Incoming Call Failures	0x00000002
Call Waiting Call Failures	0x00000004
Call Status Reporting Failures (CLCC)	0x00000008
Phonebook Download Failures	0x00000010
End Call Failure	0x00000020
2nd Outgoing Call Failure	0x00000040
Join Calls Failure	0x00000080
SCO Related Errors	0x00000100
Hold Call Failures	0x00000200
HFP AT Command Error	0x00000400
SMS Errors	0x00000800
CME Errors (phone response included)	0x00001000 Standard error values, as defined in the GSM specification, are appended to the wParam value.

Class	Code in wParam
CMS Errors (phone response included)	0x00002000 Standard error values, as defined in the GSM specification, are appended to the wParam value.
HFP Port Failures	0x00004000

Sync Manager and Phonebook

Windows Embedded Automotive 7 provides users with the ability to access their phone's contact information through the UI. This gives users a safe and fast way of accessing their phone's contact list to make calls, as well as providing OEMs a way to display contact information based on caller ID information.

Prior to Microsoft Auto 4.0, the HFP Service contained all of the phonebook download components, and the HFP API controlled the download initiation and messaging. In the legacy model, the HFP Service downloaded the phonebook using PBAP, SyncML, and then AT commands. Phonebooks were stored in a structure written to a persistent file in flash memory and read into memory as a cache.

In Automotive 7, Microsoft separated the phonebook download components into individual elements but maintained full-backwards compatibility. Each profile now has its own header in `autocomp\sdk\inc`, including a Sync Manager header. This architecture change provides significant advantages to developers using Automotive 7. During the design of the new architecture, Microsoft ensured that code paths from their shipping, field-tested, and hardened call handling and phonebook downloading compatibility code was preserved in the new architecture.

Windows Embedded Automotive 7 supports the following mechanisms for downloading phonebook data from supported phones:

- PBAP 1.1
- SyncML 1.1.2
- Phonebook-related AT commands
- OPP 1.1
- SYNCH 1.0 (Note that by default, this is disabled for phonebook and used for downloading calendar items only.)
- OBEX

Figure 8 shows the phonebook architecture.

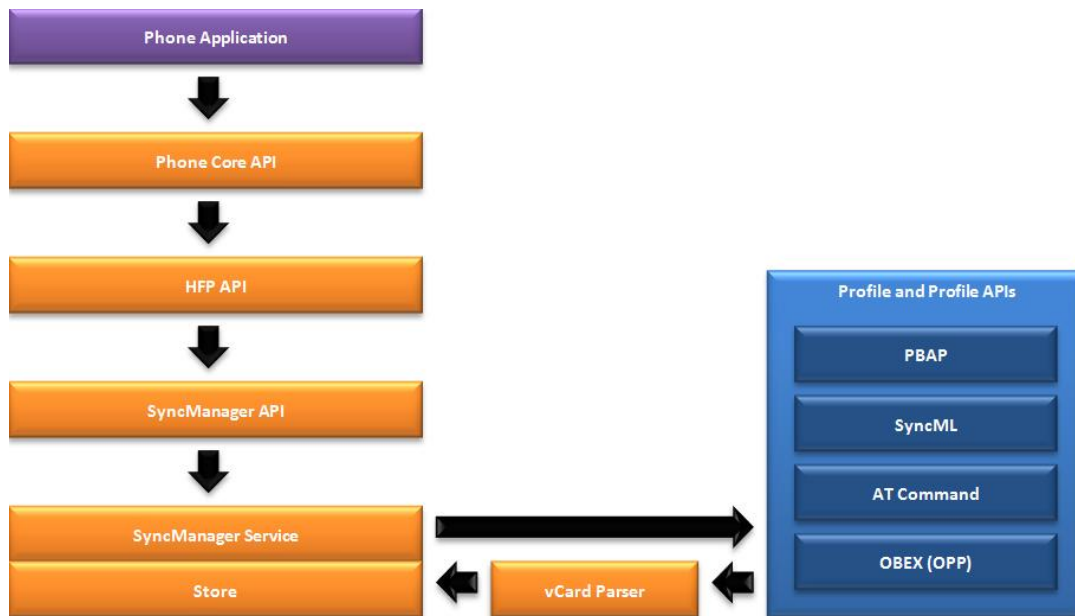


Figure 8: Phonebook architecture

Phonebooks are stored on a per-phone basis. Automotive 7 provides the following high-level phonebook storage functions:

- When a phone is first paired the phonebook is downloaded from the phone and stored locally.
- When a paired phone is reconnected, Automotive 7 refreshes the stored phonebook.
- When a paired phone is disconnected, Automotive 7 does not delete the phonebook, but keeps it in storage.
- When a paired phone is deleted, the stored phonebook associated with that phone is also deleted.

Phonebook Text Storage

Windows Embedded Automotive 7 can store phonebooks in flash storage as simple text files. The maximum text file size is 384 KB, which allows approximately 2,000 contact names with an average of four phone numbers per name to be stored. Note that the total number of contact names cannot exceed 2,000. Contacts can have more than four phone numbers, but the additional space usage may reduce the number of contacts available as the storage space cannot exceed the 384 KB limit.

Phonebook Storage Using the Pocket Outlook Object Model

The Pocket Outlook Object Model (POOM) is a Component Object Model (COM)-based library that provides access to personal information manager (PIM) data on mobile devices. Automotive 7 includes a customized version of the POOM component that is optimized for use in automotive applications.

The enhanced POOM contains the following features:

- The AUTOPOOM schema, with support for Bluetooth addresses and the Bluetooth SyncIndex
- Support for image files (for example, images of email recipients)
- Significantly increased contact search speed

Applications can access the contacts database and read and write to it when using POOM storage. However, the Sync Manager service assumes that other applications will only read from the contacts database. Therefore, other applications writing to POOM storage may cause errors in Sync Manager operations.

The following are differences in how Sync Manager handles vCard information when using POOM storage:

- If the address is identified as a business address, then the office location is saved in the office location field. The office location field is attached to the street field.
- The first number retrieved is saved as a mobile number. Subsequent numbers are saved as additional mobile numbers separated by semicolons. Numbers that exceed the length of the POOM field are discarded. The same behavior occurs with home numbers, work numbers, and other number fields.
- When the cache is populated from the POOM storage, vCard fields in the following table are mapped to the corresponding HFP service-defined fields.

Table 13 enumerates the vCard to HFP field mappings.

Table 13: vCard field to HFP field mappings

vCard Field	HFP Field
BusinessTelephoneNumber	HFP_PNTYPE_LOC_WORK
HomeTelephoneNumber	HFP_PNTYPE_LOC_HOME
OtherTelephoneNumber	HFP_PNTYPE_UNKNOWN
MobileTelephoneNumber	HFP_PNTYPE_LOC_MOBILE

Applications can save multiple records into POOM that contain the same contact name. To ensure the best user experience when the user does this, consider the following items when designing applications:

- When using the POOM synchronization manager with cache implementation, which is the default setting, the name of the contact should serve as the unique identifier for each record in your UI. The APIs are designed using this assumption and combine all phone numbers that are listed under a first name and last name pair when queried. Contact names and phone numbers should be retrieved through the HFP APIs to take advantage of the cache when retrieval time is important (for example, during an incoming call). Contact names, phone numbers, and all other contact fields can be retrieved when contacts are accessed by using the POOM APIs.
- When retrieving data from the POOM based on a first name and last name combination, the application should continue to search even after a single entry is found in the

database because several entries might exist with the same first name and last name combinations.

- When designing a UI, assume that multiples of all fields in POOM may be returned for any given first name and last name combination. Include the ability to display multiple numbers under a single location tag (for example, HOME: 2065551212, 4255551212,). The APIs provide parsing of multiple POOM fields when contact names and phone numbers are retrieved. Other fields available using the POOM APIs should also be considered in the UI design, such as a menu to select a specific field (for example, an address or picture) when multiple contact records with the same contact names have multiples of that field available.

Table 14 provides POOM storage information.

Table 14: POOM storage information

POOM Storage Item	Description
Total downloadable contacts	The default is 2,000. Contacts downloaded after 2,000 are discarded. This value can be modified in the registry for flash memory space and grammar considerations.
Total cache size	The default is 384 kilobytes (KB). Contacts downloaded after the full name , work , home , mobile , and other phonebook fields reach this limit are discarded. This value can be modified in the registry for RAM space considerations.
POOM MaxSize for flash	The default value is 16 megabytes (MB), but can be limited by placing the .vol file on a smaller partition.
POOM field maximum size	The value is 1,200 bytes for each non-binary field and is not configurable.
POOM record limit	The value is 64 KB and is not configurable.
AUTOPOOM special fields	The phonebook work , home , mobile , and other fields can accept more than one record with comma separations if an application uses Sync Manager and cache. The rest of the fields accept only one entry, and the 1,200 byte limit still applies to each field.

Phone Contact Image Auto Compression

A new feature in Windows Embedded Automotive 7 is the capability to compress images that are downloaded in phone vCards. This feature is configured via the following registry key and value:

- **HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\PhonebookOptions, MaxPOOMImageSize**

MaxPOOMImageSize, which is the POOM photo field maximum size, has a maximum value of 60 KB.



Phonebook Implementations

Automotive 7 provides modern phonebook implementation types while remaining compatible with prior Microsoft Auto phonebook types. All elements of pre-Microsoft Auto 4.0 phonebook types are statically built into the HFP service.

Microsoft recommends using the Sync Manager because it can manage multiple download types, it is extensible to providing new services and it can handle phonebook, calendar, and email synchronizations. Table 15 lists the supported phonebook implementation types.

Table 15: Supported phonebook implementation types

Phonebook Implementation Type	Description
Sync Manager	Windows Embedded Automotive 7 Sync Manager provides phonebook synchronization between Windows Embedded Automotive 7 systems and Bluetooth devices. Sync Manager calls the various APIs for PBAP, SyncML, ATCmd, SYNC, and OBEX to coordinate the download of calendar and contact information from Bluetooth phones. By default, Sync Manager automatically downloads vCard calendar and phonebook information from Bluetooth devices. Sync Manager is fully tested with devices for complete legacy phonebook compatibility. It utilizes both POOM and a cache mechanism for improved speed. It also includes RAM budgeting, multi-number handling, and command and private OBEX store strategies. If POOM storage is used, all information is downloaded, including appointments. Recursion and notifications are supported.
Legacy phonebook	The legacy phonebook type, used in Microsoft Automotive version 4.0 and earlier, uses a memory structure and flat flash file to store phonebook contacts. Phonebooks downloaded using this method are stored in a structure written to a persistent flat file structure in flash memory and then read into memory as cache. If file storage is used, only names and phone numbers are saved; no calendar information is downloaded.

OBEX Phonebook Stores

Windows Embedded Automotive 7 supports two OBEX storage methods: the common OBEX phonebook method and the private OBEX phonebook method.

The common OBEX phonebook method assigns a phonebook to the car, which is available to all of the car's users. Any contact received via OBEX is stored in a common phonebook.

The private OBEX phonebook method maintains a relationship with the last connected phone such that any contact received from any previously paired device is saved to the phonebook of the most recently connected phone. A sophisticated merging mechanism allows the HFPAPI to view the private auto-downloaded and private OBEX stores as a single phonebook location. Depending on the application UI, this method can be used to allow users to download the private phonebook associated with their individual phone with contacts from a laptop computer or personal digital assistant (PDA) device.

Note: OBEX contacts are downloaded using the HFPAPI, not through the Phone Core API.

Figure 9 shows the various components and relationships of the OBEX stores.

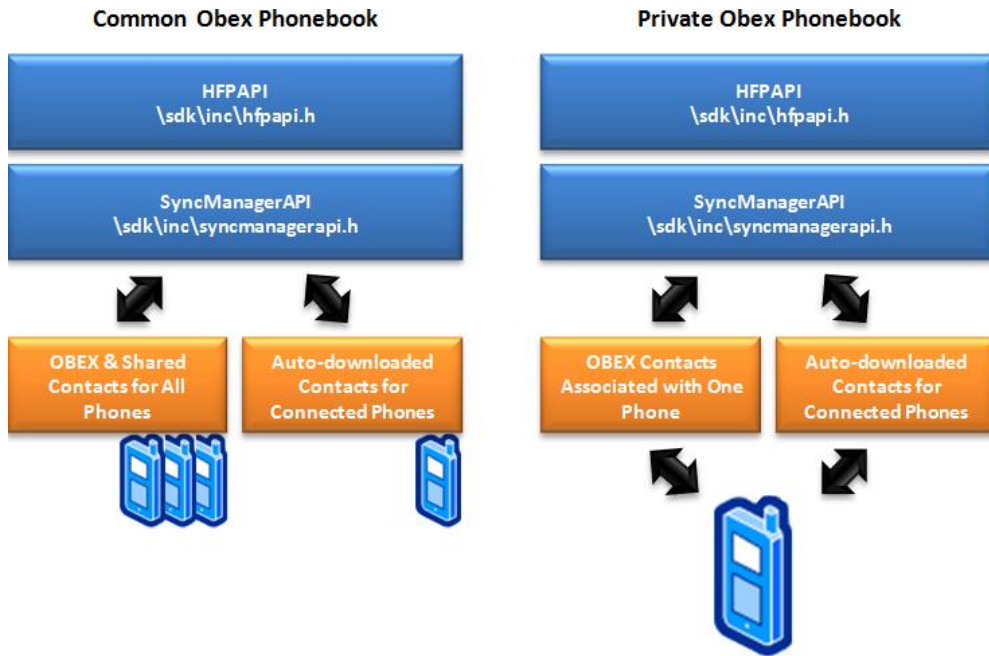


Figure 9: OBEX phonebook stores

Sync Manager Architecture

The Sync Manager architecture provides interoperability between the various phonebook features, such as POOM, image storage, multiple location, and other key features that are only available when using POOM. The cache provides rapid access for names, phone numbers, and phone number locations.

Figure 10 illustrates the Sync Manager architecture.

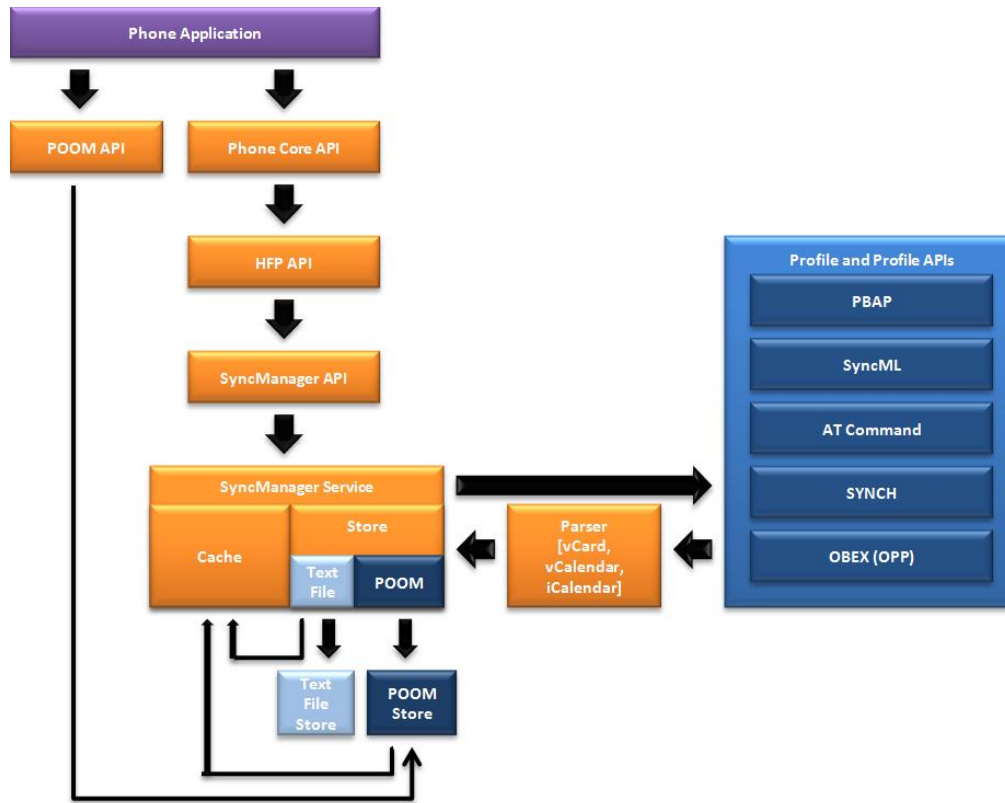


Figure 10: Sync Manager architecture

SMS Support and Email

Windows Embedded Automotive 7 supports access to and sending of SMS messages using a connected Bluetooth phone. SMS messages can be retrieved via GSM AT commands or the MAP service. Automotive 7 also supports MAP email from any MAS instance. Email can be retrieved from the Bluetooth-connected device and then handled by an appropriate application or service.

When AT commands are used, the HFP service interacts with the Bluetooth phone through either the SPP port or the HFP port. The HFP service sets up event notifications so that it can receive SMS messages and send SMS messages, but does not necessarily enumerate through and read the existing SMS messages that are stored on the phone.

Figure 11 shows the SMS support architecture.

