Windows Mobility Center – Extensibility

Version 3.1

Published: July 2006

Mike Pautz (mikepau@microsoft.com)

Business Development Manager, Mobile PC Business Unit (primary contact)

Guy Barker

Software Design Engineer, Mobile PC Business Unit (author)

Alec Berntson

Program Manager, Mobile PC Business Unit (2nd edition author)

For the latest information, please see: http://msdn.microsoft.com/mobilepc

Contents

Windows Mobility Center

Operating system and computer settings are located in various Control Panel applications (CPLs) throughout the system in a manner that is optimized for desktop use.

Because most mobile PC users change environment and context, they may need to modify certain computer settings on an ongoing basis to adapt to changing conditions and locations. Navigating between multiple control panels regularly, however, proves tedious to many users, at best. Microsoft® Windows® Mobility Center provides a discoverable, consolidated user interface (UI) of frequently-used system settings. In addition, original equipment manufacturers (OEMs) can plug in hardware-specific settings for their computers to Windows Mobility Center, providing customers additional value as well as distinguishing their name through extensibility and branding.

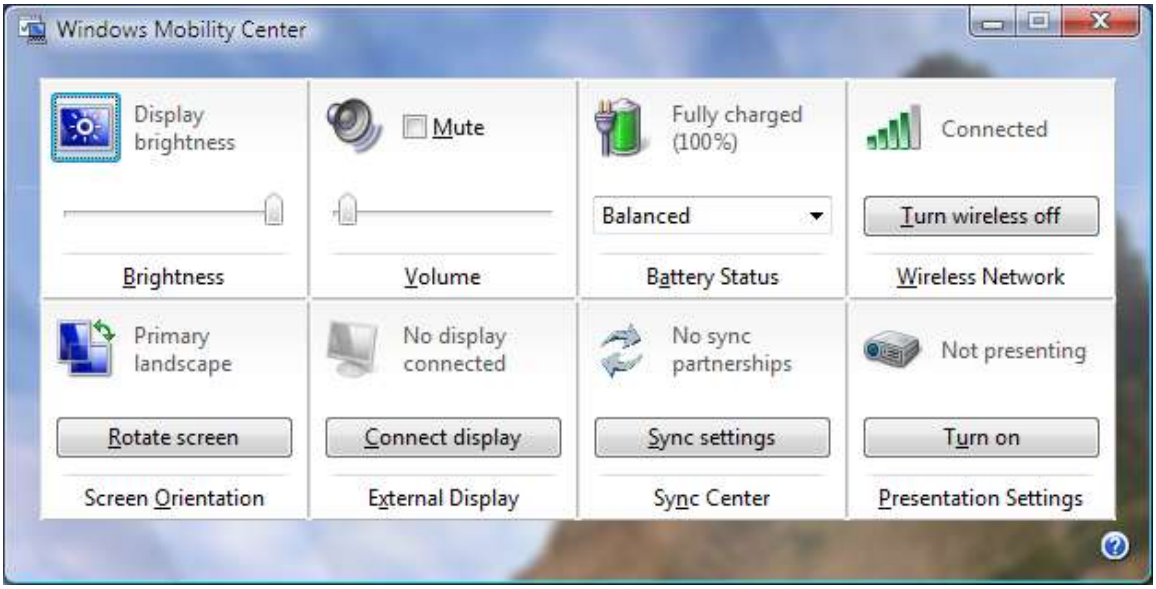


Figure 1: Windows Mobility Center

The fundamental unit in Windows Mobility Center is the *tile*. Each setting exposed in Windows Mobility Center has a dedicated tile that displays that setting's name and status. The tile also contains a link to a relevant Control Panel application and a simple, but useful control. Currently, Microsoft provides eight core tiles, which capture the most common settings present on most mobile PCs. These tiles provide mobile users with access to LCD Brightness, Volume, Power, Wireless, Display Orientation, External Display, Synchronization Center, and Presentation Settings controls.

This white paper includes a description of the functionality and extensibility inherent in Windows Mobility Center, as well as instructions for the tutorial sample code (available on the OPK CD or from OEMCOM) that can be used as a starting point to customize Windows Mobility Center.

The Eight Windows Tiles

The following table contains examples and descriptions for the eight core tiles of Windows Mobility Center.

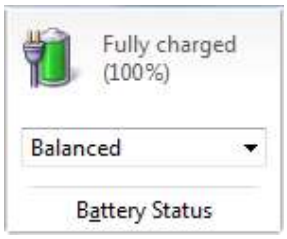
LCD Brightness
Shows the current brightness setting. The slider enables the user to modify the LCD screen brightness. The user clicks the icon to open the Personalization CPL.

> **Note:** In order for the brightness slider to work with your hardware, you must do one of the following:

1. Implement the ACPI brightness methods in the BIOS. Please see BIOS Communication for Display Drivers in Windows Vista (http://go.microsoft.com/fwlink/?LinkId=50987) for more information.

2. Work with your video independent hardware vendor (IHV) to provide brightness support in their video miniport driver.

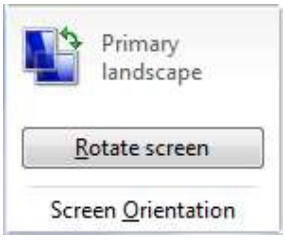
Volume
Shows the current Volume level. The slider enables the user to change the system volume. The user clicks the icon to open the Audio Mixing Console. The tile also contains a check box to control the mute setting of the system volume.



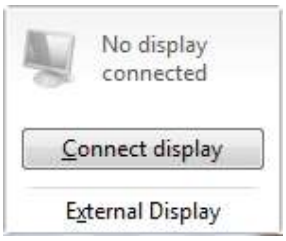
Battery Status
Shows the battery charge status (in percent) and the current power plan. The user selects which plan to use from the three available in the system power plan drop-down menu. The plans displayed in the drop-down are the same as in the system battery meter fly-out. The user clicks the icon to open the Power Options CPL.

Wireless Network
Shows the wireless connection status and signal strength. The user can toggle the wireless adaptor on and off by clicking the button. We highly recommend OEMs provide native Wi-Fi drivers to make this functionality possible. The user clicks the icon to go the Network Center CPL.

> **Note:** In order for the wireless on/off button to work with your hardware, you must provide a Native Wi-Fi (NWF) driver which implements radio on/off. Please work with your networking IHV to get a driver which supports this.



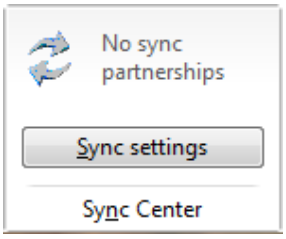
Screen Orientation (This tile only appears in Windows Mobility Center on Tablet PCs.)
Shows the current display orientation. When the user clicks the button, the system rotates the screen to the next orientation as defined in the current system settings. The user clicks the icon to open the Tablet PC CPL.