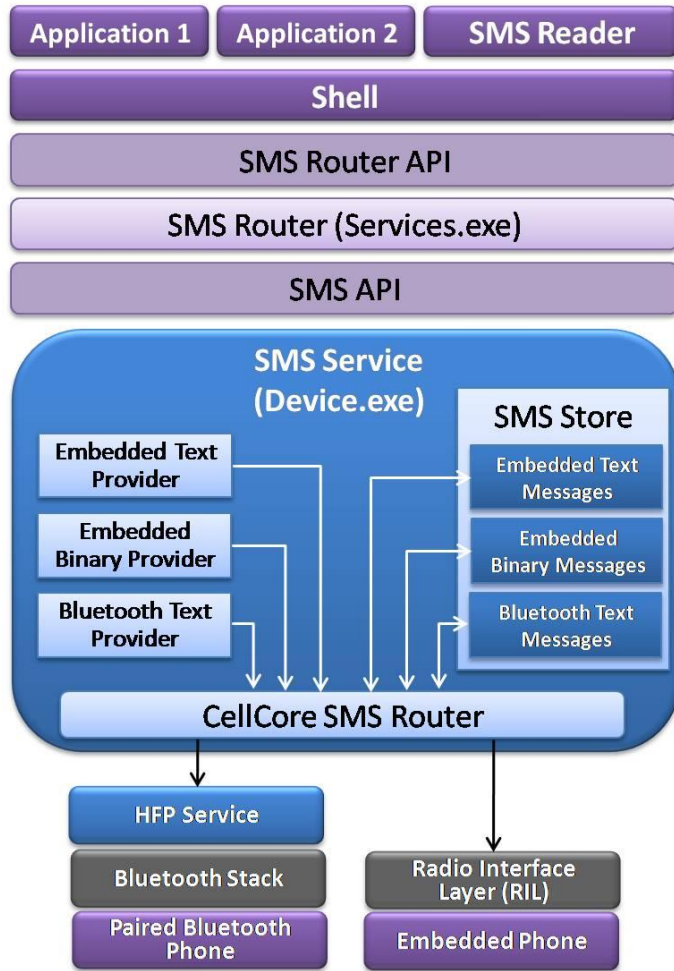


Figure 11: SMS support architecture

The SMS architecture contains both the SMS drivers and the subscriber identity module (SIM) drivers. The SMS drivers provide the standard Windows Embedded Compact device driver interface for SMS.dll. The SMS drivers also contain the SMS router and SMS store components.

The Bluetooth Text Provider handles text messages sent using a paired Bluetooth phone.

The SMS store caches SMS messages in the CellCore message queues. Each provider contains one message queue. Each message queue is stored in a memory-mapped data file and map file in the object store. Access to the message queue is protected by critical sections.

When a message is read from an embedded phone or a Bluetooth phone by the Radio Interface Layer (RIL), by the HFP service (SMS via AT command), or by the MAP Manager, it is sent using the CellCore SMS router. The message passes through the providers that are set up in the SMS service, and the message is decoded. Applications that have subscribed to SMS router notifications are alerted that the message is available. The message is cached in the SMS store for a developer-configurable period of time.

An SMS application that is provided by an OEM should use the SMS router API to subscribe to messages. This method lets messages be filtered and even targeted for specific applications through the SMS router. Message filtering, parsing, and targeting are all configurable.

GSM SMS AT Command Support

A phone must support MAP or GSM SMS AT commands to operate with Windows Embedded Automotive 7. Phones must be able to send, receive, notify, and download all unread messages. Automotive 7 supports USC2, GSM-encoded characters, and unpacked 8-bit encoding.

SMS Stack and MAP Service

The SMS stack includes services for SMS message retrieval via SMS and AT, Protocol Data Unit (PDU) message parsers, and message routing mechanisms. Figure 12 illustrates the components and relationships of the SMS stack.

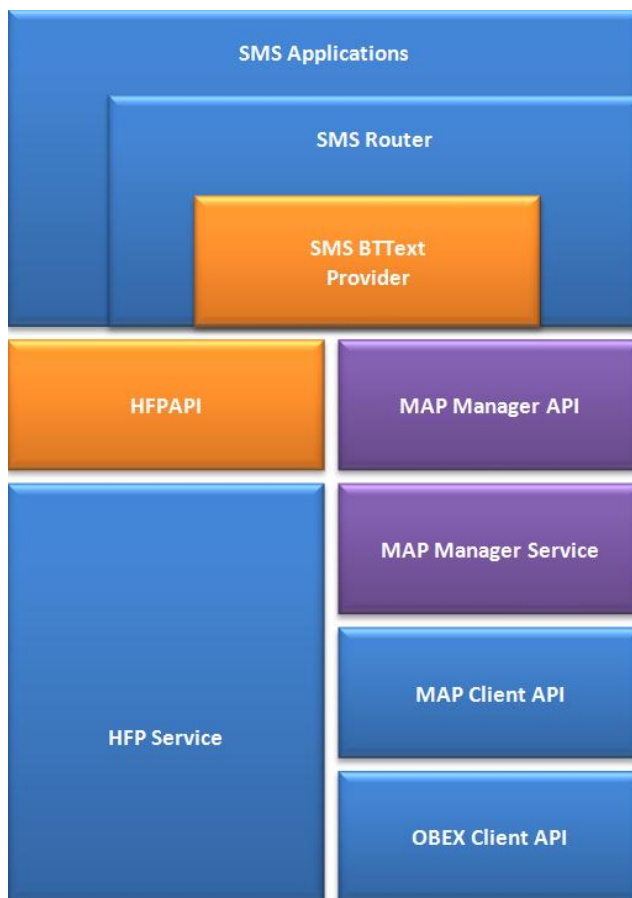


Figure 12: SMS stack

The MAP provides a set of features and procedures to exchange SMS and e-mail messages between devices. The MAP service interfaces to the OBEX layer of the Bluetooth stack and provides functionality for new message notifications and folders for incoming and outgoing SMS messages. The MAP Manager handles SMS download arbitration (SMS or MAP), phone-specific SMS behaviors, and provides the overall SMS interfaces.

Bluetooth Audio/Video (BTAV) Service

The BTAV service supports both A2DP 1.2 and AVRCP 1.4. This service manages the A2DP and AVRCP profiles and interfaces to Phone Core and Media Core. For A2DP, Media Core controls the connection management, media streaming, and player control.

Figure 13 illustrates the BTAV service architecture.

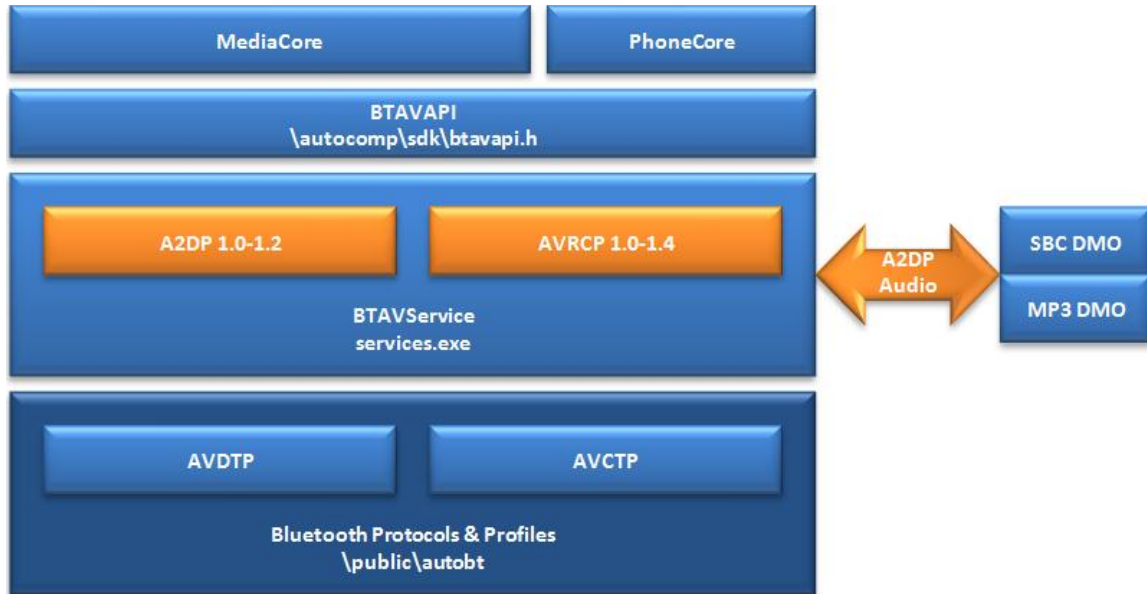


Figure 13: BTAV service architecture

The following tables list the registry settings for the BTAV service.

Note: Microsoft tests hundreds of devices and has tuned these registry settings for the widest device compatibility possible. Modifying them may result in device incompatibility.

Table 16: BTAV service registry settings

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\BTAV		
All keys and values under the BTAV key are optional. The BTAV service will use default values if a value is missing or outside of the appropriate range. This key includes one subkey for each supported codec.		
Name	Type	Description
ACPMoDe	DWORD	<p>The default acceptor mode for A2DP.</p> <p>Valid Data:</p> <ul style="list-style-type: none"> BTAV_A2DP_ACP_ROLE_DISABLED (0) indicates that the acceptor role is disabled. BTAV_A2DP_ACP_ROLE_ENABLED (1) indicates the acceptor role is enabled. <p>The default value is 1.</p>

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\BTAV

All keys and values under the BTAV key are optional. The BTAV service will use default values if a value is missing or outside of the appropriate range. This key includes one subkey for each supported codec.

Name	Type	Description
ConnectionStepWait	DWORD	The sleep time between each step (Discovery, GetCapabilities, SetCapabilities, OpenStream) while connecting to the device. The default value is 150.
MaxDevices	DWORD	The maximum number (1-12) of A2DP devices that can be open at one time. The default value is 1.
PowerRequirementDevice	String	Defines the device that provides access to the Bluetooth chip by the Bluetooth stack. The device will be set to full power mode when an A2DP or AVRCP connection has been established. There is no default value.
SCOBounce	DWORD	The A2DP audio temporarily turns off when the SCO notification is activated. This value defines the delay time between the SCO notification and the moment the A2DP audio is actually switched off. The default value is 0.
TimeoutA2DPConnection	DWORD	The timeout, in milliseconds, for establishing an A2DP connection; must be between 100 and 600,001. The default value is 10,000.
TimeoutA2DP	DWORD	The timeout, in milliseconds, for all A2DP operations other than establishing a connection, must be between 100 and 600,001. The default value is 5,000.
TimeoutAVRCP	DWORD	The timeout, in milliseconds, for all AVRCP operations. The default value is 2,500.
TimeoutDeviceRecycle	DWORD	The time, in milliseconds, that the BTAV service will keep a device open before closing connections and closing the device. Applies only when the BTAV service is in acceptor mode. The value must be greater than 100. The default value is 60,000.
TimeoutStartStream	DWORD	The amount of time, in milliseconds, to wait for the device to start the stream. The stream must be started manually if the device didn't start the stream. The value must be between 100 and 600,001. The default value is 5,000.
WaveOutDevice	DWORD	The ID of the wave out device which receives the decoded audio data. The default value is WAVE_MAPPER.
PlayThreadPriority	DWORD	The priority the play thread is running on. The value must be between 10 and 255. The default value is 230.
PlayThreadTimeout	DWORD	The timeout, in milliseconds, to wait for the play thread to start after the stream has started, or the timeout to wait for the play thread to stop after the stream is suspended. If the timeout expires upon starting, then an error message is given. If the timeout expires upon suspending, then the play thread is forcefully terminated. The value must be between 100 and 600,001. The default value is 1,500.

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\BTAV

All keys and values under the BTAV key are optional. The BTAV service will use default values if a value is missing or outside of the appropriate range. This key includes one subkey for each supported codec.

Name	Type	Description
WaveOutDevice	DWORD	The device Line Out ID for the wave driver that is assigned to A2DP when the wave driver has multiple line outs. The default value is WAVE_MAPPER.
OutputWaveBufferSize	DWORD	The size of each output buffer in the output buffer set. The value must be greater than 1,024. The default value is 1,024*5.
InputBufferTime	DWORD	The amount of time buffered for the input. The value must be greater than 100. The default value is 200.
OutputBufferTime	DWORD	The amount of time buffered for the output. The value must be greater than 100. The default value is 400.
DataTimeoutPercentage	DWORD	Indicates whether data is still being sent. The wait timeout is calculated by $\text{OutputBufferTime} * \text{DataTimeoutPercentage} / 100$. If no data is received for this amount of time, then it is presumed that the streaming has stopped. The default value is 50.
PrintBPSAverage	DWORD	The time frequency, in milliseconds, that is taken to print out the average bits/seconds. For example, if this registry setting is set to 30,000, then the service will print out a message every 30 seconds. The minimum value is 1,000 and maximum value is INFINITE. A default value is not set.
TimeoutAVRCPConnection	DWORD	The AVRCP timeout for connection setup. The default value is 5,000.
TimeoutCheckPairing	DWORD	The timeout when checking for whether or not the device is paired. The default value is 1,000.
NumItemsPerQuery	DWORD	The number for subrequests that constitute a query. With AVRCP 1.4, a device can be queried using a file system folder model. For example, a folder on the device file system named "Music" could contain all of a user's music files, which could number into the thousands. To handle the query for all of the files in this folder, the query is automatically broken up into subrequests, where the number of items per actual query is defined by this registry setting. The minimum is 1 and the maximum is 1,000. The default value is 5.

HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\BTAV\SBC

The Sub-Band Coding (SBC) codec is the only required codec for the A2DP. This registry key is used to store default information for the SBC codec.

Name	Type	Description
CLSID_SBCDMOMediaObject	GUID	The universally unique identifier (UUID) to find the DirectX Media Object (DMO) for the decoding process. The default value is {33FBF37D-60C9-4fb6-9638-081AF124036D}.

HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\BTAV\SBC

The Sub-Band Coding (SBC) codec is the only required codec for the A2DP. This registry key is used to store default information for the SBC codec.

Name	Type	Description
MinBitpool	DWORD	The minimum bit pool value supported. This value must be between 2 and 25. The default value is used when data is outside the range of 2 to 250, or if MaxBitpool is smaller than MinBitpool. The default value is 18.
MaxBitpool	DWORD	The maximum bit pool value supported. The default value is used when data is outside the range of 2 to 250, or if MaxBitpool is smaller than MinBitpool. The default value is 53.
44KHz	DWORD	The preference for a 44.1 KHz sample rate over a 48 KHz sample rate. A value of 0 indicates 48 KHz, whereas a value of 1 indicates 44.1 KHz. The default value is 1.

HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\BTAV\MP3

The information about the MP3 codec is stored in the registry. If no information for the MP3 codec is available, then the default value is used.

Name	Type	Description
CLSID_MP3DMOMediaobject	GUID	The UUID to find the DMO for the decoding process. The default value is {6b928210-84e7-4930-9b33-1eb6f02b526e}.

HKEY_LOCAL_MACHINE\Software\Microsoft\BTPairSvc\Profiles

The Bluetooth Pairing Service uses this registry key to maintain information about profiles and paired devices. The BTAV service adds subkeys to the Profiles key for A2DP, audio source, and AVRCP profiles, and examines the Devices key for profile information about paired devices. The BTAV service will not use devices that do not support the required profiles.

Name	Type	Description
4	DWORD	Profile 4, audio source. The default value is 4,362.
5	DWORD	Profile 5, the AVRCP target. The default value is 4,364.
6	DWORD	Profile 6, A2DP. The default value is 4,365.

Calendar

Windows Embedded Automotive 7 can download calendar information from mobile devices. Devices that use standard Bluetooth protocols, such as OBEX, SyncML, and Sync Profile, support import functionality for calendar and task information. Calendar storage is based on the Bluetooth address of the device and is stored in POOM storage. Automotive 7 supports virtual calendar (vCalendar) 1.0 and iCalendar 2.0, as well as vCalendar send.

Automotive 7 uses the CalNot.exe application for setting and deleting existing POOM notifications. An application can turn notifications on and off for the currently connected phone.

By default, notifications are turned on. You can also configure POOM notifications to download calendar and task information and to alert users of upcoming appointments.

New to Automotive 7 is full appointment recursion support. Users can set up recurring appointments on their mobile device. The recurring appointments are downloaded and handled as recurrences in Automotive 7.

Individual Phone Configurability

As part of ongoing device support, the Phone Core services include over 700 workarounds for specific phone devices. Certain phones may have issues that can cause problems with the user experience, so Microsoft’s Device Lab tests hundreds of devices per year for compatibility and creates workarounds for phones that do not function properly.

Automotive 7 identifies phones by issuing the AT+CGMI and AT+CGMM commands to get a phone’s model name and manufacturer. Specific compatible behaviors are tied to the various phone models that are tested by the Device Lab. These behaviors are stored in the registry. If a specific phone cannot support critical hands-free functionality and no workarounds are technically feasible, Automotive 7 can disable specific phone features.

Phones can further be fine-tuned through the following registry key and values. These settings can be optimized for either a phone name or a specific Bluetooth address.

Table 17 shows the phone configuration key values.

Table 17: Individual phone configuration key values

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\HFP\PhoneSpecificMasks\<Manufacturer>\<Model>	
Key	Description
Character Sets	Character sets supported by the device.
ENABLESYNCML	Uses the SyncML protocol to download the phonebook from the phone.
DISABLESNIFF	Disables sniff mode when connected to this phone.
SMSASSUMPTION	Assumes that the phone never reports new messages in the ME database unless the SM database is full.
AVOIDSYNCMLIFLISTENPORT	Does not start a SyncML session when the listen port is being used for the HFP port.
AVOIDSPPFORATPB	Does not use the SPP port to download the phonebook contacts when using AT commands.
DISABLESYNCML	Disables SyncML support.

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\HFP\PhoneSpecificMasks\<Manufacturer>\<Model>	
Key	Description
SKIPCERTAINPBSTORE	Allows the HFP service to ignore certain AT PB phone stores during AT phonebook download.
SKIP_AT+NREC	Does not send the AT+NREC=0 command on the opening of the HFP port. Can be overridden by HFPSetNRECOption.
TIMEOUT_AT_CPBS	Addresses some phones timing out on the AT+CPBS.
TIMEOUT_AT_CPBS_2	Provides second chance for phones that time out on AT+CPBS.
AEC_REGOVERRIDE	Allows the AEC registry settings at HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\HFP\PhoneSpecificMasks\<manufacturer>\<model name>\AECReg to be overridden by phone specific settings. Note that <model name> is optional.
USESPPFORATPB	Uses the SPP when downloading a phone book from the mobile device using AT commands.
SKIPPBAP	Does not use the PBAP when exchanging the phone book, even if the mobile device supports it.
AVOIDESCO	Avoids using the extended synchronous connection oriented (eSCO) protocol with the mobile device. Use the SCO protocol instead.
NOCLCC	Does not use the AT+CLCC command to determine the call status.
NOMAP	Does not use the MAP with the mobile device.
NOSYNCHP	Does not use Synch profile even if the phone supports it.
NO_SYNCML_DEVINF	Prevents Windows Embedded Automotive 7 from sending the device information (Devinf) SyncML flags to request specific vCard and vCalendar fields. Using this option prevents some SyncML issues.
NO_CONFCALL_END_CIEV	Does not send CIEV confirmation when ending a conference call.
SMSDOWNLOAD_SKIP_SM	Skips SMS download from SIM memory.
SMSDOWNLOAD_SCANALL	Performs SMS download using the CMGL 'ALL' command.
MAPSMS_USENATIVE	Uses the native character set, the PDU, to send or download SMS messages.
DISABLE_INBANDRING	Ignores in-band ring support.

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\HFP\PhoneSpecificMasks\<Manufacturer>\<Model>	
Key	Description
MAPSMS_SKIPUPDATESTATUS	Does not update the read status for MAP.
SEND_NREC_ON_SCO	Sends an NREC command when using SCO.

Windows Embedded Automotive 7 reports compatible behaviors for each paired device via the HFPPHONECAPS API as determined by the phone services. These values are saved in the device pairing key in the registry and can be modified by advanced users on a per-pairing basis.

Table 18 lists the key values that can be assigned to an individual phone.

Table 18: Individual phone key values

Key Value	Description
eThreewayCallSupport	Phone supports three-way calls.
eCallsetupSupport	Phone supports call setup.
eATPBSupport	Phone supports AT phonebook download.
eSMSReadSupport	This setting can have three values: <ul style="list-style-type: none"> • 2 – unknown • 1 – supported • 0 – not supported The value is unknown until the functionality has been successfully exercised.
eSMSReceiveNotify	This setting can have three values: <ul style="list-style-type: none"> • 2 – unknown • 1 – supported • 0 – not supported The value is unknown until the functionality has been successfully exercised.
eSMSSendSupport	This setting can have three values: <ul style="list-style-type: none"> • 2 – unknown • 1 – supported • 0 – not supported The value is unknown until the functionality has been successfully exercised.
eSyncMLSupport	Phone supports SyncML.
eInBandRingingSupport	Phone supports in-band ringing.

Key Value	Description
eCallerIDSupport	Phone supports AT+CLIP.
eSMSCmdSupport	<p>This setting can have three values:</p> <ul style="list-style-type: none"> • 2 – unknown • 1 – supported • 0 – not supported <p>This value is initially it is set to 2. At the first connection, this value gets set to either 0 or 1 depending on whether or not the phone passes the SMS scan. In some application designs, if this value is 0, users cannot enter the SMS application. A value of 0 should not get changed to 1 via the API unless the phone passes SMS scan. A value of 1 should never get changed to 0 even if the phone later fails SMS scan.</p>
eSMSOffsetMEIndex	On some phones the ME index is incorrectly reported by the handset. This value notifies the application of the offset should the phone exhibit this issue.
dwRingEventAgedoutPeriod	This is the maximum ring period of the phone as learned by the software. This is used internally in multi-call scenarios to improve accuracy.
ePBAPSupport	Phone supports PBAP as indicated by the existence of a PSE port.
dwBRFSupport dwCHLDSupport	These values are pass throughs for the phones BRSF and CHLD responses.

Connection Manager

The Connection Manager is the central component for managing connections on the Windows Embedded Automotive 7 platform. Connection Manager provides an API to let applications request connections, specify priorities, and close connections after use. It can be configured to manage platform network connections including embedded and Bluetooth-connected phones, wireless connections (WLAN), and Wi-Fi. In the Automotive 7 default configuration the Connection Manager is used to manage voice and data calls on the embedded phone, and voice and data calls on the Bluetooth-connected phone.

When an application requests a network connection, the following steps occur:

1. The Connection Manager first retrieves all the possible connections from a set of connection service providers (CSPs).
2. The Connection Manager then associates a set of costs with these routes and ultimately determines the optimal connection based on cost, latency, bandwidth, and other factors.
3. The Connection Manager queues the requested connection and uses the CSP to establish the connection at the appropriate time.

Figure 14 provides a schematic of the Connection Manager.

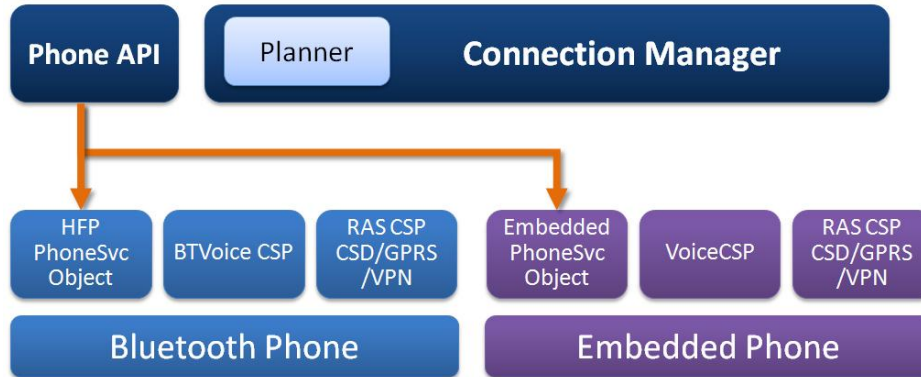


Figure 14: Connection Manager

The Connection Manager supports the following CSPs:

- **RAS CSP** provides General Packet Radio Services (GPRS) and dial-up connection support. When used with a Bluetooth phone, RAS CSP relies on the setup menu HMI for configuration of the dial strings.
- **Voice CSP** helps with the coordination of circuit switched data (CSD) and voice calls on an embedded phone. The HFP service calls into the voice CSP for operations that are related to voice calls on an embedded phone.
- **Proxy CSP** allows the insertion of proxy links between defined network destinations. It could be used to create a “virtual” destination that is used by applications logically linked to a “real” destination, which can then be reprovisioned without the need to change the applications.
- **Bluetooth Voice CSP** coordinates between data and voice calls on a Bluetooth phone. It keeps the Connection Manager aware of whether the Bluetooth phone is present or not. When a Bluetooth voice call occurs, Bluetooth Voice CSP creates a pseudo-Bluetooth voice connection so that attempts to create a CSD connection fail. Existing CSD connections are then disconnected, and GPRS connections are suspended.

Data Connectivity

Windows Embedded Automotive 7 supports applications that use a paired Bluetooth phone for data access. The Connection Manager supports the ability to share a mobile phone among various applications for different access modes through intelligent, priority-based management of the phone’s resources.

Automotive 7 provides the following data connectivity options:

- Bluetooth DUN
- Bluetooth Gateway Services (BGS)
- Virtual serial ports
- PAN

Bluetooth Dial-up Networking

Windows Embedded Automotive 7 supports the Bluetooth DUN Profile 1.1. This profile provides the ability to access the internet and other dial-up services through a Bluetooth-enabled phone.

The Bluetooth DUN profile defines two roles: Gateway (GW) and Data Terminal (DT). Automotive 7 devices operate in the DT role, whereas Bluetooth-enabled phones with internet access operate in the Gateway role.

Applications use the Connection Manager to establish a dial-up networking connection. The Automotive 7 device first calls **ActivateBTDevice** to establish a Bluetooth DUN connection to the Bluetooth-enabled phone. Once the Bluetooth DUN link is established, the Bluetooth-enabled phone serves in the Gateway role, or modem, for the Automotive 7 device, in the Data Terminal role.

The Bluetooth DUN profile depends on the SPP. SPP creates the Bluetooth link to the Bluetooth-enabled device, which acts like a wireless serial cable.

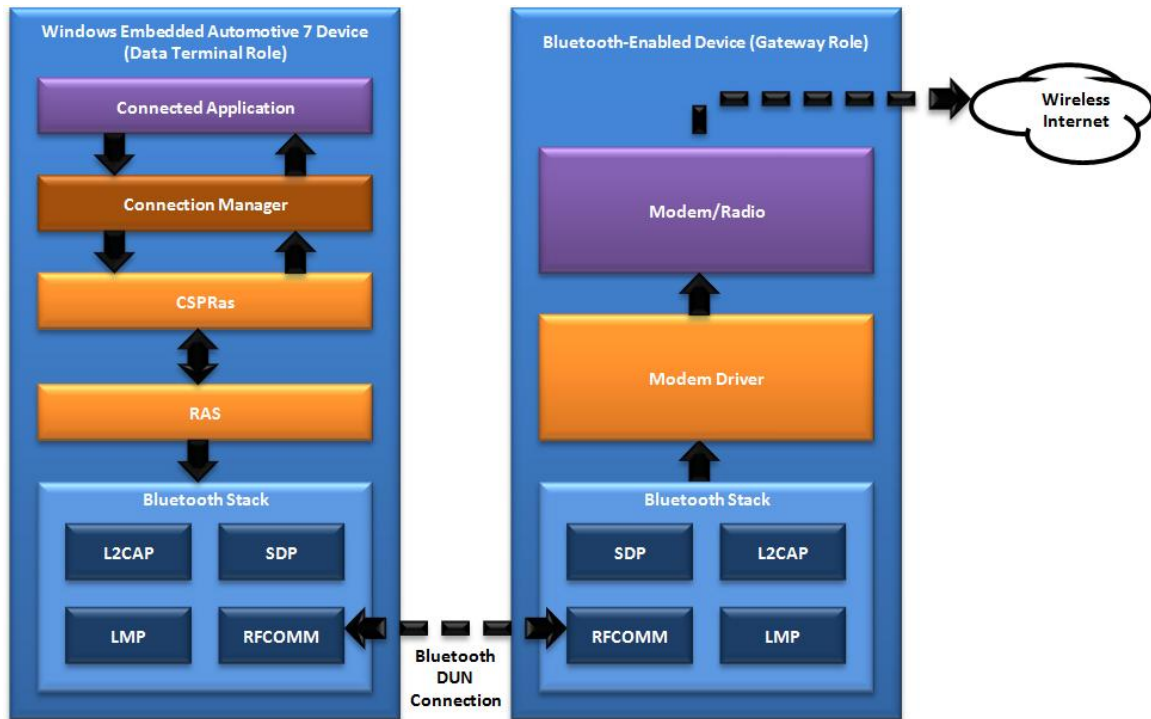


Figure 15: Bluetooth Dial-Up Networking architecture

Bluetooth Gateway Services

BGS allows Automotive 7 to expose connectivity to external Bluetooth devices for streaming National Marine Electronics Association (NMEA) data over serial ports. BGS runs in Services.exe and, at startup, creates a Service Discovery Protocol (SDP) record so that it can be discovered. It registers serial devices using port name, channel, and port number information read from the registry. Applications can access a serial port added by BGS just as they would access a regular serial port.

Personal Area Network

Windows Embedded Automotive 7 supports PAN profile networking, and in particular Personal Area Network User (PANU) 1.0. Bluetooth Device PAN provides wireless connections by enabling links between mobile devices, such as mobile phones and portable handheld devices. The PAN

service is a miniport network driver that creates network connectivity between two Bluetooth devices.

PAN can be configured via the PAN registry keys and key values, shown in Table 19 and Table 20.

Table 19: PAN registry key values

HKEY_LOCAL_MACHINE\Software\Microsoft\Bluetooth\pan	
Key	Description
ActivateOnBoot	Determines whether Bluetooth PAN is automatically activated when the device starts up. Note that when PAN is activated, an SDP record is registered and the application needs to stay loaded. If the application does not stay loaded, the Bluetooth PAN profile becomes undetectable.
Authenticate	Determines whether the connection is authenticated. Note that setting this value to 0 disables authentication, which can result in potential security risks.
Encrypt	Determines whether or not the connection is encrypted. Note that setting this value to 0 disables authentication, which can result in potential security risks.
InquiryLength	Specifies the length of the Bluetooth inquiry within the range of 1 to 255.
MediaDelay	Specifies the total length of time, in milliseconds, between two Bluetooth inquiries. Minimum value is 30,000 milliseconds (30 seconds).

Table 20: PAN registry key values

HKEY_LOCAL_MACHINE\Comm\BTPAN1\Parms	
Key	Description
AcceptConnections	Determines whether new connections are allowed for PANU and Group Ad hoc Network (GN) roles.
AdapterType	Specifies the type of adapters. Possible string values include PANU, Network Access Point (NAP), and GN.
ConnectionTimeout	Specifies the total length of time, in milliseconds, that a device waits for a connection to complete. If the specified time elapses before the connection completes, the connection is closed. The value can be set from 1,000 to 30,000.
Description	Specifies the service description to use in SDP records.
FriendlyName	Specifies the friendly name of the service to use in SDP records.
MaxConnections	Specifies the maximum number of simultaneous connections for GN and NAP roles.
SDP	Specifies a binary large object (BLOB) that contains the SDP record for the service. If this is not provided, the record is implicitly built.



Key	Description
ServiceID	Specifies the global service identifier as a standard globally unique identifier (GUID) string.

SIM Access Profile

Windows Embedded Automotive 7 provides support for SAP, a Bluetooth profile that makes the account information on a Bluetooth-enabled phone available for use by the phone module integrated into the Automotive 7–based device. The Windows Embedded Automotive SAP manager service takes advantage of the phone functionality that is provided by Phone Core and the Bluetooth Pairing Core and gives applications the ability to use SAP connections.

In Automotive 7, the embedded automotive device functions as an SAP client and connects to a Bluetooth-enabled phone (which functions as an SAP server) over an SAP profile connection. When this connection occurs, both devices enter SAP mode. The Automotive 7–based device then uses the account information that is provided by the SIM module to make calls from the Automotive 7 phone module.

The advantages of using SAP include:

- Improved reception using the antenna that is built into the car’s phone module.
- No need to hard code or manually enter account information in the car’s phone module to use an existing mobile phone account.
- Reduced driver distraction because the user does not interact directly with the Bluetooth-enabled phone used as an SAP server. Driver attentiveness can also be improved by the integration of microphones and speakers into the car, together with AEC/NS.

The disadvantages of using SAP include:

- The user’s phone UI is locked when in SAP mode, and information on that phone is unavailable. This helps prevent driver distraction, but may dissatisfy some users.
- All SMS messages received when SAP is connected are stored on the phone module integrated with the Automotive 7–based device. There is no widely used mechanism for transferring these messages back to the phone.
- No phone data that would normally be accessed using HFP AT commands is available in SAP mode.

Media Core Deep Dive

With Windows Embedded Automotive 7 Media Core, users can use intuitive user interfaces to enjoy their media—whether stored locally on the Automotive 7 device or through connected media players or Bluetooth-enabled phones. Media Core provides OEMs a robust set of APIs and services that access and control stored media and media metadata.

Media Core Architecture Overview

Media Core provides a uniform interface for indexing and using media from mass storage devices (MSDs), MP3 players, Apple devices (including the iPhone), media transfer protocol (MTP) devices, and media-capable Bluetooth phones.

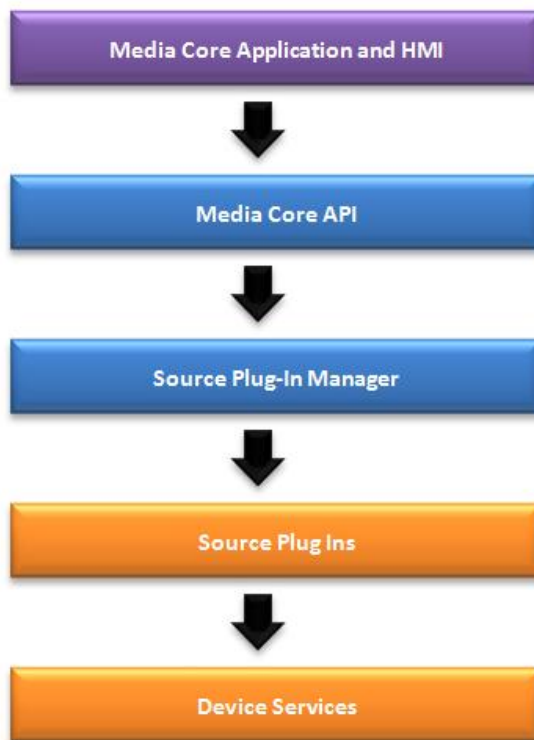


Figure 16: Media Core logical architecture

An OEM must always develop the media player application and human machine interface (HMI) layer at the top of the Media Core architecture. Automotive 7 provides a sample application (MediaPlayerSample) that demonstrates how OEM developers can use the API set, but the sample application is not intended as an example of excellent in-vehicle interface design.

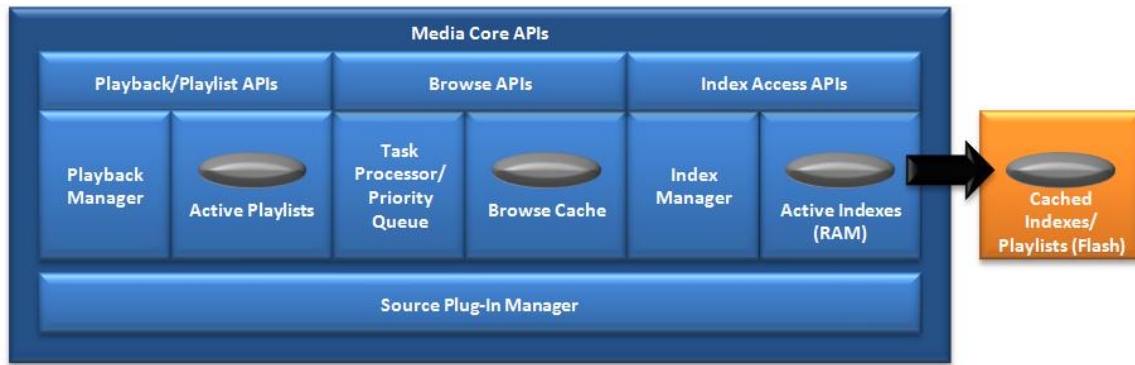


Figure 17: Media Core APIs

Another layer, the Media Core API, is delivered as part of the Windows Embedded Automotive 7 platform. Developers can configure and plug into it, but cannot change it because it is a binary. The Media Core API layer comprises three parts: Playback, Browse, and Index Access APIs.