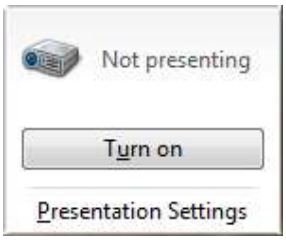
External Display
Shows the computer's current external display status. When the user clicks the button the system performs a destructive poll on the video hardware, searching for legacy displays (non-EDID). If the system finds a legacy display:

- The Transient Multi-Monitor wizard launches (a new feature in Windows Vista that simplifies connecting external monitors) if no displays were previously connected.
- The system opens the Display CPL with the Settings tab selected if 1 or more monitors were previously connected.

We highly recommend that OEMs provide a WDDM driver that supports destructive polling to ensure this feature functions properly. The user clicks the icon to open the personalization CPL.



Synchronization Center
Shows the status of the last synchronization (managed by the Synchronization Center). The user clicks the button to force a synchronization with all defined synchronization partnerships. The user clicks the icon to open the Windows Synchronization Center.



Presentation Settings
Presentation Settings is a new premium SKU feature in Windows Vista that enables the user to adapt the mobile PC for giving a presentation by:

- Disabling the screensaver, screen blanking, and standby/sleep timer.
- Silencing system alerts
- Changing the system volume (optional).
- Showing a custom desktop background (optional).

The tile shows whether Presentation Settings is applied or not. The user clicks the button to apply Presentation Settings or restore normal settings. The user clicks on the icon to open the Presentation Settings configuration dialog.

Entry Points

Windows Mobility Center has multiple entry points to increase discoverability, currently in the following locations:

- Start menu shortcut (Click **Start**, click **Accessories**, and then click **Window Mobility Center**.)
- Control Panel icon and tasks (Mobile PC category)
- Battery Meter menu and submenu
- Keyboard shortcut (WINDOWS LOGO KEY + X)
- Link in the Power control panel
- Link in the Personalization control panel
- Link in the Tablet PC control panel (Tablet PC only)
- Tablet button (Tablet PC only, optional)

Additionally, we **highly recommend** that OEMs provide a hardware button to launch Windows Mobility Center.

Extensibility

Windows Mobility Center gives OEMs the ability to add more hardware-specific settings for their mobile PCs and to provide an OEM-branded mobility experience. Tiles can be used as static links to feature specific control panel applications or as dynamic status displays by using COM objects. OEMs that create at least one new tile for Windows Mobility Center have the opportunity to display their name and logo along with the new tile(s).

Branding Windows Mobility Center with OEM images

Windows Mobility Center can display an image supplied by the OEM. This enables the OEM to create a product with a custom and OEM-specific feel while maintaining user interface consistency with the rest of Windows. Figure 2 shows the area occupied by the branding image in the Windows Mobility Center UI and the spaces reserved for OEM tile extensions, beneath the image area.

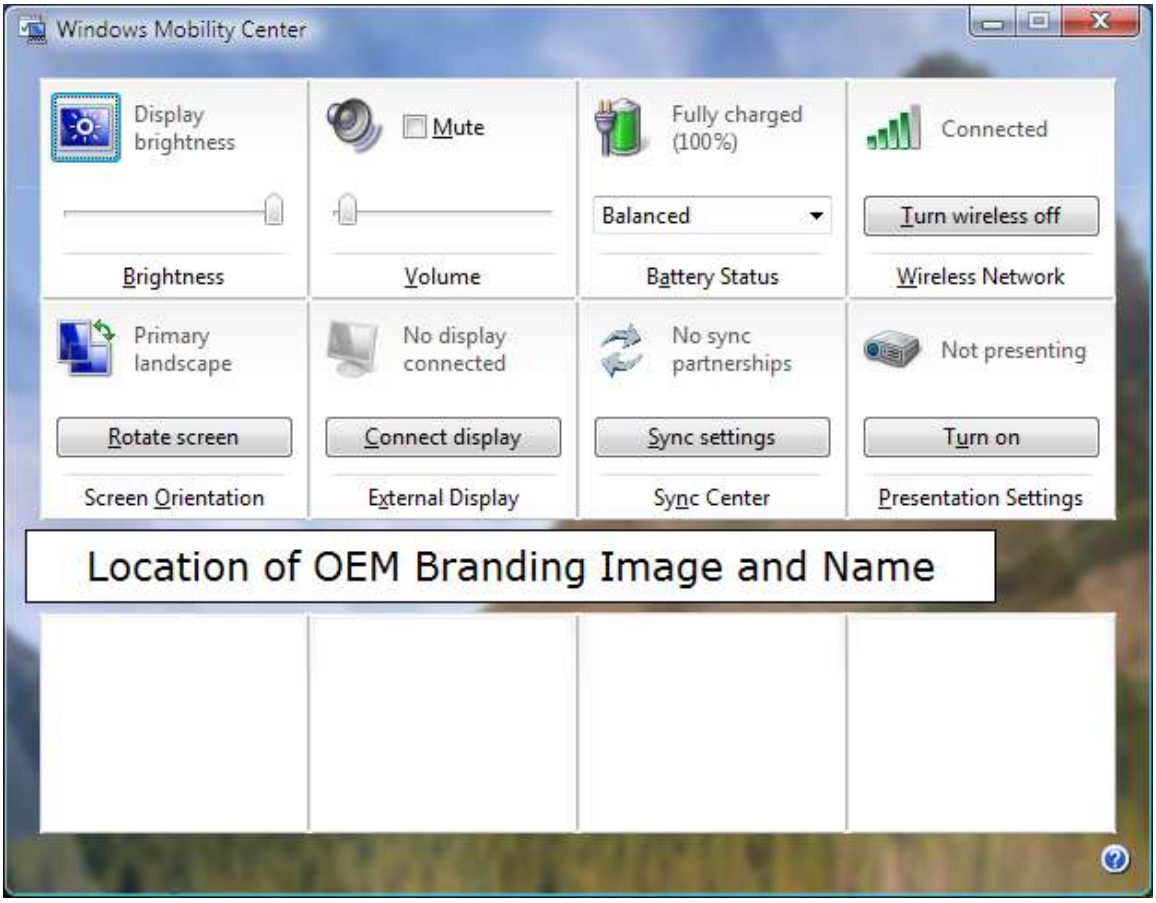


Figure 2: Branding image and name location

The branding image and name are specified by registry keys. When Windows Mobility Center starts, it first references the following key to retrieve the name of the OEM.

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MobilePC\MobilityCenter\OEMName
```

This value is mandatory; Windows Mobility Center does not display any extended tiles without it. The type of the value is `REG_EXPAND_SZ`. The value contains a path to a binary file that contains a string resource for the name, and the ID of the string within the binary file.

To provide an OEM logo, the following registry value is required:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MobilePC\MobilityCenter\OEMImage
```

The type of this value is `REG_EXPAND_SZ`. The value contains a path to a binary file that contains a PNG image resource for the image, and the ID of the image in the file, separated by a comma. For example, if the image is in a file named `OEMSettings.cpl` and has a resource ID of 101, the registry value may be:

```
%ProgramFiles%\OEMInstallPoint\OEMSettings.cpl,101
```

Any strings read by Windows Mobility Center from the registry must be no more than 255 characters long.

> **Note** Windows Mobility Center displays the OEM branding name and image only if at least one OEM-provided tile is successfully loaded.