The Playback API set offers device-agnostic command and control. It includes *play* commands (**MediaPlayByFile**, **MediaPlayByFileEx**, **MediaPlayByIndex**) and *control* commands (**MediaPlayControl** which offers **Stop**, **Pause**, **Resume**, **FFwd**, **Rewind**, **StopFFRwNext**, **Previous**).

The Browse API set (**MediaBrowse**) offers access to the contents of an attached device while the device indexes, with additional flexibility for different HMI behaviors. An index is built in the background, depending on the device type. Once the index is complete, the browse APIs seamlessly switch to use the index data for even faster performance. The index is cached for future connections to the device. The API uses a set of heuristics to determine how much of a previously connected device's content has changed and, based on the outcome, will reload the index from flash memory, update the index, or recreate the index automatically.

The Index Access API set is the legacy Media Core API set that relies on a completed index before the application can directly access the device. Much like the Browse API set, these indexes are cached and reloaded automatically when a known device is reconnected.

Microsoft will add new Media Core functionality and extension to the Browse APIs, but the Index APIs will be deprecated in a future release. OEMs should choose the Browse APIs for all new development, and consider updating existing applications to Browse APIs.

All three of the API sets interact with the source plug-in manager, which enables the device-agnostic aspects of the Media Core design. The source plug-in manager is not a separate module—it is a Media Core layer beneath the APIs. Like the Media Core API layer, the source plug-in manager is delivered as part of the Windows Embedded Automotive 7 platform. Developers can access it but cannot change it directly.

The source plug-ins, another layer of the Media Core architecture, provide playback, browsing, and indexing across a variety of devices. Windows Embedded Automotive delivers some standard components at this layer, but developers can add components to bring additional functionality.

Finally, the device services layer of the Media Core architecture contains service modules for devices that are supported out of the box. As with the source plug-ins layer, OEMs can access the Windows Embedded Automotive 7 platform to extend or add functionality to the device services layer.

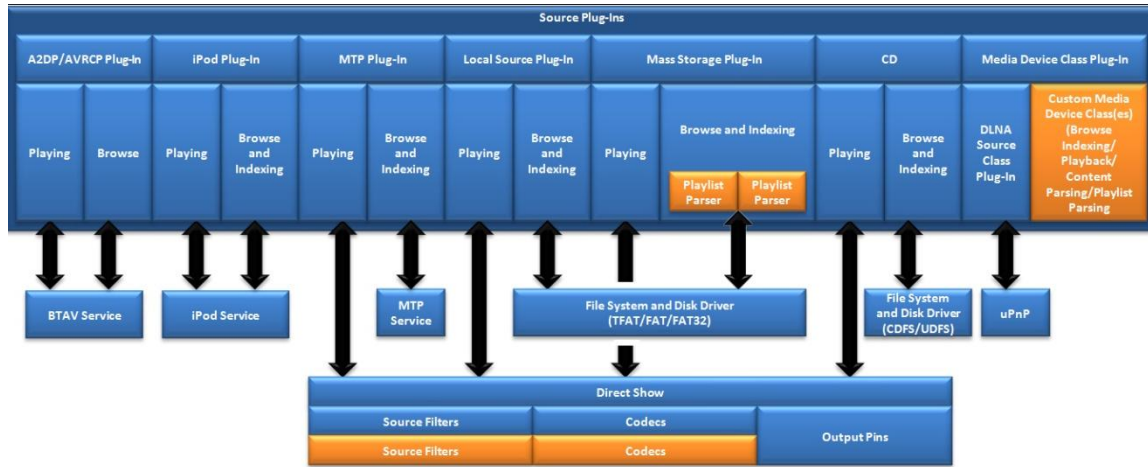


Figure 18: Media Core source plug-ins

Media Source Plug-ins

Illustrated in Figure 18, the media source plug-ins provide Media Core with the necessary access to the device to enable playback, browsing, and indexing across a variety of device types. These plug-ins support the following types:

- A2DP/AVRCP: Supports Bluetooth devices
- iPod
- MTP: Supports MTP-based devices
- Local source
- MSDs, such as USB flash drives
- Compact Disc (CD)
- DLNA: Supports connecting to a Digital Media Server (DMS) or Mobile Digital Media Server (M-DMS) devices to act as a Digital Media Player (DMP)
- Custom media device classes: Supports additional sources, such as devices based on future media protocols, and provides a standard interface to expose indexing and playback to the Media Core

Some devices (such as iPod and MTP-based devices) directly return the metadata and playlists for audio tracks using their protocol. However, for MSDs developers can write and register a DLL to extend the metadata or playlist parsing capabilities and provide metadata parsing for file types that are not supported out of the box. Each file name extension (such as .mp3) is associated with a file parser, encapsulated to prevent conflict between Media Core and the media file format.

Each metadata parser is a COM object associated with a registered file name extension and a registered class identifier (CLSID). On startup, the mass storage class MSC plug-in reads the registry and generates a list of known file name extensions. When an MSD or a direct mass storage device (DMSD) source plug-in reads a file, the plug-in creates a metadata parser on demand. Playlist parsers work in much the same way.

Supported Technologies

Media Core supports common media technologies and is responsible for actual media playback, metadata indexing, hardware event handling (for example, power and speech), and maintaining a “now-playing” list with history and shuffling ability. Supported technologies include:

- **Zune:** Windows Embedded Automotive 7 offers a software add-on package that lets a device fully interact with all the available Zune devices for audio content playback through a USB connection. Zune support includes full support for all digital rights management (DRM)-protected content that is purchased through the Zune Marketplace or obtained through the Zune Pass subscription service.
- **iPod/iPhone:** Older generation iPod models are supported through two-wire style connections using USB Serial. Newer generation models that support a one-wire connection using USB HID with Apple authentication hardware are also supported. Playback of all audio content, even from the iPhone and iPod Touch (including content that is protected by the FairPlay DRM mechanisms) is supported. Browsing and playback of video from Apple devices is also supported, but requires an OEM to acquire the Apple authentication hardware directly from Apple.
- **Mass storage file allocation table (FAT) and FAT32 file systems:** Media Core lets a user bring digital audio that is not DRM protected into the vehicle on MSDs, such as USB storage devices and SD cards.
- **MTP-based devices:** MTP refers to the communication protocol used to communicate with a variety of media players over the USB connection. Automotive 7 supports MTP-based devices from companies such as Sansa, Creative, and iRiver (including the DRM-protected content on those devices) with a software add-on pack.
- **Local storage:** Media Core supports playback of digital audio and video stored on the platform’s local storage device, which is typically flash memory.
- **CD-ROM:** Media Core supports Red Book audio (standard uncompressed format) and can transfer music to the Windows Media Audio (WMA) format. It also supports data CD compressed audio, CD-ROM file system (CDFS), and user-defined File System (UDFS). Media Core provides an API that enables ripping CD content that lets users play back ripped content while ripping is in progress. Ripped CD content is stored on fixed, local storage and is accessible as an MSC source. Media Core natively supports CDTEXT metadata and offers a plug-in to support other metadata sources accessed through a look-up API. A WMA encoder is provided in the platform and features encoding speeds of about 1.8x. Developers can replace this native encoder by adding a new encoder DirectX Media Object DMO.

- **Digital Living Network Alliance (DLNA):** DLNA defines a standard for transferring movies, photos, music, and other media from one device to another. DLNA servers can store media in one location and stream the media to DLNA-compliant players without any setup or configuration. Media Core acts as a Digital Media Player (DMP) attached to a Digital Media Server (DMS) or Mobile Digital Media Server (M-DMS) to consume audio content.
- **Bluetooth A2DP and Audio/Video Remote Control Profile (AVRCP) Bluetooth profiles:** Media Core can enable playback of music wirelessly from phones and other devices that support these Bluetooth profiles.
- **Supported media formats:** Media Core can access, index, and play WMA, MP3, PCM, WAV, and AAC files. Media Core also supports playlists, including those in Moving Picture Experts Group Audio Layer 3 Uniform Resource Locator (M3U), Advanced Stream Redirector (ASX), and Windows Media Player Playlist (WPL) formats, in addition to the native formats that are supported on iPod and Zune devices. New playlist formats and codecs can be added through the Media Core extension models.
- **Album art:** Supported album art formats include embedded metadata, folder.jpg, iPod/iPhone, Zune, and MTP.

Media Core can query custom metadata, add custom metadata to the index, use a metadata file parser plug-in, and direct access to services and devices. Additionally, new codecs can be added, new playlist formats can be supported, and entirely new classes of devices can be added to Media Core by using the extensibility model.

Media Feature Enhancements

The Media Core provided with Windows Embedded Automotive 7 provides a number of enhancements over previous versions of Microsoft Auto.

DLNA provides the ability for media, such as music, pictures, and video, to be transferred between multiple devices. Media can be stored on DLNA servers and then streamed to DLNA-compliant players without any setup or configuration. Automotive 7 supports connecting to a DLNA server to act as a digital media player.

Automotive 7 Media Core also adds and enhances support for Apple iPhone/iPod Touch accessory protocol technologies, including changes made in iOS 4 that support new technologies such as iDPS and the extended application framework (EAF) protocol. iDPS is a new authentication and handshake protocol supported on iPhone/iPod Touch firmware 3.x and higher. EAF requires iDPS and allows application-to-accessory communications. For example, EAF could enable a telediagnosis application that could upload vehicle information to an application on an iPhone for processing or forwarding to a technical support server.

Also new in Media Core is the ability for multiple applications to access the media index. A secondary application can access the index for MSD devices and receive notifications from any device types supported through Media Core.



Building a Media Application

Media applications use Media Core to abstract and manage device-specific interactions, such as indexing and playing. Automotive 7 includes a sample media player that OEMs can use as a foundation for creating a media application.

Sysgen Variables

Before you create an application, make sure the following Sysgen variables are set to include Media Core in your OS image:

- **SYSGEN_AUTOMEDIA_GROUP1**
- **SYSGEN_AUTOMEDIA_DSHOW_AUDIO_CODECS**

Browsing the Content

You can determine whether the user experience will be driven by buttons on the display or by voice, depending on your design requirements. Note that Microsoft recommends that OEMs use the Browse APIs because the Index APIs will be deprecated in a future release.

The following steps provide an example of how a media application is structured to allow media browsing.

1. Initialize the media application with the function **MediaInitializePlayer**.
2. Wait for the **WM_STORAGE_INSERTED** message to be passed once a media device is attached.
Once this message is received by the media application, the customer can then start the Browse process.
3. Call **MediaBrowseOpen** to get a session handle, and then use that handle to select and retrieve items.
4. Call the following functions to browse your media application:
 - **MediaBrowseSelect**: Changes the directory.
 - **MediaBrowseGetItems**: Retrieves items in the selected directory.
 - **MediaBrowseNowPlayingPlayEx**: Builds the “Now Playing List” and starts playing the media (either a folder or a file) the user has selected.
5. Call **MediaBrowseClose** to close the browse session.

Note: These functions provide a starting point for developing a media player application. The Browse APIs provide additional functions to meet your specific application requirements.

Playback

You can use the following control functions to automate stop, resume or fast forward for your media application:

- **MediaPlayControl:** Pass commands to this function.
- **MediaGetCurrentStatus:** Retrieves the structure that has the current status and the position of the current track. If you want to display the artist and title, use this function to display metadata.
- **Media Core Messages:** You can use the following to get current play status (same as Playback Update):
 - **WM_TRACK_STARTED:** Passed when each track is started.
 - **WM_TRACK_COMPLETE:** Passed when each track in a playlist is finished.

For information about other Windows messages, see [Media Core Windows Messages](#).

Album Art

The album art functionality lets OEMs create interfaces that present album art to users. Album art metadata groups and displays tracks by album, such that a media player application can display album art for tracks from the current connected device.

The following steps provide an example of how to display album art:

1. Call the **MediaBrowseGetAlbumArtCount** or **MediaNowPlayingGetAlbumArtCount** function to get the count, if album art is available. If album art is available, then go to Step 2.
2. Call the following functions to get information about the image, such as the type of image:
 - **MediaBrowseGetAlbumArtInfo**
 - **MediaNowPlayingGetAlbumArtInfo**
3. Call either the path or data functions, depending on the type of device:
 - If the device is an MSD device, use the path functions **MediaBrowseGetAlbumArtPath** or **MediaNowPlayingGetAlbumArtPath**.
 - If the device is an iPod, use data functions **MediaBrowseGetAlbumArtData** or **MediaNowPlayingGetAlbumArtData**.

Note: For iPod devices, you can only retrieve art from the currently playing track.

Device Tips

When Media Core requests a file to be played, the request is directed to a connected device. Each connected device has unique behavior.

Consider the following when you are working with devices:

- Indexing starts when a user plugs in the device.
- Use the same set of APIs for all devices.
- Use Browse API commands.
- iPod devices have a category-based hierarchy model (genre/artist/album/track); therefore, browse API commands work as soon as the device is connected.

- Browsing by categories on iPod devices is not supported when the selection order is lower to higher (for example, album, artist, and genre). There is no workaround for this.
- Other types of devices have a file/folder hierarchy model. Therefore, you must wait until indexing is complete before calling browse API commands. If you issue a browse command before the connected device is indexed, the media player will generate an error message.
- If a user removes a CD or the power/ignition is cycled, the CD will begin playing at the first track, regardless of where it was. To work around this issue, save the track number and position in the track across ignition cycles. When the CD begins playing again, play the specific track by using the function **MediaNowPlayingPlayEx**.

Media Core Configuration

Developers can configure and customize Media Core settings and behavior through the following registry keys.

- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\MSD
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\MTP
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\MultiApp
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\iPod
- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\Fields\- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\Fields\- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\LocalSource
- HKEY_LOCAL_MACHINE\Drivers\BuiltIn\IPDSvc
- HKEY_LOCAL_MACHINE\Services\IPDSvc
- HKEY_LOCAL_MACHINE\Services\IPDSvc\AccessoryInfo
- HKEY_LOCAL_MACHINE\Services\IPDSvc\AccessoryInfo\Protocols
- HKEY_LOCAL_MACHINE\Services\IPDSvc\MetadataEx\- HKEY_LOCAL_MACHINE\Drivers\BuiltIn\IPDSvc1
- HKEY_LOCAL_MACHINE\Drivers\BuiltIn\IPDSvc2

You can find information on the specific key values in

Appendix 3: Media Core Registry Settings.

Media Core Windows Messages

Media Core features a full set of messages it can send to registered applications. The messages include SOURCEID, which is an important element that comes with many of the messages in the IParam argument. SOURCEID allows the application to accurately map the message to the source.



Table 21 lists the messages provided by Media Core.

Table 21: Media Core messages

Message	Description
WM_INDEXING_DONE	Posted when all indexing has been completed.
WM_MEDIA_COMMAND_ERROR	Posted when a MediaPlayControl command is not implemented by the receiving source class type, or when the command failed to execute.
WM_MEDIA_COMMAND_NOTIMPLEMENTED	Posted when a MediaPlayControl command is not implemented by the receiving source class type, or when the command failed to execute.
WM_MEDIA_CONNECTION_FAILURE	Posted when the MediaInitiateDeviceConnection function is called to connect to a specific remote device, but the connection fails.
WM_MEDIA_CORE_ERROR	Posted when an asynchronous error occurs.
WM_MEDIA_DUMP_INDEX	For internal use and testing only.
WM_MEDIA_FOUND	Posted as soon as media files are detected on the attached media storage device.
WM_MEDIA_INITIALIZED	Posted when Media Core has been initialized with a call to MediaInitializePlayer .
WM_MEDIA_NO_CONNECTIONS_AVAILABLE	Posted when MediaInitiateDeviceConnection is called with a search parameter specified, but no devices are available to connect to.
WM_MEDIA_PLAYBACK_HOOK	For internal use and testing only.
WM_POWER_FULL	Posted when Media Core is working at full power.
WM_POWER_LOW	Posted when Media Core has detected a low power condition and will be suspending.
WM_STORAGE_AVAILABLE	Posted as soon as a device is inserted and before any internal consistency checks are performed.
WM_STORAGE_FULL_POWER	Posted when a media source has been identified and is ready for use after a power state change.
WM_STORAGE_INSERTED	Posted when a storage card is inserted.

Message	Description
WM_STORAGE_LOST	Posted when an index is lost because the media source for the index was removed, and the original index was overwritten by another index from a recently inserted device.
WM_STORAGE_LOW_POWER	Posted when a media source is no longer available because of a low power state.
WM_STORAGE_REMOVED	Posted when a storage card is removed.
WM_TRACK_COMPLETE	Posted when each audio track in a playlist is finished.
WM_TRACK_ENDOFPLAYLIST	Posted by some device types in autoplay mode when there are no more audio tracks to play.
WM_TRACK_PAUSED	Posted when an audio track is paused.
WM_TRACK_RESUMED	Posted when a paused audio track is started again by a call to MediaResume .
WM_TRACK_STARTED	Posted when an audio track is started.
WM_TRACK_STOPPED	Posted when an audio track is stopped by a call to MediaStop .
WM_TRACK_SYSTEMMUTE	Posted while in shared mode when the playback manager pauses playback in the front zone because the speech service is currently busy (see SSN_BUSY).
WM_TRACK_SYSTEMPAUSED	Posted when the playback manager detects a loss of primary or secondary stereo.
WM_TRACK_SYSTEMRESUMED	Posted when the playback manager gets a resume request that undoes a system pause state (indicated previously by a WM_TRACK_SYSTEMPAUSED message).
WM_TRACK_SYSTEMUNMUTE	Posted while in shared mode when the playback manager resumes playback in the front zone because the speech service is idle (see SSN_IDLE).
WM_BROWSE_ENUMERATION_DONE	Posted by MediaBrowseGetItems when the enumeration of a browse item is finished.
WM_BROWSE_NOWPLAYING_BUILT	Posted by MediaBrowseNowPlayingPlay when the NowPlaying playlist is built.
WM_BROWSE_GETMETADATA_DONE	Posted by MediaBrowseGetPlayingMetadataItems when the metadata values for the currently playing media are available.