Adding OEM tiles

Tiles are the primary area available for extensibility. Windows Mobility Center can show up to eight extensible tiles. Figure 2 showed the location of OEM tiles beneath the OEM branding image. Figure 3 shows an example tile. To make sure users receive a consistent experience and to reduce development costs for those who extend Windows Mobility Center, the tile area does not present custom UI.

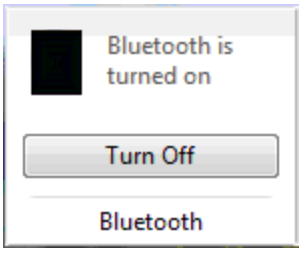


Figure 3: Example Windows Mobility Center tile

Each tile is composed of the same basic parts, illustrated in Figure 4.

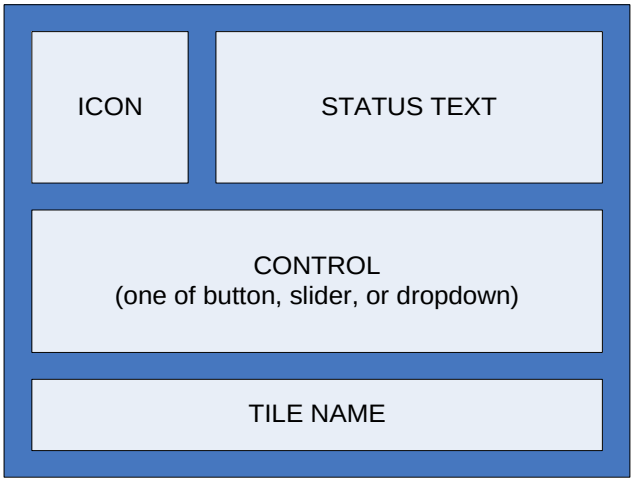


Figure 4: Layout of Windows Mobility Center tile content

The content of each tile can be specified by a developer as follows:

- **Icon** – An icon, which can optionally perform an action when clicked. The Click event should only be used to invoke a Control Panel or other application with extended settings.
- **Status text** – Text that shows status related to the purpose of the tile.
- **Control** – One button, slider, or dropdown, which performs an action when the user interacts with the control.
- **Tile name** – Text that identifies the tile.

The following table summarizes the OEM provided data for these tile parts:

Tile Component	Sub-Component	Mandatory/ Optional	Description
Icon	Icon	Mandatory	OEM provides the icon to be displayed on the tile. At least one **32x32** icon or **48x48** icon must be provided. Windows Mobility Center will choose which of these two icons to display based on the user's current UI settings. Preferably both icon sizes will be provided to present the user will cleaner graphics at different UI settings.
	Action	Optional	OEM can specify an action to be taken if the user clicks the icon. The recommended action is to launch a related Control Panel application.
	ToolTip	Mandatory if Icon Action specified	OEM can provide ToolTip text enabling screen readers to provide a useful description of an actionable icon.
Status text	Text	Mandatory	OEM provides the text that indicates some status associated with the tile.
Tile name	Text	Mandatory	OEM provides additional text displayed to the user.
Control	Text	Mandatory	OEM provides the text associated with the control, (either shown on the control, or accessible to a screen reader)
	Action	Mandatory	OEM specifies an action to be taken if the user interacts with the control.

Table 1: Tile data specified by OEM

Implementing OEM Tiles

Windows Mobility Center uses the following registry key to find the OEM provided tiles:

```
HKLM\Software\Microsoft\MobilePC\MobilityCenter\Tiles
```

Individual OEM tiles are represented by a subkey. The name of the subkey can be any string, but will generally be something related to the purpose of the related tile. For example:

```
HKLM\Software\Microsoft\MobilePC\MobilityCenter\Tiles\Bluetooth
```

Order of tiles

The order in which the eight Windows tiles appear in Windows Mobility Center cannot be changed. The order of the OEM tiles presented by Windows Mobility Center is specified by the following registry value:

```
HKLM\Software\Microsoft\MobilePC\MobilityCenter\TileOrder
```

The type of this value is `REG_MULTI_SZ`. The value contains one string per OEM tile. For example, the value could contain:

```
Bluetooth
SecondTile
ThirdTile
FourthTile
```

The strings must exactly match the strings used to represent each tile under the `HKLM\Software\Microsoft\MobilePC\MobilityCenter\Tiles` key.

> **Note** Windows Mobility Center loads only tiles specified in the `TileOrder` registry value. This value specifies which tiles are loaded and the order in which the tiles are displayed.

Types of tiles

OEM-provided tiles fall into two categories: Tiles with fixed UI, and tiles with UI that can periodically change, (for example to reflect a change in the status of some mobility-related setting.)

Static tiles

The UI for these tiles is loaded when Windows Mobility Center starts, but the UI remains fixed while Windows Mobility Center is running. Static tiles can be used as launch points for Control Panel applications and for other applications, but they do not provide current status for settings that can change while Windows Mobility Center is running. Static tiles are generally less useful than dynamic tiles, as they are stateless. This is explained further in the following section, *Dynamic Tiles*.

Static tiles only contain button controls, not slider or dropdown controls.

The data associated with a static tile is specified in the registry. As described earlier, the tile is represented by a key in the registry. For example:

```
HKLM\Software\Microsoft\MobilePC\MobilityCenter\Tiles\StaticTile
```

This key contains values for the seven tile sub-components listed in Table 1. All the values associated with a static tile are of type `REG_EXPAND_SZ`. The values are:

Tile Component	Value
`TileName`	Path to a file that contains a string resource and the ID of the resource.
`Icon`	Path to a file that contains an icon resource and the ID of the resource.
`IconAction`	Command line to be executed when the icon is clicked.
`IconToolTip`	Path to a file that contains a string resource and the ID of the resource.
`StatusText`	Path to a file that contains a string resource and the ID of the resource.
`ButtonAction`	Command line to be executed when the button is clicked.
`ButtonLabel`	Path to a file that contains a string resource and the ID of the resource.

The path and ID in these values are separated by a comma. (Again, all the values associated with a static tile are of type `REG_EXPAND_SZ`.) For example:

Tile Component	Value
`TileName`	%ProgramFiles%\OEMInstallPoint\OEMMobility.dll,201
`Icon`	%ProgramFiles%\OEMInstallPoint\OEMMobility.dll,301
`IconAction`	%ProgramFiles%\OEMInstallPoint\OEMSettings.cpl
`IconToolTip`	%ProgramFiles%\OEMInstallPoint\OEMMobility.dll,202
`StatusText`	%ProgramFiles%\OEMInstallPoint\OEMMobility.dll,203
`ButtonAction`	%ProgramFiles%\OEMInstallPoint\OEMConfigure.exe
`ButtonLabel`	%ProgramFiles%\OEMInstallPoint\OEMMobility.dll,204

Note If the path specified with the action for the icon or button contain spaces, then the path

must be contained within double quotes. For example, the action: **c:\my folder\prog1.exe** must be entered as: **“c:\my folder\prog1.exe”.** Paths specifying resource locations do not need double quotes unless the path itself contains a comma.

Dynamic tiles

Dynamic tiles respond to changes in some mobility-related settings while the tile is being shown in Windows Mobility Center. These tiles provide current status for the setting, and are therefore expected to update regularly after Windows Mobility Center starts.