Custom Media Device Class

The custom media device class in Windows Embedded Automotive gives additional sources the ability to plug into Media Core by providing standard interfaces to expose indexing and playback to Media Core. Additional sources that take advantage of this capability might include DLNA-certified devices and devices based on future or proprietary protocols.

This feature provides the ability to developers to create optional extensions for applications. For example, developers could create a content parser to extract metadata or could enable playlist parsing to interpret playlist formats. Developers could also create a service that interacts directly with the device, similar to the MTP or iPod service Media Core provides.

The interfaces for a custom media device class are defined in `%WINCEROOT%\public\automediamedia\sdk\inc\mediacustomsource.h`.

Each custom media device class is registered with Media Core in the registry, as shown in the following example: `[HKEY_CLASSES_ROOT\Mediacore\Devices\<NameOfDevice>]` where `<NameOfDevice>` is the device name. The default value for the key is the CLSID of the COM object.

Media Core implements a custom, media-device class as a COM object. In addition, you must implement the plug-in by using the following interfaces:

- IDevice
- IDeviceBrowse
- IDeviceHost
- IDeviceIndexer
- IDevicePlugin
- IDevicePlayback

Custom File and Playlist Parser

The custom file and playlist parser brings additional indexing and browsing functionality to the mass storage media plug-in by extending the existing file type support. It is applicable to the mass storage class only.

File Parser

The file parser extracts metadata from image files and maps that data to standard Media Core metadata fields. The file parser could set one of those fields, such as Genre, to a standard string, such as "X-JPEG" or "X-GIF." The media player application could then easily filter these genre fields when working with media, and use the genre fields when in picture viewer mode.

Each metadata parser is a COM object whose associated file extension and CLSID must be registered so Media Core knows how to create it. On startup, Media Core reads the registry and generates a list of known file extensions. When a file is read by the MSD or DMSD source plug-in, Media Core creates a metadata parser on demand (at most, once per device insertion) and lets the parser parse the file.

To support the DMSD source plug-in, Media Core first tries to call a parse function that takes file data as the input parameter. If the parser does not support such an API, Media Core then tries

to call a parse function that takes the file path as an input parameter. If both parse functions fail, Media Core tries the next parser of the file extension, if there is any.

Each metadata file parser must implement at least one of **IMediacoreMetadataBufferParser** and **IMediacoreMetadataFileParser**. These interfaces are defined in `public\automediamedia\oak\inc\mediacoremetadataparser.h`.

Playlist Parser

The playlist parser interprets playlist format. A playlist parser must create a special value in its registry entry called Playlist. If a registry entry for a file extension contains a value by the name of Playlist, Media Core treats this file extension as a playlist extension. When the MSD plug-in encounters a file that it does not recognize, it checks with the parser manager to see if a parser is registered for the file. If a parser is registered, then it is invoked to parse the file and feed the file content to Media Core so that the file can be indexed. If a parser is not found, then the file is skipped.

Each playlist parser is a COM object, and the registry entry for the file extension contains the COM object's CLSID so that Media Core can instantiate it.

The custom playlist parser must implement the **IPlaylistParser** interface, which is defined in `public\automediamedia\oak\inc\iplaylistparser.h` and contains functions including **OpenPlaylist**, **GetPlaylistTitle**, **GetFirstFileinPlaylist**, **GetNextFileinPlaylist**, and **ClosePlaylistHandle**, that are called by Media Core so that Media Core can read the playlist through the custom parser.

Album Art

Windows Embedded Automotive 7 supports iPod album art for the now-playing item only, due to a limitation of the iPod device. On MTP, mass storage, and Zune devices, Media Core supports album art on all now-playing and indexed items if indexing is complete.

Metadata Plug-in API

The metadata plug-in API supports metadata functionality during both CD ripping and playback. CDTEXT is supported out of the box, and developers can extend the API to include a built-in or service-based metadata database.

The metadata provider is a COM object that can be registered with Media Core. When track or CD information is available Media Core calls the provider, which can look up data (preferably asynchronously). The provider can then call back into Media Core using the local source edit APIs to update the metadata.

The following is the CD audio flow:

1. An audio CD is inserted and indexed, although the initial index is sparse.
2. At the end of indexing, the CD audio source class requests the appropriately registered metadata provider to resolve the CD.
3. The **WM_INDEXING_DONE** message is sent to the application. Meanwhile, the metadata provider retrieves the metadata and uses Media Core APIs to update the metadata for those tracks in the index.



4. As soon as track metadata is updated, the **WM_UPDATE_METADATA_COMPLETE** message is sent to the application.

The COM object must implement and expose the **IMediametadataProvider** interface. This interface is defined in `public\automeia\oak\inc\mediametadataprovider.h`.

These interfaces provide Media Core the ability to interface with the COM object. In addition, the COM object uses some public Media Core APIs to complete the metadata process. These APIs are: **MediaBeginEditMetadata**, **MediaEditMetadataItem**, and **MediaEndEditMetadata**.

These interfaces are defined in `public\automeia\oak\sdk\inc\automeiacore.h`.

iPod Application to Accessory Communication

Windows Embedded Automotive 7 supports iPod device application-to-accessory communication. This communication provides the capability for an application that resides on an Apple iPod device to communicate with an application on the Automotive 7 device. The iPod must have 3.x firmware and above and must be connected to the Automotive 7 device using the standard iPod 30-pin-to-USB connector (1-wire connection using USB HID).

A **BundleSeedID** must be registered in order to activate the application-to-accessory functionality. This value is set with the **BundleSeedIDPrefToken** key under the **HKEY_LOCAL_MACHINE\Services\IPDSvc** registry key. Supported protocols for application-to-accessory communication must be registered in the **HKEY_LOCAL_MACHINE\Services\IPDSvc\AccessoryInfo\Protocols** portion of the registry in the **ProtocolTokenX** key with the value `<protocol id string>`. In order to communicate with the iPod service directly the application must use the **MediaGetDeviceHandle** API to get a handle to the device.

To communicate with an iPod device:

1. Call the **CreateMsqQueue** function, and then create a queue for iPod service messages.
2. Call **IPSRegisterForNotifications** to receive a notification when the user attaches the iPod device to the USB port.
3. After an **IPS_MSG** message of type **IPM_IPODATTACHED** is received, call **IPSOpenDevice** to start working with the attached iPod device. If the iPod device is already attached when you call **IPSRegisterForNotifications**, this message will be sent to the message queue immediately.

After the iPod device connects, the application receives an **IPM_IDPSCOMPLETED** notification that indicates whether the authentication protocol handshake was completed. If **wParam** is set to **TRUE**, the application can receive open session requests from the iPod device. If it is set to **FALSE**, the device does not support the Apple proprietary authentication protocol.

To subscribe to an application-to-accessory session request notification for a supported protocol, the application must call **IPSWatchA2ASessions**. Only one application can be

subscribed for notifications for a specific protocol. To unsubscribe, the application must call **IPSunregisterForNotifications**.

After an application requests a session with Windows Embedded Automotive 7, the subscribed application receives an **IPM_OPENSESSION** message with **wParam** set to **SessionID** and **lParam** set to **ProtocolIndex**. The application can then open an application-to-accessory session by using **IPSOpenA2ASession**. The application must provide a message queue that will be used to receive notifications and data from the iPod device. To send data from the application to Windows Embedded Automotive, use **IPSSendA2AData**.

When the iPod device closes a session or the device is disconnected, the application that opened the session receives an **IPM_A2A_CloseSession** message. The application should honor any **IPM_A2A_Notification** message that instructs the application not to send any requests during a specified time period.

If an iPod application supports streaming media applications in Windows Embedded Automotive 7 information can be passed to the iPod service about the current playback state by using the following APIs:

- **IPSGetsSetDigitalAudioStreaming()** to start or stop audio streaming.
- **IPSSetPlaybackStatus()** to notify Automotive 7 about playback status and information about the currently playing item.

iAP and MTP Passthrough

Passthrough capability for iPod accessory protocol (iAP) and MTP gives any application the ability to directly call any iAP or MTP interface. Since passthrough mode is a state transition in the service, developers must use it very carefully. An application should always wait until any browsing functions are complete and be aware of any other interactions that may be going on with the iPod.

Considerations for iPod

iPod passthrough is only available to one-wire iPod connections that have already been authenticated. The application must get a handle to the device through the iPod Service.

The service handles are found through the following APIs:

- **MediaGetDeviceHandle**
- **MediaBrowseGetItemObjectHandle**
- **MediaGetItemObjectHandle**

These APIs are defined in `public\automedia\sdk\inc\automediacoreex.h`. These handles can be used to directly issue iAP commands through the iPod Service using the passthrough APIs. These APIs are defined in `public\automedia\sdk\inc\ipdsrv.h`.

The following are the iPod Service interfaces for passthrough:

- **IPSEnterPassthroughMode**
- **IPSExitPassthroughMode**
- **IPSSendPassthroughData**

Considerations for MTP

To control devices that use MTP an application must get a handle to the device and control the device through the MTP service. The service handles are found through the following APIs:

- **MediaGetDeviceHandle**
- **MediaBrowseGetItemObjectHandle**
- **MediaGetItemObjectHandle**

These APIs are defined in `public\automedia\sdk\inc\automediacoreex.h`.

Applications can issue commands directly to the device handle using MTP IOCTLs. All of the standard, extended, and proprietary MTP IOCTLs are supported. Unlike iPod devices, there is no passthrough state when calling directly to an MTP device. An application could cause performance or other issues if using passthrough while the MTP device is being heavily used for other functions, such as indexing.

Device Lab

Microsoft has created a device lab to help facilitate compatibility between Windows Embedded Automotive 7 and a growing number of media devices, including devices under development by automakers. The Device Lab tests approximately 400 phones and 150 media players each year to ensure that the Windows Embedded Automotive platform works with the latest devices. Each release of the Automotive platform includes test reports and feature compatibility statements for each supported device.

Staff members from the Device Lab select devices for testing from markets in North America, Europe, and Asia based on several elements, including:

- Popularity/sales figures
- Mobile operator distribution and supported technologies
- OEM/customer request
- Compatibility complaints from end users

The Device Lab staff considers devices to be compatible devices if there is a test report listed for the device on the Automotive results website on or before the release date of each new version of the Automotive platform. A compatible device should function as reported on the results website.

Appendix 4: Compatible Devices lists the compatible devices as of this document's publication date.

During extensive testing in the Device Lab, Microsoft engineers make a reasonable effort to work around device issues. The engineers inform device manufacturers when incompatibility and specification compliance issues are discovered during testing. If problems with compatible devices are discovered, the Microsoft Auto QFE process is used to address them.

Microsoft provides device interoperability testing and software patches so that devices that may not strictly adhere to standards function correctly with Automotive 7 products. Automotive OEMs benefit from the Device Lab testing, which provides the OEMs the ability to focus on other development areas.

Conclusion

Windows Embedded Automotive 7 provides a proven, highly reliable, and extensible software platform and hardware reference design on which automakers can distinguish themselves by building innovative solutions to help drive sales and customer loyalty. In particular, the Phone Core and Media Core components provide automakers, suppliers, and developers with the building blocks that they need to set themselves apart from the rest of the field while quickly and reliably creating a broad range of advanced in-vehicle solutions that meet the growing needs of automotive consumers.



Appendix 1: Globalization Features

In today's international business climate, OEMs with multinational product lines require a platform that can be used across the globe. Windows Embedded Automotive 7 is designed with globalization in mind, and supports multiple code pages and encodings for various global languages. This support for a wide range of character sets and languages makes Automotive 7 an ideal platform for creating global solutions.

Table 22 shows the supported locales and languages for Windows Embedded Automotive 7.

Table 22: Supported locales and languages

Locale	Languages	
Europe and North America	<ul style="list-style-type: none"> US English Canadian French French Italian German Spanish Dutch Danish Swedish Norwegian 	<ul style="list-style-type: none"> Finnish Portuguese for Portugal Portuguese for Brazil Turkish Polish Czech Slovak Russian Greek Hungarian
Asia	<ul style="list-style-type: none"> Japanese Korean 	<ul style="list-style-type: none"> Chinese Traditional Chinese Simplified

Table 23 shows the code pages and encodings supported by Windows Embedded Automotive.

Table 23: Feature code pages and encodings

Feature	Code Pages/Encodings	
vCard	<ul style="list-style-type: none"> ASCII UTF8 BIG5 (Taiwanese) GB18030 (Chinese Simplified and Chinese Traditional) 	<ul style="list-style-type: none"> SHIFT_JIS (Japanese) EUC-KR (Korean) GB2312/GBK (Chinese Simplified)
SMS	<ul style="list-style-type: none"> 7-bit SMS PUD 	<ul style="list-style-type: none"> Text mode
HFP	<ul style="list-style-type: none"> UTF-8 ISO-8859-1 	<ul style="list-style-type: none"> GSM USC(2)
PBAP	<ul style="list-style-type: none"> UTF-8 	
Media Indexing	<ul style="list-style-type: none"> ISO-8859-1 UTF-16 	<ul style="list-style-type: none"> BIG5 (Taiwanese) GB18030 (Chinese Simplified and Chinese Traditional)

Feature	Code Pages/Encodings	
Media Playlist Parsing	<ul style="list-style-type: none">• UTF-8• Unicode big-endian• Unicode little endian	<ul style="list-style-type: none">• BIG5 (Taiwanese)• GB18030 (Chinese Simplified and Chinese Traditional)



Appendix 2: Bluetooth Pairing Service Registry Key Values

After a Bluetooth-enabled device has paired successfully to a Windows Embedded Automotive 7 device, information about the paired device is available in the Automotive 7 system registry.

Each paired device has a corresponding registry entry under **HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices**. This registry key contains subkeys for each Bluetooth profile supported, and a subkey named **\Attributes** which contains device-specific information.

To retrieve device attribute values stored in the registry, you can use the **GetBTDeviceAttribute** functions. You can retrieve or set attributes in the form of a DWORD, a BYTE array (BLOB), or a WCHAR.

The functions that retrieve device attributes are **GetBTDeviceAttributeDWORD**, **GetBTDeviceAttributeBLOB**, and **GetBTDeviceAttributeWCHAR**. The functions that set device attributes are **SetBTDeviceAttributeDWORD**, **SetBTDeviceAttributeBLOB**, and **SetBTDeviceAttributeWCHAR**.

Note: When the Windows Embedded Automotive 7 device is cold booted, the registry, including the paired device list, is cleared.

Table 24 lists the device attributes available under the **\Attributes** subkey.

Table 24: Device list attributes under the **\Attributes** sub-key

Sub-key	Value	Description
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\RingtoneOption]	DWORD	Indicates whether ring-tone option is present for notification of incoming calls.
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\AccessOrder_HFP]	DWORD	Indicates order in which paired phones are accessed for Hands-Free Profile (HFP).
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\SupportsATCOPS]	DWORD	Indicates whether device supports AT+COPS command, which is used to indicate whether the phone is connected to a network, or to change the GSM network.
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\ListCurrCalls]	DWORD	Indicates list of current calls.
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\HFPSMSSupport]	DWORD	Indicates whether device supports Hands-Free Profile (HFP) Short Message Service (SMS).
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\HFPPBSupport]	DWORD	Indicates whether device supports Hands-Free Profile (HFP) Phonebook (PB).

Sub-key	Value	Description
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\CHLDFeatures]	DWORD	Indicates CHLD features.
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\Manuf]	String	Indicates manufacturer of the paired device.
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\Model]	String	Indicates model of the paired device.
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\BatteryCINDIndex]	DWORD	AT+CIND command that indicates cell-indicator status for "battery".
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\CallheldCINDIndex]	DWORD	AT+CIND command that indicates cell-indicator status for "callheld".
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\SignalCINDIndex]	DWORD	AT+CIND command that indicates cell-indicator status for "signal".
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\Call_StatusCINDIndex]	DWORD	AT+CIND command that indicates cell-indicator status for "call_status".
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\RoamCINDIndex]	DWORD	AT+CIND command that indicates cell-indicator status for "roaming".
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\CallSetupCINDIndex]	DWORD	AT+CIND commands indicates cell-indicator status for "callsetup".
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\ServiceCINDIndex]	DWORD	AT+CIND command indicates cell-indicator status for "service".
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\CallCINDIndex]	DWORD	AT+CIND command indicates cell-indicator status for "call".
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\CINDCount]	DWORD	AT+CIND command indicates cell-indicator status for "count".
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\CINDData]	BLOB	AT+CIND command indicates cell-indicator status for "data".
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\BRSFeatures]	DWORD	Indicates Broadband Radio Service (BRS) features. BRS provides high-speed and high-capacity broadband service.
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\IsHFP10]	DWORD	Indicates support for HFP 1.0 spec.

Sub-key	Value	Description
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\PhoneCAP]	BLOB	Indicates phone capabilities.
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes>ContactAdded]	DWORD	Indicates whether a new contact has been added.
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\Exclusive]	DWORD	Indicates exclusive attribute value.
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\AccessOrder_BTAV]	DWORD	Indicates order in which paired devices are accessed for Bluetooth Audio/Video.
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BTPairSvc\Devices\<UniqueDeviceID>\Attributes\AccessOrder_PAIR]	DWORD	Indicates order in which paired devices are accessed.