Dynamic tiles are provided in the form of COM objects which are loaded when Windows Mobility Center starts. These COM objects can take whatever action they need to in order to query and track related mobility settings, and can request that Windows Mobility Center update the tile UI as necessary.

COM tile server runs out-of-process

Windows Mobility Center loads the extensible tiles as local out-of-process servers, rather than in-process servers. This means that instead of the tile server being a DLL that is loaded in the same process space as Windows Mobility Center, the tile server runs in a separate process. This prevents individual tile malfunctions from disrupting Windows Mobility Center.

The sample solution builds a tile server called `MobilityCenterSample.exe`. You register this tile server as a COM server by running the following command:

```
MobilityCenterSample.exe /regserver
```

To undo the registration of the COM server, run the following command:

```
MobilityCenterSample.exe /unregserver
```

The full lists of steps required for registering the sample tiles can be found in *Building the Sample,* later in this document.

Implementing dynamic tiles

The data associated with a dynamic tile is specified in the registry. For example:

```
HKLM\Software\Microsoft\MobilePC\MobilityCenter\Tiles\DynamicTile
```

This key typically contains a single value that represents the class ID for the COM object that is providing the details for the dynamic tile. The value associated with a dynamic tile is of type `REG_SZ`. The value is:

```
GUID        Contains the class ID for the COM object
```

An example of this value might be:

```
GUID        REG_SZ     {B158C8A3-5A35-45A5-9FF5-3844241E06C4}
```

The COM object implements an interface that enables Windows Mobility Center to retrieve the data to be displayed in the Windows Mobility Center UI. The specific interface implemented by the COM object depends upon the controls which are to be shown on the tile. A dynamic tile can be one of a button tile, a slider tile or a dropdown tile. Each of these tile types implements one of the **IMobilityCenterButtonTile**, **IMobilityCenterSliderTile**, or **IMobilityCenterDropDownTile** interfaces respectively. All of these interfaces derive from the **IMobilityCenterTile** interface, which has methods relating to the UI that exists on all tiles, (for example, the tile name and icon.)

The dynamic tile interfaces contain the following methods:

```
interface IMobilityCenterTile
    HRESULT  GetTileName(BSTR *pbstrTileName );
    HRESULT  GetControlInfo(BSTR *pbstrControlText, BOOL *pfEnabled);
    HRESULT  GetStatusText(BSTR *pbstrStatusText );
    HRESULT  GetIcon(int nWidth, int nHeight, HICON *phIcon, BOOL
*pfActionable, BSTR *pbstrIconToolTip);
    HRESULT  PerformIconAction();

interface IMobilityCenterButtonTile
    HRESULT  PerformButtonAction();

interface IMobilityCenterSliderTile
    HRESULT  GetInitialRange(int *pnMin, int *pnMax);
    HRESULT  GetCurrentPosition(int *pnPosition);
    HRESULT  GetSliderToolTip(int nPosition, BSTR *pbstrToolTip);
    HRESULT  PerformSliderAction(int nPosition, BOOL fSlideComplete);

interface IMobilityCenterDropDownTile
    HRESULT  GetItems(int cItemsMax, BSTR *pbstrItems, int
*pcItemsFetched);
    HRESULT  GetCurrentSelection(int *pnCurrentSelection);
    HRESULT  PerformDropDownAction(int nCurrentSelection);
```

The interfaces are defined in the file `MobilityCenterExtensibility.idl`, which is included in the Windows Mobility Center sample.

The methods implemented by the **IMobilityCenterButtonTile** interface perform the same functions as the related registry keys, described in the *Static Tile* section of this document, except that the **GetControlInfo()** method can specify whether the tile button should be enabled or disabled. For example, the button might be disabled in response to a change in some system setting that makes the button action inappropriate.

If the dynamic tile requires that the tile UI presented to the user be updated to reflect some change in a mobility-related setting, it calls the **MobilityCenterRefresh** API. This API is declared in the file MobilityCenterRefresh.h, included in the Windows Mobility Center sample. To build the OEM tile after a call to the API has been included, the tile links with the file MobilityCenterRefresh.lib, which is also included in the Windows Mobility Center sample.

The declaration of the **MobilityCenterRefresh** API is as follows:

```
HRESULT MobilityCenterRefresh(LPCWSTR pszTile);
```

The tile passes in the string identifier of the tile as specified for the subkey name under the `HKLM\Software\Microsoft\MobilePC\MobilityCenter\Tiles` key. (For example "`DynamicTile`".) After this API has been called, Windows Mobility Center calls the interface methods implemented by the COM object that provide the UI resources for the tile. This enables the tile to supply the strings and other resources that are appropriate to the current mobility-related settings relevant to the tile. Note that the case of the tile name supplied to the **MobilityCenterRefresh** API has to match the case of the string in the registry.

If Windows Mobility Center does not load the COM object, or if any part of the required data cannot be provided by the COM object, Windows Mobility Center attempts to load the tile as if it were a static tile. For this to be successful, the seven registry values associated with static tiles must exist in the registry key that contains the class ID of the dynamic tile. We highly recommend that OEMs provide these resources for every tile to provide a consistent user experience in the event of a failure. Note that this fallback approach of trying to load the tile as a static tile will be taken regardless of whether the dynamic tile was a button tile, a slider tile, or a dropdown tile.

Behavior of different control types on dynamic tiles

Different controls have subtle behavior differences on dynamic tiles.

Button tile

There is only one method specific to the **IMobilityCenterButtonTile** interface, **PerformButtonAction()**. All the other methods on **IMobilityCenterButtonTile** interface are those defined in the base **IMobilityCenterTile** interface.

PerformButtonAction() is called whenever the user clicks on the tile button.

Slider tile

When Windows Mobility Center calls the tile's **GetInitialRange()** method during initialization, the slider tile returns the minimum and maximum positions on the slider. The same instance of Windows Mobility Center will not call **GetInitialRange()** again, and so it is not possible for the slider range to change while a specific instance of the tile exists. The values returned by **GetInitialRange()** are only considered valid if the minimum value is less than the maximum value, and both values lie on or between -32,768 and 32,767. It is expected that in practice the range will not normally exceed the number of moveable positions that the slider has in the UI.

Windows Mobility Center calls **GetCurrentPosition()** to set the slider position in the UI. The position returned by the tile must lie on or between the values returned by the earlier call to **GetInitialRange()**. Note that Windows Mobility Center will not call **GetCurrentPosition()** if the

user is currently moving the slider in the UI. For example, say the slider tile gets a notification of a change in some system settings relating to the slider. The tile may determine the new slider position based on the system settings change, and call **MobiltyCenterRefresh()** to request that the tile UI is updated. Normally, Windows Mobility Center will respond to the request by calling **GetCurrentPosition()** and setting the new slider position. However, if the user is currently moving the slider, then Windows Mobility Center will not call **GetSliderPosition()**. This avoids a potential conflict between the position considered current by the slider tile, and the position to which the user is moving the slider.

While the user is moving the slider, Windows Mobility Center will call **GetSliderToolTip()** regardless of any requests the tile may have made to refresh the tile UI. By doing this, the user gets constant feedback about the position of the slider in the UI. The tooltip supplied by the tile can be any text string of 255 characters or less. The Microsoft Visual Studio® sample contains two sample slider tiles. The first slider tile returns a string representing the slider value passed in to **GetSliderToolTip()**, (for example “20”). The second slider tile maps the supplied slider position to a array of color strings, and returns that string as the tooltip, (for example “Red”).

Windows Mobility Center calls **PerformSliderAction()** when the slider is moved in the UI. This could be due to the user moving the slider with keyboard input, (for example by using the left or right arrow keys, or page up or down keys,) or by using the mouse or pen, (for example while dragging the slider, or clicking to the right or left of the slider.) When **PerformSliderAction()** is called, Windows Mobility Center supplies a **fSlideComplete** value to indicate if the slider movement is still in progress. The **fSlideComplete** value is **FALSE** when the user is dragging the slider but has not yet released it, or when the user is moving the slider via the keyboard and the key has not yet been released.

It is strongly recommended that the slider tile does not perform any actions that take a non-trivial time to complete while slider movement is still is progress. Any time spent beneath **PerformSliderAction()** will block an update of the slider tooltip and so will impact the user experience during the slider movement. The first slider tile in the Visual Studio sample simulates a slider tile that only takes its non-trivial action when the slider movement is complete. So during the slider movement, only the tooltip is updated. When the slider movement is complete, the entire tile UI is refreshed, and the status text is updated to include the final slider position. The second slider tile in the Visual Studio sample simulates a tile that needs the tile UI to be updated while the slider movement is in progress, so the tile calls **MobilityCenterRefresh()** during the slider movement. In response to this, Windows Mobility Center will get the current UI for the tile during the slider movement, and can update the icon and status text accordingly.

In order to maintain high performance feedback to the user while the slider is being moved, it is important to spend as little time as possible beneath the **IMobilityCenterSliderTile** methods until the slide is complete.

DropDown tile

When Windows Mobility Center calls the tile’s **GetItems()** method, the **DropDown** tile returns the number of items in the dropdown list, and the item strings themselves. Windows Mobility Center passes in the maximum allowed count of items, in the **cItemsMax** value. The current version of Windows Mobility Center passes in a **cItemsMax** value of **500**. Windows Mobility Center passes in an empty BSTR array, into which the tile adds the BSTR values for the item strings.

If **GetItems()** is called later in response to the tile requesting a UI update, the tile can return a different set of strings from those returned in an earlier call. However, given that the set of strings will often be constant, the tile can return S_FALSE from **GetItems()**. This signifies that the most recent set of items returned is still valid, and should continue to be shown in the tile UI.

It is recommended that **GetItems()** returns S_FALSE when the list of items has not changed

since the previous call to **GetItems()**. If **GetItems()** returns S_OK, then the re-population of the list of items in the UI may be noticeable if the number of items in the list is large.

The tile's **GetCurrentSelection()** returns the zero-based index of the item to be selected in the list. If a dropdown list contains no items, the **GetCurrentSelection()** method will still be called. In that case, the selected item index returned by the method will be ignored. (The return value from **GetCurrentSelection()** should still be success if the tile is to remain enabled.)

When the user selects an item in the dropdown UI, Windows Mobility Center calls the tile's **PerformDropDownAction()** method, passing the zero-based index of the newly selected item.

> **Note** Even if the button, slider, or dropdown control is disabled, the tile's **GetControlInfo()** method should still return a string representing the control. This text will be accessible to a screen reader.

Order of interface calls

Windows Mobility Center calls the dynamic tile interface methods in a specific order. Any slider or dropdown specific methods are called first, followed by the base interface methods in the following order.

```
interface IMobilityCenterSliderTile
    GetInitialRange()
    GetCurrentPosition()
    GetSliderToolTip()

interface IMobilityCenterDropDownTile
    GetItems()
    GetCurrentSelection()

interface IMobilityCenterTile
    GetTileName()
    GetStatusText()
    GetControlInfo()
    GetIcon()
```

Extensibility Sample

Note The sample assumes the reader has some familiarity with Microsoft Visual Studio and COM.

General Notes

This section contains general programming notes as they pertain to the Windows Mobility Center sample.

Security

If an OEM installs extensible tiles, the files on disk that relate to the tiles should be installed in a folder which requires high privileges to change. This helps prevents the files being overwritten by a 3rd party. If the files are overwritten, the user experience for the extensible area of Windows Mobility Center changes.

Visual Studio versions

The Visual Studio sample has been generated by using Visual Studio® .NET 2003. The sample can be opened in Visual Studio 2003 and Visual Studio® 2005. Earlier versions of Visual Studio cannot open the sample solution. If the sample is opened in Visual Studio 2005, it will be converted from Visual Studio 2003 format. (Any warnings generated during the conversion can be ignored.)

This section describes a sample that provides seven dynamic tiles and one static tile. The sample was generated by using Visual Studio's ATL Server Project Wizard. You need not use this approach for building a COM server, but the sample is available as a foundation for OEMs who want to use it to build their tiles. Use of the sample requires some familiarity with Visual Studio and COM.

After installing the OPK the sample is included in the **"%ProgramFiles%\Windows OPK\SDKs\Mobility Center\Sample"** folder. After installing the sample, it is recommended that the sample is copied from the installation folder to a user-related folder. (Some users may not have privileges to save modified sample content beneath %ProgramFiles%.)

The sample includes the following subfolders:

Subfolder	Description
MobilityCenterSample	Contains the Visual Studio sample for five sample tiles.
include	Contains the files defining the extensibility interfaces and the `MobilityCenterRefresh` API.
lib	Contains the x86 version of the library file implementing the `MobilityCenterRefresh` API. (If you need a 64-bit version of the library file, copy and paste it to this folder from the relevant Include folder)
output (not included)	Target folder that is created by Visual Studio for binary files generated when the sample is built.

Sample Contents

The sample contains the files in the following lists. Some project-related files are largely unchanged from the files generated by the Visual Studio wizard. The source files that implement the tile functionality contain comments that describe the file contents.

Project configuration and registration files

- MobilityCenterSample.reg
- MobilityCenterSample.sln
- MobilityCenterSample.rgs
- MobilityCenterSample.vcproj

Sample resource files

- MobilityCenterSample.rc
- Icon1.ico
- Icon2.ico
- Icon3.ico
- Icon4.ico
- Icon5.ico
- Icon6.ico
- Icon7.ico
- Icon8.ico
- OEMLogo.png

Sample source code files

- MobilityCenterSample.cpp
- MobilityCenterSample.h
- StdAfx.cpp
- StdAfx.h
- TileButton1.cpp
- TileButton1.h
- TileButton2.cpp
- TileButton2.h
- TileButton3.cpp
- TileButton3.h
- TileButton4.cpp
- TileButton4.h
- TileDropDown.cpp
- TileDropDown.h
- TileSlider.cpp
- TileSlider.h
- TileSliderLive.cpp
- TileSliderLive.h
- Resource.h

> **Note** The file containing the definition of the interfaces to be implemented by extensible tiles is contained in MobilityCenterExtensibility.idl. The related files generated by the MIDL compiler are not included with the sample, but are instead built when the sample solution is built.