Appendix 3: Media Core Registry Settings

Table 25: Media Core registry settings

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config		
Key Value	Type	Description
MountWaitTimeout	DWORD	How long the media player waits after an ignition on event before releasing audio focus back to the radio if no media is detected. If this time-out period is longer then there is more quiet time while no media is found. The default value is 0x1f40 (8 seconds).
PauseDelay	DWORD	The amount of time the system sleeps after the media player pauses for speech interaction (SSN_BUSY). Sleep time prevents overlap between media player output and speech output. A short sleep time may result in “loud” media playback. The default value is 0x0064 (100 milliseconds).
MaxPlaylistSize	DWORD	The maximum size of a playlist. For WPL parsing, the whole file must be loaded into memory, which means that increasing this value could cause out-of-memory errors. The default value is 1 MB.
MaxID3V2HeaderSize	DWORD	The maximum size for MP3 metadata sections. Very large MP3 metadata sections can cause slow performance, so for headers over this size, metadata is not parsed. You can increase or decrease this value. The default value is 1 MB.
IgnDebounce	DWORD	The amount of time that must elapse after an ignition on event before the system processes media player events. This value should be fairly short and is intended to help primarily with testing scenarios in which the ignition switch is toggled for long periods of time at fairly high rates. The default value is 2,000.
IgnFilterDebounce	DWORD	The time in which all devices must reattach after an ignition event. In the event that a device fails to reattach within this time, Media Core assumes that the device is no longer plugged in. The default value is 20,000.
FileCacheSize	DWORD	The amount of space to allocate for non-metadata media files. The default value is 1 MB.
MetadataCacheSize	DWORD	The amount of space to allocate for string data per table (for example, for artists, for albums, and so on). The default value applies to each string table.
GenreCacheSize	DWORD	The amount of space to allocate for genres. Generally, this amount is much smaller than the other space allocations. The default value is 50,000.
TotalIndexSize	DWORD	The maximum size for a single index. The default value includes 2.4 MB plus the FinalBuffer value.
FinalizeBuffer	DWORD	The amount of buffer that remains to leave some space for playlists. The current indexing algorithm only files metadata up to TotalIndexSize minus FinalizeBuffer bytes for file metadata. This remaining space exists to make sure that, even with very large devices, Media Core indexes playlists, which are always indexed towards the end of the indexing process. The default value is 15,000, which is 86 KB.
CurrentSrcId	DWORD	An internal counter to determine an order for device inserts. Do not modify this value.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config

Key Value	Type	Description
AllowableUsedSpace	DWORD	The threshold at which Windows Embedded Automotive dumps the old index because it contains too much stale information. This value is used when multiple Quick Scans are performed on the index. The default value is A (10).
SpaceDifference	DWORD	The space difference between two indexes. This is used to determine whether the current index is a possible match for a previously built index. This value is expressed in a percentage of space difference between two indexes, and accounts for changes that are made to the index—such as adding a new album—that result in space differences. Lower percentage values for this entry increase the constraints and are used to determine whether the indexes are the same. For example, if the percentage is changed from 10 percent to 5 percent, the probability that two indexes will be considered from the same device will decrease. The default value is 10 percent.
FileCheckDuration	DWORD	The amount of time that Media Core looks for files to determine whether the new disk is the same disk that was seen last time. The default value is 0x07D0 (2 seconds).
IdleDelay	DWORD	The amount of time to wait after a speech idle (SSN_IDLE) message is posted before Media Core resumes playback. The default value is 0x01C0 (448 milliseconds).
MaxIndexedSourceCount	DWORD	The maximum number of RAM sources, also known as full sources. These contain full sets of metadata. This controls the number of concurrently available supported devices. The default value is 5.
MaxCncCount	DWORD	The maximum number of command-and-control devices that are supported. Be aware that these sources are not persisted so this is the number of concurrent sources. A Bluetooth music device is an example of a command-and-control device. The default value is 1.
MaxPersistedCount	DWORD	The maximum number of full sources that are persisted to flash memory. The default value is 6.
ExtraDataCacheSize	DWORD	The maximum size of the extra data table in bytes. The default value is 1.3 MB, which is enough for 86,000 GUIDs. The default value is 150,000 (1.3 MB).
StringHashSize	DWORD	The number of pages to allocate each string metadata hash table (for Genre, Title, Artist, Album, and so on). The default value is 2.
FileHashSize	DWORD	The number of pages to allocate for the file hash table. The default value is 10 pages.
ExtraHashSize	DWORD	The number of pages to allocate for the extra hash table. The default value is 2 pages.
VideoSupport	DWORD	Whether Media Core can play video from video-enabled media files. The default value is 0 (no video).

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config

Key Value	Type	Description
PlaybackTimerDelay	DWORD	The number of milliseconds to wait after a play request before begin rendering media. This is a performance-tuning setting that enables the playback manager to better handle the action of the user quickly pressing a button that corresponds to playback controls such as Play, Next, or Previous. The default value is 0x00C8 (200 milliseconds).
RemovableMediaOnly	DWORD	Whether to index attached as well as removable media. 1: Media Core indexes removable media only. This is the default value. 0: Media Core indexes all attached media.
MaxIndexingThreadsCount	DWORD	The maximum number of concurrent indexing threads allowed. This controls the number of concurrently indexed devices and the number of index builder instances per media source. The default value is 3.
IndexingPriority	DWORD	Used for each indexing thread. This is one of the priorities from Winbase.h. Uses the default data if the key is missing. The default value is THREAD_PRIORITY_NORMAL.
IndexMgrPriority	DWORD	Used for each indexing thread. This is one of the priorities from Winbase.h. Uses the default data if the key is missing. The default value is THREAD_PRIORITY_NORMAL.
PlayPriority	DWORD	Used for each indexing thread. This is one of the priorities from Winbase.h. Uses the default data if the key is missing. The default value is THREAD_PRIORITY_NORMAL.
PlayMgrPriority	DWORD	Used for each indexing thread. This is one of the priorities from Winbase.h. Uses the default data if the key is missing. The default value is THREAD_PRIORITY_NORMAL.
ZoneFrontBus	DWORD	This value must be set to the corresponding arbitrator bus ID used for the front zone. Be aware that there are no checks made to make sure that ZoneFrontBus and ZoneBackBus are not the same. The default value is AUDIO_BUS_PRIMARY_STEREO. Uses the default data if the key is missing.
ZoneBackBus	DWORD	This value must be set to the corresponding arbitrator bus ID used for the back zone. Be aware that there are no checks made to make sure that ZoneFrontBus and ZoneBackBus are not the same. The default value is AUDIO_BUS_SECONDARY_STEREO. Uses the default data if the key is missing.
TotalBrowseSize	DWORD	The maximum size in bytes for the browse cache for a particular source device. The default value is 2 MB.
DisableAutoIndexing	DWORD	Disables automatic indexing. Set the value to 1 to disable automatic indexing when a device is inserted. If set to 0 then auto indexing is enabled. The default value is 0.
DisableAlbumartIndexing	DWORD	Disables album/CD cover art indexing. Set the value to 1 to disable album art indexing. If set to 0 then album art indexing is enabled. The default value is 0.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\MSD

Key Value	Type	Description
PreScanDuration	DWORD	The length of time that the media player collects track information before it starts AutoPlay. The collected track information becomes the complete selection of files available for AutoPlay. Increasing this length of time also increases how long you must wait to hear media playback. The default value is 0x0FA0 (4 seconds).
BrowseOption	DWORD	This value specifies how the browse APIs determine whether the device or the index tables should be used for queries. For MSDs, the APIs use this value as follows: 0—Browse APIs only query index tables. If index tables are not built, then the API returns an error. 1—Browse APIs only query the device, ignoring index tables. 2—Default. Browse APIs query the index tables if they are built. Otherwise, they query the device.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\MultiApp

Key Value	Type	Description
MaxServers	DWORD	The number of client applications that can be supported concurrently. The maximum is four. The default value is 2.
ServerThreadPriority	DWORD	Thread priority for client applications. The default value is 251.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\iPod

Key Value	Type	Description
PreScanDuration	DWORD	The length of time that the media player collects track information before it starts AutoPlay. The collected track information becomes the complete selection of files available for AutoPlay. Increasing this length of time also increases how long you must wait to hear media playback. The default value 0x0FA0 (4 seconds).
BrowseOption	DWORD	This value specifies how the browse APIs determine whether the device or the index tables should be used for queries. For iPod devices, the APIs use this value as follows: 0—Browse APIs do not work. Index tables are built when the device is attached. 1—Default. Browse APIs query the device directly. No index tables are built; the API works on top-level category lists.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\Fields\<field name>
 These keys enable new metadata fields to be indexed by Media Core. Each key contains the following values.

Key Value	Type	Description
CategoryId	DWORD	The unique identifier for this metadata field. The value must be greater than 0x100 and cannot exceed 0xFFFF.
MediaDataType	DWORD	The type of data that is expected for the metadata. The value must be of type MEDIA_DATA_TYPE. The default value is MediaTypeUnknown.
IndexField	DWORD	Optional. A Boolean flag that specifies if the metadata field is to be indexed as an extended metadata field. If the value is not specified, then the field is not indexed. The default value is 1 (TRUE).
TrackType	DWORD	A Boolean flag that specifies if the metadata field is to be a playable type. If the value is 0, it is not played by Media Core. The default value is 0.
FieldNameString	REG_SZ	Optional. The default string name for the field. If this value is specified, it is returned by the MediaGetSupportedFieldsInfo API. Otherwise, <field name> is returned.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\Fields\<field name>\<source plug-in>

These keys contain configuration values that are specific to source plug-ins. These values describe how metadata is retrieved from the source device.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\LocalSource

These keys contain configurations common to all local sources.

Key Value	Type	Description
CopyDRM	DWORD	If set to 1, all files with DRM are copied to local source. Otherwise, they are ignored. This field is optional. The default value is 0.
CopyDups	DWORD	If set to 1, then files that have the same name as an existing file (but contain different metadata) are copied to local source by appending a unique number to their name. This field is optional. The default value is 1.
CopyThreadPriority	DWORD	0 to 255. Thread priority for thread copying files to local source. This field is optional.
CurrentLocalSource	STRING	Name of the current active local source. This field is mandatory. If this is not set, then no local source is used.
TotalIndexSize	DWORD	Size of the index in bytes.

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\AutoMediaCore\Config\LocalSource

These keys contain configurations common to all local sources.

Key Value	Type	Description
BackupPersist	DWORD	If set to 1, then a backup of a persisted index is created before trying to create a new persisted index. If the current persisted index is corrupt when initializing the local source then the backup is used.
SourcePath	STRING	Fully qualified path of the local source. For example, \ATAPI Disk\User 1\Local Source. This field is mandatory, and this folder should exist already. Media Core does not create this folder structure.
CachePath	STRING	Fully qualified path of the folder to put persisted metadata information. For example, \\LocalSourceCache\User 1. This field is optional.

HKEY_LOCAL_MACHINE\Drivers\BuiltIn\IPDSvc

Key	Type	Value
StreamingThreadPriority	DWORD	Digital audio streaming thread priority. The default value is F9 (249).
MaxWavePoolSize	DWORD	The maximum number of wave headers. The default value is F.
WaveBufferFrameCount	DWORD	Number of wave frames sent at a time. The default value is 5.
WaveDataTimeout	DWORD	The timeout value. The default value is 1388.
InputSampleRate	DWORD	The sampling rate. The default value is AC44.
MinReadyHeaders	DWORD	The number of wave headers that must be received before playback will start. The default value is 5.
MaxnotReceiveTime	DWORD	The maximum time without receiving wave data before streaming will be stopped. The default value is 190.
PersistShuffleAndRepeat	DWORD	Sets the shuffle and repeat persistence when an iPod device is attached. A value of 1 means that the current shuffle and repeat settings of the attached device will be used for playback. The default value is 1.

HKEY_LOCAL_MACHINE\Services\IPDSvc

These keys contain general settings for the iPod service.

Key	Value
ComPort	Force service to search for iPod device on specified COM port. If this is not set, service will monitor for attachment of USB2SERIAL cable.
SamplingRates	List of audio sampling rates that hardware audio decoder supports. By default, it is 8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100, 48000. iPod specification requires support for at least: 32000, 44100 and 48000
BundleSeedIDPrefToken	A null-terminated UTF-8 string that identifies the vendor of the application. Case sensitive. This string is derived from the vendor's App ID which is assigned by Apple.
AudioDevice	Audio device ID that will be passed to waveOutOpen. If that is not specified, iPod driver will open WAVE_MAPPER device.
DisplayImage_Color/DisplayImage	File with color/black-and-white display image that will be sent to iPod.
HandlesComIndex	Service instance that handles 2-wire devices.
DisableDigitalAudio	Disables 1-wire digital audio streaming. A value of 0 implies not set. The default value is 0.

HKEY_LOCAL_MACHINE\Services\IPDSvc\AccessoryInfo

Accessories are required to send information to the connected iPod device that identifies them. Accessory information is required to get “Works for iPod/Made for iPhone” certification. The iPod service will load the following configurable registry keys and send the required information to the iPod when the iPod device is connected.

Key	Type	Value
FirmwareVersion	DWORD	Accessory firmware version. The format is 0x00[Major Byte][Minor Byte][Revision Byte]. For example, firmware version 1.2.0 is 0x00010200. The default value is 0x00000000.
HardwareVersion	DWORD	Accessory hardware version. The format is 0x00[Major Byte][Minor Byte][Revision Byte]. The default value is 0x00000000.
AccessoryName	String	The general name for this accessory. The value will be trimmed to a maximum of 64 characters (not including the null terminator). If not specified, an empty string is used.
Manufacturer	String	The name of the manufacturer of this accessory. The value will be trimmed to a maximum of 64 characters (not including the null terminator). If not specified, an empty string is used.
ModelNumber	String	The model number of this accessory. The value will be trimmed to a maximum of 64 characters (not including the null terminator). If not specified, an empty string is used.
MaxPayloadSize	DWORD	The maximum payload size of the accessory. The default value is 3fa, but can be changed by the OEM.

HKEY_LOCAL_MACHINE\Services\IPDSvc\AccessoryInfo

Accessories are required to send information to the connected iPod device that identifies them. Accessory information is required to get “Works for iPod/Made for iPhone” certification. The iPod service will load the following configurable registry keys and send the required information to the iPod when the iPod device is connected.

Key	Type	Value
BundleSeedIDPrefToken	String	Identifies the application that is set up for EAF with the automotive device.

HKEY_LOCAL_MACHINE\Services\IPDSvc\AccessoryInfo\Protocols

Key	Type	Value
ProtocolTokenCount	DWORD	Minimum value is 1 if a BundleSeedID is registered. The value must match the number of ProtocolToken values in the registry.
ProtocolToken<value>	String	The protocol ID string. <value> begins with 1 and proceeds incrementally.

HKEY_LOCAL_MACHINE\Services\IPDSvc\MetadataEx\<name>

Key	Type	Value
FieldId	DWORD	Any value 32 and above. Must be unique for all sub-keys.
Name	REG_SZ	Any value. Optional, empty string will be used if not specified.
CmdType	DWORD	0 - basic command, with 24 byte return value. 2 - buffer command, with variable sized buffer return value. Optional, default value is 0.

HKEY_LOCAL_MACHINE\Drivers\BuiltIn\IPDSvc

Key	Type	Definition
StreamingThreadPriority	DWORD	Digital audio streaming thread priority: 249.
MaxWavePoolSize	DWORD	Maximum number of wave headers.
WaveBufferFrameCount	DWORD	Number of wave frames sent at a time.
WaveDataTimeout	DWORD	Timeout number.
InputSampleRate	DWORD	Sampling rate.
MinReadyHeaders	DWORD	Number of wave headers that must be received before playback will start.
MaxNotReceiveTime	DWORD	Maximum time of no wave data received before streaming will be stopped.

HKEY_LOCAL_MACHINE\Drivers\BuiltIn\IPDSvc

Key	Type	Definition
PersistShuffleAndRepeat	DWORD	Sets the shuffle and repeat persistence when an iPod device is attached. A value of 1 means that the current shuffle and repeat settings of the attached device will be used for playback.

HKEY_LOCAL_MACHINE\Drivers\BuiltIn\IPDSvc1

Key	Type	Definition
DLL	REG_SZ	Name of DLL. The default value is IPDSvc.DLL.
Prefix	REG_SZ	Prefix for registered device name. Prefix and Index together are the device name. In this example, "IPD1:" is the registered device name. The default value is IPD.
Index	DWORD	Hex value. Unique index from 1 to 9, inclusive. The default value is 1.
Order	DWORD	Hex value. Boot order: 110. The default value is 6E.
Flags	DWORD	DEVFLAGS_LOADLIBRARY DEVFLAGS_LOAD_AS_USERPROC
UserProcGroup	DWORD	Hex value. Default group for udevice.exe. The default value is 3.

HKEY_LOCAL_MACHINE\Drivers\BuiltIn\IPDSvc2		
Key	Type	Definition
DLL	REG_SZ	Name of DLL. The default value is IPDSvc.DLL.
Prefix	REG_SZ	Prefix for registered device name. Prefix and Index together are the device name. In this example, "IPD1:" is the registered device name. The default value is IPD.
Index	DWORD	Hex value. Unique index from 1 to 9, inclusive. The default value is 1.
Order	DWORD	Hex value. Boot order: 110. The default value is 6E.
Flags	DWORD	DEVFLAGS_LOADLIBRARY DEVFLAGS_LOAD_AS_USERPROC

MTP-related Registry Keys

The following registry keys are used by the MTP service. The registry will not be explicitly flushed whenever one of these values changes, which means that a sudden power loss could result in a registry change not being saved.

Table 26: Picture transfer protocol (PTP) interface device registry key

HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients\Default\Default\6_1_1\MtpHostUsbCddClass

Table 27: Generic USB device registry key

HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients\Default\Default\Default\MtpHostUsbCddClass
This registry key causes the MTP class driver to be loaded for any device that is not recognized by another driver.

Table 28: MTP driver registry key

HKEY_LOCAL_MACHINE\Drivers\MtpHostUsbCdd		
Name	Type	Description
DLL	String	This key will cause MtpHostUsbCdd.DLL to be loaded as the driver for PTP devices, which includes MTP devices. The default value is MtpHostUsbCdd.DLL.
FriendlyName	String	Friendly name of the DLL. The default value is the MTP Host Device Driver.
Prefix	String	The default value is MHU.
Flags	DWORD	DEVFLAGS_LOADLIBRARY DEVFLAGS_LOAD_AS_USERPROC
Profile	String	The default value is MTPHOST.
IClass	String	The unique identifier of the interface exposed by this driver. The default value is {75DF55D2-E8EA-46e5-ABB5-5CB701205209}.

HKEY_LOCAL_MACHINE\Drivers\MtpHostUsbCdd		
Name	Type	Description
UnsupportedDeviceList	Binary	The list of devices with unknown interfaces that are not MTP. Will be updated through usage. USB_DEVICE_LIST format. The default value is 0.
FuzzFlags	DWORD	Used by the test MtpHostUsbCddFuzzer.DLL only. Bitfield of ORed values: <ul style="list-style-type: none"> 0x0001 Fuzz header length 0x0002 Fuzz header type 0x0004 Fuzz header code 0x0008 Fuzz header transaction id 0x0100 Fuzz data Note that fuzzing the header will often cause the device to appear unresponsive because the MTP service will be waiting for data that the device will never send. The device should be unplugged and plugged back in.
TenthsOfPercent	DWORD	What percentage of bits to fuzz. Only valid when FuzzFlags is used. The default value is 0.

Table 29: MTP service registry key

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\MTP		
Name	Type	Definition
BufferSize	DWORD	Buffer used for internal transactions. Needs to be large enough to store GetObjectPropList PUID query for 32,000 songs and 3,000 other objects. Each result is 29 bytes so 29 x 35,000 ~ 1 MB. The default value is 0x100000.
FindMemoryThreshold	DWORD	How much memory is allowed for the various types of MTP finds. If the current finds exceed this amount, no further finds are possible until those find handles are closed. The default value is 0x200000.
MaxOpenFinds	DWORD	How many find handles can be open at once. The default value is 256.
MaxOpenSessions	DWORD	How many MTP service sessions can be open at once. The default value is 16.
MaxOpenDevices	DWORD	How many MTP devices can be open at once. The default value is 16.
MaxRegisteredApps	DWORD	The number of message queues that can be registered for notifications. The default value is 16.
PnpThreadDelay	DWORD	How many milliseconds to delay the plug and play thread before communicating with MTP devices. The default value is 0x1388.
ConnectionThreadCount	DWORD	How many connection threads are used by the service to handle device connect/disconnects. The default value is 2.
EventFlags	DWORD	0 - Ignore MTP Events from devices. 1 - Forward MTP Events from devices to applications.

HKEY_LOCAL_MACHINE\BuiltIn\MTPSVC

Name	Type	Definition
DLL	String	DLL for MTP service. The default value is MtpSvc.DLL
Prefix	String	MTP Service name prefix. The default value is MTP.
Index	DWORD	MTP Service name index. The default value is 1.
Order	DWORD	Controls the order in which the MTP service is loaded compared to other services. The default value is 0x6E.
Flags	DWORD	DEVFLAGS_LOADLIBRARY DEVFLAGS_LOAD_AS_USERPROC

Table 30: MTP indexing feature registry key

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\MTP\Indexing		
Name	Type	Definition
PUIDHashTableSize	DWORD	Sets the size of the PUID hash table. Should be prime for best performance. Larger values result in better performance but consume more memory. The default value is 0.
MaxPUIDHashSize	DWORD	Maximum number of PUIDs to store in the hash table. PUIDs beyond this number are stored in a list. The default value is 0.
UseObjectPropertyCache	DWORD	0 – Query for object properties on demand. 1 – Cache all properties of an object using GetObjectPropList to improve property access performance. Caching is done when the PUID is requested. The default value is 1.

Table 31: MTP audio feature registry key

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\MTP\Audio		
Name	Type	Definition
UseGetObject	DWORD	Enables/disables the use of GetObject to stream data. If this is set to 0, devices that only support GetObject streaming will show up as unsupported devices. The default value is 1.
UseObjectPropertyCache	DWORD	This flag only applies if property groups are not supported or the song metadata is not already cached due to indexing. This can improve MtpSvcGetSongInfo performance. 0 – Query for song object properties on demand. 1 – Cache all properties of an object using GetObjectPropList to improve song property access performance. Caching is done when the MtpSvcGetSongInfo is called. The default value is 1.

Table 32: MTP song formats registry key

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\MTP\Audio\SongFormats
 This registry key is used to enumerate the object formats that will be considered songs by the MTP service. Supported song formats are listed under this key by creating a new key with the matching MTP format code in hex. Defaults to WAVE, MP3 and WMA (3008, 3009, B901).

Table 33: MTP USB transport registry key

HKEY_LOCAL_MACHINE\Drivers\MtpUsbTransport		
Name	Type	Definition
UsbMaxTransferSize	DWORD	Sets the largest recommended transfer size for transferring data. The default value is 0x10000.
PowerResetOnRecovery	DWORD	0 – Nothing 1 – Repeated MTP errors will cause the MTP device to be automatically disconnected and reconnected. The default value is 0.

Table 34: Specific MTP devices registry keys

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\MTP\Devices\<device name>
 This registry key holds device-specific registry settings.

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\MTP\Devices\<device name>\Indexing
 This registry key holds settings for the MTP indexing feature.

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\MTP\Devices\<device name>\Audio
 This registry key holds settings for the MTP audio feature.

Zune

The only specific device currently supported is the Zune. By default, there are no Zune registry settings because different default values are used and you don't need to override them. Any SongFormats or PlaylistFormats registry entries will replace the defaults. They are not additive. If you want to add one SongFormat, you must list all SongFormats in the registry.

Table 35: Zune registry key

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\MTP\Devices\Zune\Indexing		
Name	Type	Definition
Indexing\PUIDHashTableSize	DWORD	Set to 0 to turn off PUID caching. The default value is 1021.

HKEY_LOCAL_MACHINE\Software\Microsoft\Automotive\MTP\Devices\Zune\Indexing

Name	Type	Definition
Indexing\MaxPUIDHashSize	DWORD	Set to 0 to turn off PUID caching. The default value is 50000.
Indexing\UseObjectPropertyCache	DWORD	This is should be set to 0 because PUID caching overlaps with ObjectPropertyCaching. The default value is 0.
Audio\UseGetObject	DWORD	The default value is 1.
Audio\UseObjectPropertyCache	DWORD	Object property groups cannot be used with the Zune because it always reports songs as WAVE format, so the actual property group cannot be determined. The default value is 1.
Audio\SongFormats\XXXX	DWORD	3008—WAVE 3009—MP3 B901—WMA B215—M4A
Audio\PlaylistFormats\XXXX	DWORD	0xBA05—Abstract Audio Video Playlist 0xBA03—Abstract Audio Albums

Appendix 4: Compatible Devices

The following is a list of phones tested and determined to be compatible with Windows Embedded Automotive 7.

Table 36: Compatible phones

Phone Manufacturer	Model	Phone Manufacturer	Model
Acer	DX900	Acer	M900
	S200		F900
Alcatel	Playboy	Apple	iPad A1219 64GB WiFi
Apple	iPad Wi-Fi + 3G 64GB		iPhone
	iPhone (3G)		iPhone 3GS
	iPhone4	BenQ	T60
BenQ-SIEMENS	EL71		7105t
BlackBerry	7290	BlackBerry	8120
	8330		8350i
	8520		8703e
	8830 World Edition		8900
	9000		9000 Bold
	9100 3G		9700
	Curve 8310		Curve 8320
	Curve 8900		Pearl 8100
	Pearl 8110		Pearl 8130
BlackBerry	Pearl Flip 8220	BlackBerry	Pearl Flip 8230
	Tour		7130v
	Storm		Cingular
Cingular	Sync	Dopod	
Dopod	C720w	Garmin	G60
Google	Nexus One	HP	iPAQ 514
HP	iPAQ 910		iPAQ 210
HTC	Advantage X7500	HTC	Apache (Sprint)
	Apache (Verizon)		Dash3G
	Desire		Droid Eris
	EVO 4G		Excalibur
	FUZE		HD2
	Hero		Hero A6262
	Kit T-Mobile G1		Magic A6161
	Mogul		My Touch 3G
	Ozone		P3400
	P3470		P3600i
	Pure		S710
	S730		Shadow
	Snap		Tattoo A3288
	TiLT2		T-Mobile G1
	Tornado (2.0)		Touch (Alltel)
	Touch (Sprint)		Touch Cruise P3650
	Touch Diamond		Touch Diamond2
	Touch Pro		Touch Pro 2
	Touch_HD_T8282		Typhoon C500
	TyTN II		Wizard
Huawei	S660	Huawei	T2211
	T550		U1270
	U1300		U1310
	U3300		U5700

Phone Manufacturer	Model	Phone Manufacturer	Model		
Huawei	U7300	Huawei	U7310		
	U7510		U9100		
Kyocera	E2500	Kyocera	E3500		
	Strobe		600G (NET 10)		
LG	AX490	LG	AX565		
	AX585 Rhythm		BL40		
	CF360		Chocolate (MX8500)		
	Chocolate (VX8500)		Chocolate (VX8500R)		
	CT810 [Incite]		CU400		
	CU500		Dare [VX 9700]		
	enV Touch VX11000		enV2[VX9100]		
	expo GW820		Fusic		
	GC900		GD510		
	GD580		GD900		
	GD910		Glimmer (AX830)		
	GM200		GM730		
	GR500 Xenon		GT505		
	GW520		GW525		
	HB620T		Invision		
	KC910		KC550		
	KC780		KE260		
	LG		KE500	LG	KE600
			KE850		KE850 Prada
KE970		KF300			
KF350		KF510			
KF700		KF701			
KF750		KF900 (Prada II)			
KG195		KG320			
KG800 (Chocolate)		KM380			
KM501		KM900 Arena			
KP500 Cookie		KS20			
KS200		KS360			
KS500		KS660			
KU800		KU970			
KU990		KU990I			
LX160		LX290			
ME770		MG810c			
Muziq (LX570)		Neon			
Rumor		Rumor2			
Scoop (AX260)		Shine (CU720)			
Trax (CU575)		TU915			
Venus (VX8800)		Versa			
Voyager		VX 8560 [Chocolate]			
VX 9200 [LG enV3]		VX5500			
VX8300		VX8360			
VX8575 Chocolate Touch		VX8610 [Decoy]			
VX9900		KB620			
KF600		KM500D			
Lotus	W300				
MIO	A501	Motorola	E8		
	L72		MB200		
Motorola	RAZR V8	Motorola	VE66		
	W7		W755		
	Motoming A1600		Entice W766		

Phone Manufacturer	Model	Phone Manufacturer	Model
Motorola	A1200 Ming	Motorola	A810
	ACTV		Barrage
	Clutch i465		Droid
	E770		EM30
	EM330		Evoke QA4
	Hint QA30		i1
	i365		i576
	i580		i615
	i776		i870
	i880		Karma QA1
	Krave ZN4		KRZR K1m (Sprint)
	KRZR K1m (Verizon)		MILESTONE
	MPX220		Q
	Q9C		Q9h
	Q9M		Rapture VU30
	RAZR Maxx Ve		RAZR V3
	RAZR V3a		RAZR V3i
	RAZR V3i/V3r		RAZR V3m (Sprint)
	RAZR V3m (Verizon)		RAZR V3t
	RAZR V3xx		RAZR V6 Maxx
RAZR V8	RAZR VE20		
RAZR2 V8	RAZR2 V9		
RAZR2 V9M	Renegade V950		
Rival A455	RIZR Z3		
RIZR Z6TV	ROKR E6		
Sidekick Slide	SLVR L6		
SLVR L7	SLVR L7C		
U9	V325i		
V525	V600		
V635	V750		
VA76r Tundra	VE465		
VE538	W376g		
W377	W385		
W490	W510		
Z9	ZINE ZN5		
L9	Atom Life		
MWG	Atom V	MWG	Zinc 11
NEC	e132	Nokia	2323
Nokia	2660B		2680s-2
	2680s-2b		2700 Classic
	2760		3230
	3555		5000
	5230		5300 Xpress Music
	5310 Express		5310 XpressMusic
	5500		5530
	5610 XpressMusic		5630d XpressMusic
	5730		5800 Express Music
	6021		6085
	6103		6111
	6125		6126
	6131		6133
	6133 Black		6133b
	6165		6208c
	6210 Navigator		6230

Phone Manufacturer	Model	Phone Manufacturer	Model
Nokia	6230i	Nokia	6263
	6267		6270
	6280		6300
	6301		6303 Classic
	6315i		6500 Classic
	6500 Slide		6555
	6600		6600i
	6620		6650
	6682		6700 Slide
	6760		6820
	6822a		7205 Intrigue
	7210 Supernova		7310c
	7373		7500 Prism
	7510a-b		7610
	7705 Twist		7900 Prism
	809		8600 Luna
	8800		8801
	9500		9500 Communicator
	E51		E52
	E61i		E63
E65	E66		
E71	E72-1		
E73 Mode	E75		
E90	N72		
N73	N75		
N76	N78		
N79	N80		
N81	N82		
N85	N85-1		
N86	N900		
N91	N95		
N96	X6-00		
Palm	Centro (AT&T)	Palm	Centro (Sprint)
	Pixi		Pre
	Treo 680		Treo 700WX
	Treo 755p		Treo 800w
Pantech	Treo Pro	Pantech	Breeze C520
	C3B		C610
	C630		CDM8950
	Duo		Matrix [C740]
Pharos	Pro C820	Pharos	Slate C530
	Traveller 117		Traveller 137
Porsche	P'9521	Sagem	my511X
Samsung	A237	Samsung	A257
	A657		Ace
	Alias 2		B2100
	Behold-SGH-T919		Blackjack
	C3053		D600
	D807		D900
	Delve SCH-R800		E2100B
	E250		E370
	Eternity		Exclaim SPH-M550
	F200		Flight A797
	Giorgio Armani -SGH-P520		Gleam

Phone Manufacturer	Model	Phone Manufacturer	Model		
Samsung	GT-B3310	Samsung	GTC 3510		
	GT-I8000		GT-M2310		
	GTS 5230		GTS 3100		
	GT-S3653		GT-S5233S		
	GT-S5600		GT-S5600T		
	HIGHNOTE		Hue		
	i600		i627		
	i760		Innov 8		
	Instinct		Instinct s30		
	Intensity SCH-U450		L600		
	L770s		M320		
	M520		Mantra SPH-M340		
	Omnia		Player Pixon		
	Player5		Propel (SGH A767)		
	Rant		Reclaim		
	Renown		Rogue SCH-U960		
	S3600		S3600i		
	S8300		SCH-A950		
	SCH-A990		SCH-i770 Saga		
	SCH-R600 Hue II		SCH-U520		
SCH-U550	SCH-U650(Sway)				
Samsung	Seek	Samsung	SGH-A747 (SLM)		
	SGH-A777		SGH-A827		
	SGH-A837		SGH-A877		
	SGH-A887		SGH-A897		
	SGH-D880		SGH-E215L		
	SGH-E380		SGH-E830		
	SGH-F110		SGH-F250		
	SGH-F330		SGH-F400		
	SGH-F480		SGH-F490		
	SGH-G800		SGH-I450		
	SGH-I637		SGH-I780		
	SGH-I900 Omnia		SGH-J600		
	SGH-M150		SGH-M200		
	SGH-P180		SGH-T329		
	SGH-T429		SGH-T469		
	SGH-T539 (Beat)		SGH-T619		
	SGH-T639		SGH-T729 (Blast)		
	SGH-T819		SGH-U600		
	SGH-U700		SGH-U800		
	SGH-U900		SGH-V777		
	SGH-Z510		SGH-ZV50		
	SPH-A900		SPH-A900M		
	SPH-i350 Intrepid		T349		
	T509 Black		T559		
	T629		T739		
	T939		U410		
	U740		V709		
	Z230		Z400		
	Z510		C6625		
	GT-I7500		S8003		
	Sanyo		Katana	Sanyo	Katana (Bell Mobility)
			Pro 200		SCP2700
			SCP-3810	Sharp	Sidekick 2008

Phone Manufacturer	Model	Phone Manufacturer	Model
Sharp	Sidekick LX	Sharper Image	202 TSI
SoftBank	706SC	Sony Ericsson	C510
Sony Ericsson	C902		C903
	C905		C905A
	D750i		F100i
	F305		G502
	G705		G900
	J105i		K330
	K510i		K618i
	K630i		K660i
	K700i		K750i
	K770i		K800i
	K810i		K850i
	M600i		NWZ-A828
	P1i		P910a
	P990i		R300iTelfort
	R306		S302
	S312		S500i
	S710a		Saito
T280i	T303		
T610	T630		
T637	T650i		
T700	T707		
T715	TM506		
TM717	U100i		
U10i	V800		
W205	W302		
W350i	W380i		
W395	W508		
W518a	W595		
W660i	W700i		
W705	W760		
W760a	W810i		
W850i	W880i		
W890i	W910i		
W950i	W960i		
W980	W995		
X1	X2		
Z310a	Z555		
Z600	Z610i		
Z750	z770i		
T-Mobile	Dash (1.0)	T-Mobile	Dash (1.5)
	MDA		SDA
	Sidekick 3		Tap
	Wing	Toshiba	Portege G710
Toshiba	Portege G900		TG01
	UTStarcom	Blitz TXT8010VW	UTStarcom

The following is a list of media devices tested and determined to be compatible with Windows Embedded Automotive 7.