Building the Sample

To build the Visual Studio solution that contains the sample Windows Mobility Center tiles:

1. Start Visual Studio.
2. Click the **File** menu, and then click **Open Solution**.
3. Navigate to and double-click the file

   "**Mobility Center\Sample\ MobilityCenterSample\MobilityCenterSample.sln**.
4. Click the **Build** menu, and then click **Build Solution**.
5. Verify that the Visual Studio output window displays **Build: 1 succeeded, 0 failed, 0 skipped.**

Note: Extensible tiles must be specifically built as either 32-bit or 64-bit binary files, depending on the target system on which they will be run. The previous steps build the sample for running on an x86 system. To build the sample for a 64-bit system, first copy the MobilityCenterRefresh.lib file from the relevant folder beneath the Mobility Center\Include Files folder on the OPK CD into the Mobility Center\Sample\Lib folder used when building the sample in Visual Studio.

When you build the sample solution, Visual Studio builds the MobiltyCenterSample project. This builds the COM server, which contains the seven sample dynamic tiles. The server is called MobilityCenterSample.exe, and is placed in the **Mobility Center\Sample\output** folder. (This folder does not exist on the OPK CD, but is created by Visual Studio when the sample is built.) If your build computer is the same Windows Vista computer on which you run Windows Mobility Center, you can leave the MobilityCenterSample.exe file where it was generated during the build. Otherwise, manually copy the file onto the target computer.

Once the MobilityCenterSample.exe file is in the folder from which it will be run, you must perform three registration related steps.

> **Note** You must be running with elevated privileges during these steps for the registration to succeed and to create and edit the required registry keys. If you are not running as administrator, the COM server registration silently fails and the registry keys are set for the current user only.

1. COM registration.

The first registration step is the COM registration required for any local server. From an elevated command prompt, run the following commands:

- cd <target folder>
- MobilityCenterSample.exe /regserver

2. Inform Windows Mobility Center

The second registration step makes Windows Mobility Center aware of the extensible tiles to be loaded. From an elevated command prompt, run the following commands:

- cd *<folder containing the Windows Mobility Center sample project>*
- MobilityCenterSample.reg

Note that the MobilityCenterSample.reg registration file puts only placeholder paths in the registry.

3. **Overwrite placeholder paths**

   The final registration step is to overwrite the placeholder paths in the registry with the actual path containing the MobilityCenterSample.exe. This path may be the **Mobility Center\Sample\output** folder in which the MobilityCenterSample.exe file was placed during the build. Perform the following actions in order to complete the registration steps:

   - Run regedit.exe with elevated privileges.
   - In the HKLM\Software\Microsoft\MobilePC\MobilityCenter key, modify the paths of the OEMName and OEMLogo values to contain the actual path of the MobilityCenterSample.exe file.
   - In the HKLM\Software\Microsoft\MobilePC\MobilityCenter\Tiles\StaticButton key, modify the five path values to contain the actual path of the MobilityCenterSample.exe file.

You do not need to repeat these registration steps unless you later run MobilityCenterSample.exe from a different folder.

Verifying the Sample Solution

The sample can be verified directly in Windows Vista by running Windows Mobility Center. If the extensibility interfaces have been implemented as expected and the server has been registered as expected, the tiles appear beneath the OEM logo in the Windows Mobility Center UI.

When a build of the sample solution is complete, the build report shows **Build: 1 succeeded, 0 failed, 0 skipped**. To verify everything works, invoke Windows Mobility Center. The UI shown in Figure 5 appears.

Figure 5: UI presented by Windows Mobility Center for default tiles provided in sample solution

Note If any of the strings for the tile status, button text, or tile name are too long to fit in the UI, then the string will be truncated and shown with a trailing ellipsis. (For example, see the status text on the fourth tile in Figure 5.) If the user places the cursor over a string that has been truncated, then a tooltip will appear containing the full string. If the full string fits in the UI, then no tooltip will appear for these UI elements.

Important: Please consider the length of localized strings when choosing the text that will appear on the tile. It is quite possible that a particular English string will fit in an area on tile, (for example in the Tile Name area), but the equivalent non-English string will appear truncated.

When you click any icon on a sample tile Windows Mobility Center launches Display Settings in Control Panel. When you click any button, Notepad launches, and the status text for the associated dynamic tile changes to **Tile button has been pressed**.

The registration steps described earlier result in seven dynamic tiles and one static tile being shown in Windows Mobility Center by default. You can modify the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MobilePC\MobilityCenter\TileOrder value to control the order in which the tiles appear. Notice that modifying the order of the entries in the TileOrder key is reflected the next time Windows Mobility Center runs.

Modifying the Sample Solution

You can modify the sample to present different UI and behavior. Previous sections describe how to do this by modifying resources contained in the sample. The last section, *Adding Specific OEM Functionality to a Tile*, contains more detail about how to change specific C++ files to change the behavior of the tiles in the sample.

Changing the OEM branding image

All the strings, icons, and images included in the sample are held as resources in the MobilityCenterSample project. To change the image displayed by the sample, change the image in the OEMLogo.png file and rebuild the solution. The next time the Windows Mobility Center runs, it displays the new OEM image.

The sample solution contains the following line in the .rc resource file:

```
IDR_IMAGE_OEMLOGO PNG   "OEMLogo.png"
```

The resource must be of type PNG. The maximum height of the image is 36 pixels, and the maximum width is 120 pixels. If the image is larger than this, Windows Mobility Center does not load any OEM data.

Windows Mobility Center also reads the OEM company name from the registry, in the following location:

```
HKLM\SOFTWARE\Microsoft\MobilePC\MobilityCenter\OEMName
```

This name is loaded as a string resource from an installed component at run-time in a similar way to the loading of the OEM image.

The OEM name is mandatory, but you need not supply the OEM image. However, if the **OEMImage** entry exists in the registry, it must reference an image resource that can be loaded and displayed in Windows Mobility Center.

Changing the Tile Order

Once the sample tiles have been registered by running the MobilityCenterSample.reg registry file, the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MobilePC\MobilityCenter\TileOrder value contains the following strings:

- Button1
- Button2
- Button3
- Button4
- StaticButton
- DropDown
- Slider
- SliderLive

To specify that the static button tile should be the first tile presented, move that tile name string to be the first tile specified. The **TileOrder** key would then contain the following values:

- StaticButton
- Button1
- Button2
- Button3
- Button4
- DropDown
- Slider
- SliderLive

After making this change, Windows Mobility Center presents the UI shown in Figure 6.

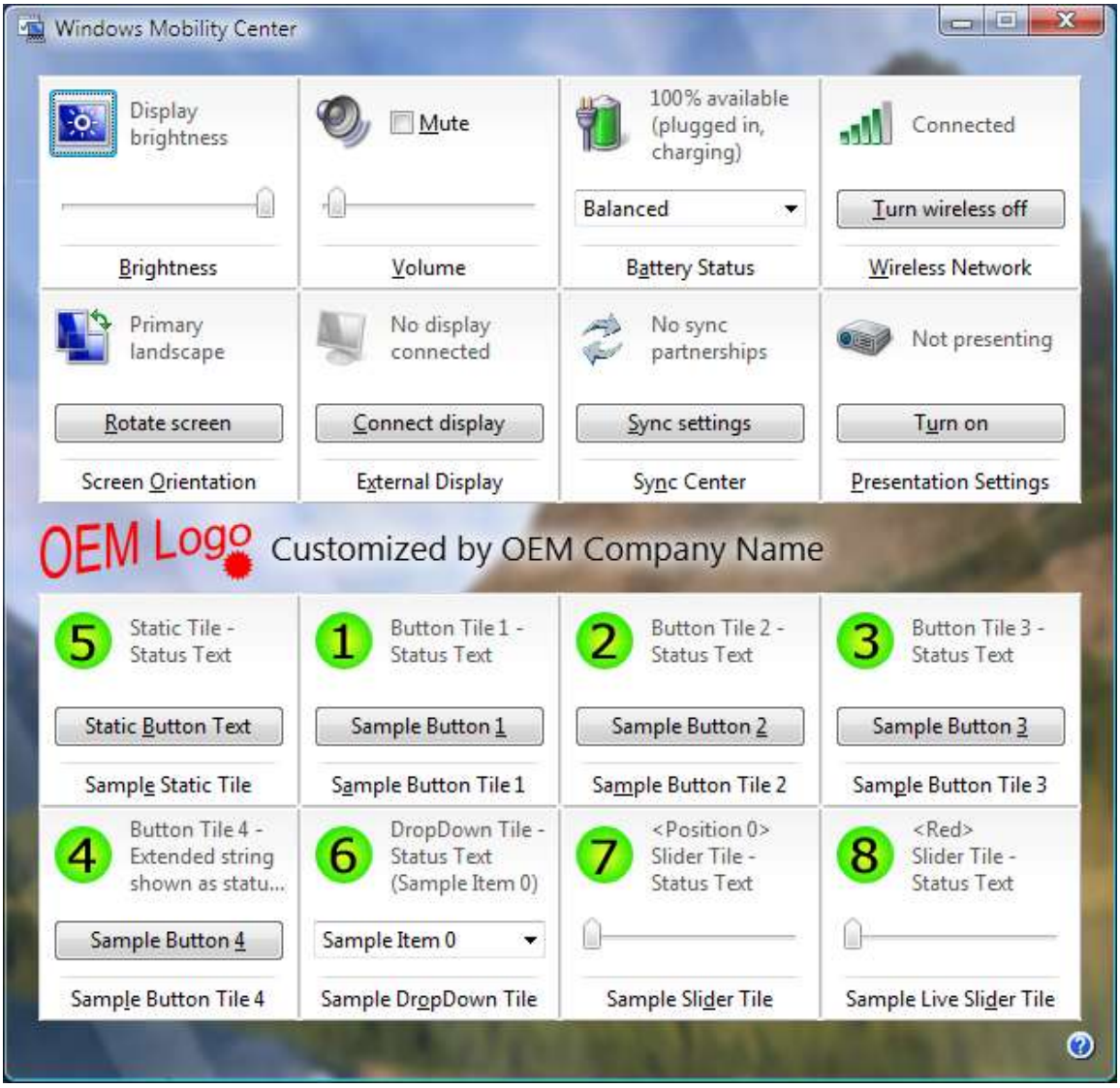


Figure 6: UI presented by Windows Mobility Center that shows 1 static tile and 7 dynamic tiles

Only tiles listed in the **TileOrder** value will be shown in Windows Mobility Center. Any of the sample tiles can be removed from the UI by removing its name from the **TileOrder** registry value.

Changing the text shown on a tile

To change the text presented on one of the tiles, edit the text contained in the string table in the sample solution. To view the string table, open the MobilityCenterSample.rc resource file, and select **string table**. The ID for each string indicates which tile will show the string.

For example, the ID of the button text on the third dynamic button tile is `IDS_BUTTON3_BUTTONTEXT`. To change the text shown on that button, edit the `IDS_BUTTON3_BUTTONTEXT` string. If that string is changed from **Sample Button 3** to **MODIFIED Button Text**, after the solution is built, the Windows Mobility Center presents the UI shown in Figure 7.

Figure 7: UI presented by Windows Mobility Center that shows modified button text

Changing the icon on a tile

To change an icon on a tile, modify the icon resource associated with the tile in the sample solution. In the sample solution, the image for each tile icon is an icon image file. These files are listed in the Resource Files section of the MobilityCenterSample project, and have the name

`icon<n>.ico` where *n* represents the tile associated with the icon.

Each icon is referenced from the icon section of the MobilityCenterSample resource file. To change the icon shown on a tile, first open the MobilityCenterample.rc file in the Visual Studio Resource View. Expand the icon section in the view, and then import the new icon to be shown in the tile. The Windows Mobility Center sample shows 32bpp icons as it is recommended that high quality icons of this bit depth are provided for all tiles. Given that these icon types cannot be generated from within Visual Studio, icons of this bit depth must be generated in another application, and then imported into the Visual Studio project. When changing tile icons, remember that the ID of the final icon must either be specified in the registry (for static tiles), or referenced by the extensible tile server (for dynamic tiles). The icon change is reflected in Windows Mobility Center after the MobilityCenterSample is next built.

Figure 8 shows the UI presented by the Windows Mobility Center after the icon for sample tile 3 has been modified with the steps described previously.

> **Note** It is highly recommended that 32-bpp icons are used in order to present the highest quality images with transparent backgrounds in the UI.

Figure 8: UI presented by Windows Mobility Center that shows a modified tile icon

Changing the action taken when a user clicks an icon or button

The action that is taken when an icon or button on a dynamic tile is clicked is specified by a string in the string table in the MobilityCenterSample project. This is the same string table as that described in the section, *Changing the Text Shown on a Tile*.

The ID of the strings specifying the action taken when a user clicks an icon on a dynamic button tile is `IDS_BUTTON<n>_ICONACTION`, where *n* represents the button tile associated with the action. Similarly, the ID of the action taken when a button is clicked is `IDS_BUTTON<n>_BUTTONACTION`.

To change the action taken when an icon is clicked, edit and save the string identified by the related `IDS_BUTTON<n>_ICONACTION ID`, rebuild the sample, and then run the Windows Mobility Center. For example, to have the icon of button tile 2 launch the “Regional and Language

Options” Control Panel, change the string identified by ID `IDS_BUTTON2_ICONACTION` to **intl.cpl**.

To change the action taken when a user clicks a button, edit and save the string identified by the related `IDS_BUTTON<n>_BUTTONACTION` ID, rebuild the sample, and then run Windows Mobility Center. For example, to have the icon of tile 3 launch Microsoft Paint, change the string identified by ID `IDS_BUTTON3_BUTTONACTION` to **mspaint.exe**.

Changing the range of a slider on a dynamic slider tile

Each slider tile in the sample has hard coded values for the minimum and maximum slider values. Typically a fully functional slider tile will base the minimum and maximum slider values on some system settings.

By default the first slider tile has a maximum value of 50. In order to change this to 100, change the **m_nMax** value in TileSlider.h. Rebuild the MobilityCenterSample.exe and restart Windows Mobility Center with the updated tile. After doing this, the slider range will be from 0 to 100, as shown in Figure 9.