Table 37: Compatible media devices

Media Device Manufacturer	Model	Media Device Manufacturer	Model
Apple	iPad A1219 64GB WiFi	Apple	iPad Wi-Fi + 3G 64GB
	iPhone		iPhone (3G)
Apple	iPhone 3GS	Apple	iPhone4
	iPod Classic (6G)		iPod Touch
	iPod Nano (3G)		iPod Classic (4G)
	iPod Nano (2G)		iPod Classic (5G)
	iPod Mini (2G)		Archos 7
Archos	Archos 104	Archos	Archos 5
	Archos 704 Wifi		Archos AV500
	Archos Gmini 402		Archos 7 6700
	604 WiFi		705 WiFi
	Gmini XS 100		Audiovox LYRA SLIDER
Centon	MP3 player	Coby	MP836-4G
Coby	MP305-4G		MP705
	MP705-2G		MP-C7095
Cowon	J3	Cowon	iAudio9
	Q5W		S9
	iAudio U2		iAudio U5
	iAudio X5		D2
	iAudio 6		ZEN MX
Creative	Creative Zen	Creative	Creative Zen Micro Photo
	Creative Zen Sleek Photo		Creative Zen V
Creative	Creative Zen V Plus	Creative	Creative Zen Vision M
	Creative Zen Vision W		Muvo T200
	MuVo TX FM		Zen Nano
	Zen Stone Plus		E-Matic EM108VIDB
	E-Matic		EM108VIDB
Haier	Haier ibiza Rhapsody	Insignia	Insignia Kix
Insignia	Insignia Sport	iRiver	SPINN
iRiver	E150		E100
	Clix		H10
	LPlayer		T10
	T30		T60
iSonic	Snapbox X-2	LG	KS500
LG	KM501		GW520
	KM900 (Arena)		KP500 Cookie
	KS660		T505
Meizu	MP4		Microsoft
Microsoft	Zune (2G)	Zune HD	
MobiBLU	DAH-1500i	MobiBLU	DHH-200
Motorola	VE66	Motorola	A1600
	E8	Nexstar	MA715
Nokia	N76	Nokia	X600
	5730		5230
	6303 classic		6610i slide
	6760		7210 Supernova
	7310c		E52
	E72		N85-1

Media Device Manufacturer	Model	Media Device Manufacturer	Model
	N86		N97
	5610		7510
	6301		6650
Philips	GoGear ViBE SA1VBE08K/17	Philips	GoGear Ariaz
	GoGear-Aria		GoGear HDD1630
	GoGear SA6045		GoGear SA9200
	SA1ARA08K/17		SA52XXBT
RCA	Lyra X3030	RCA	M5002
RCA	Opal M4004		U5
	Samsung Player 5		A657
	SGH-A897		L770S
	F330		GT-M2310
	S3600i		L770s
	SGH-i450		GT S5600
	GT-S3100		GT-S5233S
	Intensity SCH-U450		S8003
Samsung	Samsung S5		Samsung YP-K3
	Samsung YP-K5		Samsung YP-P2
	Samsung YP-S5		Samsung YP-T10
	Samsung YP-T9		Samsung YP-U2
	SGH F-480		SGH-A837
	SGH-A877		YP-S3
	YP-T10JAG		YP-T8A
	YP-Z5AB		Sansa Clip+
SanDisk	Sansa e250	SanDisk	Sansa c250
	Sansa Connect		Sansa E270
SanDisk	Sansa Fuze	SanDisk	Sansa M250
	Sansa View		Sansa Clip
	Sansa E140		Sansa Express
	Walkman NWZ-X1061		Walkman NWZ-S545
	Walkman NWZ-E345		Walkman NWZ-E344
	NW-A1200		NW-E003
	NWZ-A829		NWZ-B105F
	NWZ-S718		PSP
	Walkman NW-S203F		T700
	T707		W518a
	U100i		G502
	T700		TM717
	C903		C905A
	F100i		J105i
	NWZ-A726		NWZ-B135F
	T715		U10i
	W508		Walkman NWZ-A726
	Walkman NWZ-B135F	Super Talent	MEGA Screen
Toshiba	Gigabeat	Toshiba	Toshiba Gigabeat S
	Gigabeat MEG-F40S	Transcend	T.Sonic 610
Transcend	MP860	WalleTex	Wallet Flash
Zvue	Zvue 250		

Glossary

A2DP—Advanced Audio Distribution Profile. A2DP defines how high-quality audio (stereo or mono) can be streamed from one device to another over a Bluetooth wireless technology connection.

AAC—Advanced Audio Coding. AAC is a standardized, lossy compression and encoding scheme for digital audio that is designed to be the successor of the MP3 format. AAC generally achieves better sound quality than MP3 at many bit rates.

AEC/NS—Acoustic echo cancellation/noise suppression. AEC/NS is the process of removing noise and echo from a voice communication to improve voice quality on a phone call.

API—Application programming interface. An API is a source code interface that an operating system or library provides to support requests for services to be made by computer programs.

ASX—Advanced Stream Redirector. One of the three Windows Media metafile formats (ASX, WAX, and WVX). The ASX file is a metafile (a file that contains data about another file).

ATCI—AT command interpreter. ATCI enables the Windows Embedded Automotive 7–based device to be used as a modem. ATCI receives AT commands through an input serial port and then parses and interprets them into TAPI, Ex TAPI, or RIL calls. The responses are then converted to AT response codes and returned through the output serial port handle. Typically, the input and output serial ports are the same port. ATCI is used with DUN over Bluetooth.

AT commands—The Hayes command set, also called the AT (for attention) command set, is used by dial-up modems. The

command set consists of a series of short strings that combine together to produce complete commands for operations such as dialing, hanging up, and changing the parameters of the connection.

AVRCP—Audio/Video Remote Control Profile. AVRCP is designed to provide a standard interface to control devices to let a single remote control be in control of all of the audio/visual equipment to which a user has access.

BB—Baseband. The baseband is the physical layer lying on top of the Bluetooth radio layer in the Bluetooth stack. It manages physical channels and links, apart from other services such as error correction, data whitening, hop selection, and Bluetooth security. The baseband protocol is implemented as a Link Controller, which works with the link manager for carrying out link-level routines such as link connection and power control. The baseband also manages asynchronous and synchronous links, handles packets, and does paging and inquiry to access and inquire Bluetooth devices in the area. The baseband transceiver applies a time-division duplex (TDD) scheme.

Bluetooth wireless technology—An industrial specification for wireless personal area networks. Bluetooth provides connection and information exchange between devices, such as mobile phones, laptops, personal computers, printers, digital cameras, and video game consoles over a secure, globally unlicensed, short-range radio frequency.

Bluetooth version 2.0 + EDR introduced an Enhanced Data Rate (EDR) of 3.0 megabit/sec (basic signaling rate; the practical data transfer rate is 2.1 megabit/sec).

Codec—A device or a program that is capable of encoding and decoding a digital data stream or signal. Windows Embedded Automotive 7 only provides production-licensed decoders for Windows Media Audio and a development license for MP3.

CSP—Connection Service Provider. A CSP provides connection information to the Connection Manager application, writes provisioning information that is received from the service providers to the registry, and binds connection requests to the NDISUIO (NDIS User-Mode I/O) Driver.

DirectShow—A multimedia framework/API produced by Microsoft. Software developers can use DirectShow to perform various operations with media files or streams; DirectShow is based on the Windows® Component Object Model (COM) framework and provides a common interface for media across many programming languages. It is an extensible, filter-based framework that can render or record media files on demand.

DLL—Dynamic-Link Library. DLLs are implementations of the shared library concept in the Windows and OS/2 operating systems, and they have the file extension DLL, OCX (for libraries containing ActiveX® controls), or DRV (for earlier system drivers). DLLs can contain code, data, and resources, in any combination.

DLNA—Digital Living Network Alliance. A standard for moving movies, photos, music, and other media from one device to another. DLNA servers can store media in one location and stream the media to DLNA-compliant players without any setup or configuration.

DMP—Digital Media Player. Any home theater system or game console that plays audio or video material and/or displays photos.

DMS—Digital Media Server. Software that makes computer files available on the network.

DRM—Digital Rights Management. DRM refers to the access control technologies that are used by publishers and copyright holders to limit usage of digital media or devices.

DUN—Dial-Up Networking profile. DUN provides a standard to access the Internet and other dial-up services over Bluetooth wireless technology. DUN can be used to access the Internet from a laptop by dialing up wirelessly on a mobile phone.

Executable—A file the contents of which are meant to be interpreted as a program by a computer.

FAT—File Allocation Table. FAT is the primary file system for various operating systems. A TFAT is a Transaction Safe FAT.

Flash memory—Non-volatile computer memory that can be electrically erased and reprogrammed.

GOEP—Generic Object Exchange Profile. The GOEP provides a basis for other data profiles and is based on OBEX.

GPIO—General Purpose Input/Output. GPIO devices provide a set of I/O ports that can be configured for either input or output.

GPS—Global Positioning System. GPS utilizes at least 24 satellites that transmit precise microwave signals, enabling a GPS receiver to determine its location, speed, direction, and time.

GSM—Global System for Mobile Communications. The most popular standard for mobile phones in the world.

HCI—Host Controller Interface. An HCI is a basic interface to Bluetooth hardware,

responsible for controller management, link establishment, and maintenance.

HFP—Hands-Free Profile. HFP is commonly used to allow automotive hands-free kits to communicate with mobile phones in the car.

HMI—Human-Machine Interface. The HMI is the means with which users can interact with the system, including input and output capabilities.

IMGFS—Image File System. IMGFS is the main Windows Embedded CE image with the TFAT partitions included.

IOCTL—Input/Output Control. A part of the user-to-kernel interface of a conventional operating system, IOCTLs are typically used to enable userspace code to communicate with hardware devices or kernel components.

IPC—Inter-Process Communication. IPC is a set of techniques (message passing, synchronization, shared memory, and remote procedure calls) for exchanging data among multiple threads in one or more processes that are running on one or more networked computers.

L2CAP—Logical Link Control and Adaptation Layer Protocol. The L2CAP is layered over the baseband protocol and resides in the data link layer. L2CAP provides connection-oriented and connectionless data services to upper layer protocols with protocol multiplexing capability, segmentation and reassembly operation, and group abstractions. L2CAP permits higher-level protocols and applications to transmit and receive L2CAP data packets up to 64 kilobytes in length.

M3U—Moving Picture Experts Group Audio Layer 3 Uniform Resource Locator (also MP3 URL). M3U is a computer file format that stores multimedia playlists.

MAP—Message Access Profile. MAP defines a set of features and procedures to exchange messages between devices.

M-DMP—Mobile Digital Media Player.

M-DMS— Mobile Digital Media Server.

Middleware—Computer software that connects software components or applications. Middleware consists of services that allow multiple processes that are running on one or more computers to interact across a network.

MOST—Media-Oriented System Transport. MOST is a serial communication system for transmitting audio, video, and control data through fiber-optic cables. This multifunctional, high-performance multimedia network technology is based on synchronous data communication and requires professional software tools and hardware interfaces.

MP3—MPEG-1 Audio Layer 3. MP3 is a digital audio encoding format that is used to create a file to store a single segment of audio so that it can be organized or easily transferred between computers and other devices.

MSD—Mass Storage Device. MSDs are used to store data. MSDs use a set of computing communications protocols defined by the USB Implementers Forum that run on the Universal Serial Bus.

MTP—Media Transfer Protocol. MTP is a set of custom extensions to the Picture Transfer Protocol (PTP) from Microsoft. MTP supports the transfer of music files on digital audio players and movie files on portable media players. MTP is closely related to Windows Media Player.

OBEX—Object Exchange. OBEX is a communications protocol that facilitates the exchange of binary objects between

devices. Many PDAs use OBEX to exchange business cards, data, and applications.

OPP—Object Push Profile. OPP defines the requirements for the protocols and the procedures to be used by the applications that are involved in the pushing and pulling of data objects between Bluetooth devices.

PBAP—Phone Book Access Profile. PBAP enables the exchange of Phone Book Objects between devices. It can be used between a car kit and a mobile phone to let the car kit display the name of the incoming caller.

PCE—Phonebook Client Equipment role. The PCE role is for the device that retrieves phonebook objects from the phonebook server.

PCM—A term for data that is encoded as Linear Pulse Code Modulation (LPCM). LPCM is a method of encoding audio information digitally.

PDA—Personal Digital Assistant. PDAs are handheld (or palmtop) computers. Newer PDAs also have color screens and audio capabilities, which allows them to be used as mobile phones (smartphones), Web browsers, or portable media players.

PMP—Portable multimedia player. PMPs are consumer electronics devices that are capable of storing and playing digital media. The data is typically stored on a hard drive, microdrive, or flash memory. Mobile phones are also sometimes referred to as PMPs because of their playback capabilities.

POOM— Pocket Outlook Object Model, a Microsoft Component Object Model (COM)-based library that provides programmatic access to Microsoft® Office Outlook® Mobile Personal Information Management (PIM) data items and container objects, for phonebook storage.

PTT—Push-To-Talk. PTT is the process that gives a user the ability to start a dialog with the system by pressing a button and verbally issuing a command. Any speech process that is executing is paused, and the system switches to listening mode. When the system finishes listening, any process that was paused resumes.

PWM module—Pulse Width Modulation module. The purpose of the PWM module is to enable time-critical waveform operations to be handled by the hardware instead of by software.

RDS—Radio Data System. A communications protocol standard for embedding small amounts of digital information in conventional FM radio broadcasts. The RDS system standardizes several types of information transmitted, including time, station identification, and program information.

Remote layer—The layer that enables the speech service to be invoked remotely on the Windows Embedded Automotive 7–based device.

RIL—Radio Interface Layer. RIL provides a uniform radio interface API that can interface with a diverse set of radio modules and standards in the wireless industry. The RIL makes port communication easier by providing a uniform API, because not all radio interfaces that use an AT interface use the same command set.

RPP—Recognition Pre-Process. RPP determines a speech-recognition confidence score based on user audio input. The confidence score enables the speech service to determine the best match between user audio input and the current grammar.

RTC—Real-Time Clock. A computer clock, usually in the form of an integrated circuit, that keeps track of the current time.

SAPI—Speech API. SAPI is an API that was developed by Microsoft for speech recognition (converts spoken words to machine-readable input) and text-to-speech (the artificial production of human speech) within Windows-based applications.

SAT—Satellite radio. A satellite radio or subscription radio (SR) is a digital radio signal that is broadcast by a communications satellite, which covers a much wider geographical range than terrestrial radio signals.

SBC—Sub-band Codec. SBC is an audio encoder and decoder used to connect to the Internet, in addition to Bluetooth high-quality audio devices, like headphones or loudspeakers.

SBP2—Serial Bus Protocol 2. SBP2 is a transport protocol that is defined within Serial Bus, IEEE Standard 1394-1995 (also known as FireWire or i.Link), developed by T10.

SCO—Synchronous Connection-Oriented protocol. SCO is a type of communications link that is used within the Bluetooth wireless communications standard for small electronic devices.

SD card—Secure Digital card. An SD card is a flash (nonvolatile) memory card format that was developed by Matsushita, SanDisk, and Toshiba. SD cards are used in portable devices, such as digital cameras, handheld computers, PDAs, mobile phones, and GPS units.

SDIO—Secure Digital Input/Output. Devices that support SDIO (typically PDAs, laptops, or mobile phones) can use small devices that are designed for the SD form factor, such as GPS receivers, Wi-Fi or Bluetooth adapters, modems, Ethernet adapters, or

other mass storage media, such as hard drives.

SDP—Service Discovery Profile. SDPs are network protocols that enable automatic detection of devices and services offered by the devices on a computer network. (For example, the Bluetooth SDP is a profile that is used to find out which Bluetooth services are offered by the remote device.)

SMS—Short Message Service. API and router support the SMS services that are available on mobile phones. SMS is a communications protocol that enables the interchange. The SMS API talks to the SMS router to implement most of the short text messages between mobile telephone devices. SMS technology has facilitated the development and growth of text messaging.

SPP—Serial Port Profile. The SPP emulates a serial cable to provide an easily implemented wireless replacement for existing RS-232-based serial communications applications, such as familiar control signals. It provides the basis for other profiles, such as DUN, Headset Profile (HSP), and AVRCP profiles.

SyncML—Synchronization Markup Language. SyncML is a platform-independent information synchronization standard.

System grammar—The speech grammar that is associated with the system. The system grammar is always active when the speech system is in listening mode, even if no application has the token.

TDI—Transport Driver Interface. An interface in the Windows Embedded CE operating system that serves as an adaptation layer to Winsock-based user APIs. It isolates the highly asynchronous callback-based architecture of the stack

that is presenting a Windows® Sockets Specification 1.1 interface.

TFAT—Transaction-safe FAT. A TFAT file system is a file system that is designed specifically to provide transaction safety for data that is stored on a disk. TFAT requires a hardware-specific driver that is designed for the type of media on which the TFAT volume resides.

TLB—Translation Lookaside Buffer. A TLB is a CPU cache that memory management hardware uses to improve virtual address translation speed.

TMC—Traffic Message Channel. TMC is digitally encoded traffic and travel information (typically encoded using FM-RDS).

UART—Universal Asynchronous Receiver/Transmitter. UART is computer hardware component (an individual or a part of an integrated circuit) that translates data between parallel and serial forms. UARTs are now commonly included in microcontrollers.

UPL—Update Loader.

USB—Universal Serial Bus. USB is a serial bus standard for interface devices that is designed to let peripherals be connected by using a single standardized interface socket, improving plug-and-play capabilities because devices can be connected and disconnected without restarting the computer (called hot swapping).

vCard—A file format standard for electronic business cards. vCards are frequently attached to email messages but they can also be exchanged on the World Wide Web. vCards can contain name and address information, telephone numbers, URLs, logos, photographs, and audio clips.

VPN—Virtual Private Network. A VPN is a computer network in which some of the links between nodes are carried by open connections or by virtual circuits in some larger networks (such as the Internet), as opposed to running across a single private network.

VSP—Virtual Serial Port. A VSP is a redirector without network software support that is usually used to create a pair of back-to-back virtual COM ports on the same computer.

WAV—Waveform Audio Format. WAV is a Microsoft and IBM audio file format standard for storing an audio bitstream on a computer. It is a variation of the RIFF (Resource Interchange File Format, a generic meta-format for storing data in tagged chunks) bitstream format method for storing data in “chunks” and it is the main format used on Windows for raw and typically uncompressed audio. The default bitstream encoding is the Microsoft Pulse Code Modulation (LPCM) format.

Wi-Fi—Wireless Fidelity. Wi-Fi is a wireless technology that promotes standards for the interoperability of wireless local area network products based on the Institute of Electrical and Electronics Engineers (IEEE) 802.11 standards. Common applications for Wi-Fi include Internet and voice over IP (VoIP) phone access, gaming, and network connectivity for consumer electronics.

WMA—Windows Media Audio. WMA is an audio data compression technology. WMA can refer to the audio file format or its audio codecs.

WPL—Windows Media Player Playlist. WPL is a computer file format that stores multimedia playlists.