Figure 9: UI presented by Windows Mobility Center that shows a modified slider range

Changing the list of items shown in a dynamic dropdown tile

The sample dropdown tile has a hard-coded list of strings to be shown in the tile dropdown. Typically a fully functional dropdown tile will base the list of strings on strings supplied by the system or some feature external to Windows Mobility Center.

In order to change the strings shown in the dropdown, edit the **FinalConstruct()** function shown in the TileDropDown.h. For example, change the string **Sample Item %d**, to **DropDown Item %d**. Rebuild the MobilityCenterSample.exe and restart Windows Mobility Center with the updated tile. After doing this, the dropdown items all appear with the leading text **DropDown Item**, as shown in Figure 10.

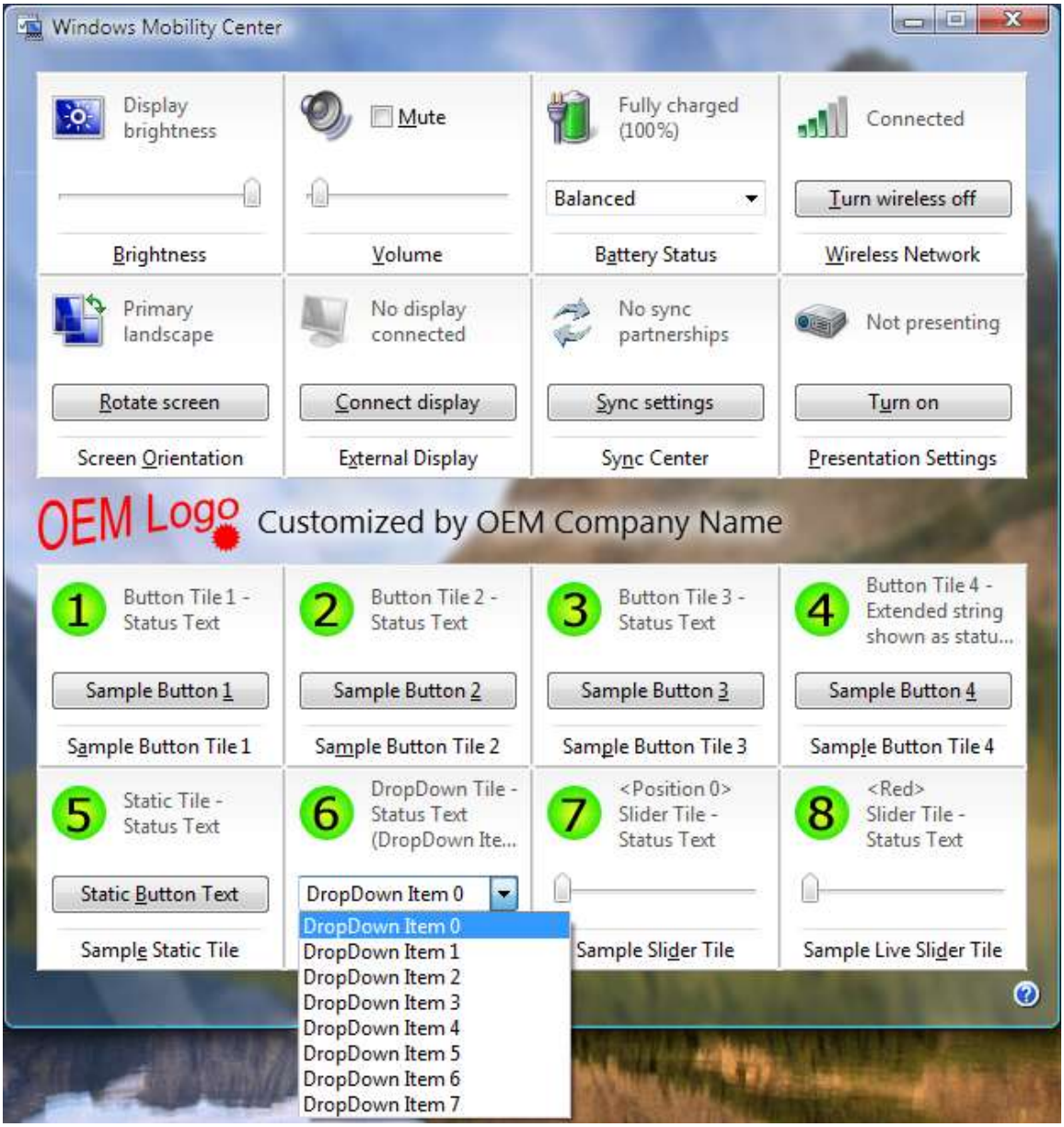


Figure 10: UI presented by Windows Mobility Center that shows modified dropdown list items

Adding specific functionality to a tile

The implementation of the **IMobilityCenterButtonTile** interface for each tile is contained in the C++ file TileButton<n>.cpp, where *n* represents the tile associated with the implementation. For example, TileButton1.cpp loads the string and icon resources shown in button tile 1 by Windows Mobility Center. These C++ files, (and the related header files,) can be modified to perform the mobility-related functionality required by the OEM. Similarly the implementations of the **IMobilityCenterSliderTile** and **IMobilityCenterDropdownTile** interfaces is contained in the TileSlider*.cpp and TileDropDown.cpp files.

If a tile provides some up-to-date status information to the user, then typically the tile registers for related notifications when it is initialized. When the tile later receives notifications, it calls the

MobilityCenterRefresh API and so requests that Windows Mobility Center retrieves updated strings and icons from the tile reflecting the current status.

For example, to have button tile 1 indicate whether the power source is AC or battery following a change in the power source, that tile must be notified of a change in the power source. The tile can be notified of a change in power source by handling the **WM_POWERBROADCAST** message sent by the system. In order to do this, the tile needs to create a window that receives the message. You must modify the **FinalConstuct** function for tile 1 in TileButton1.h for this to occur. Please note that we are not suggesting nor advocating that OEMs create a power tile such as this one; this example is included purely for instructional purposes.

In this example, replace the existing **FinalConstruct** and **FinalRelease** functions in TileButton1.h with the following code:

```
HWND m_hWnd;
BOOL m_fOnBattery;


HRESULT FinalConstruct()
{
     HRESULT hr = S_OK;

     // Cache the module handle for loading resources later.
     m_hInst = GetModuleHandle(NULL);
     if(NULL != m_hInst)
     {
         WCHAR szWindowClass[] = L"TileTest";

         WNDCLASSEX wcex = {0};
         wcex.cbSize         = sizeof(WNDCLASSEX);
         wcex.style          = CS_HREDRAW | CS_VREDRAW;
         wcex.lpfnWndProc    = (WNDPROC)TileTestWndProc;
         wcex.hInstance      = m_hInst;
         wcex.lpszClassName  = szWindowClass;

         // Register the class of a window to receive notifications
         // for the sample tile.
         RegisterClassEx(&wcex);

         m_hWnd = CreateWindow(szWindowClass, NULL, 0, 0, 0, 0, 0,
                     NULL, NULL, m_hInst, (LPVOID)this);
         if(NULL != m_hWnd)
         {
             // Get the current power source.
             SYSTEM_POWER_STATUS sps;
             GetSystemPowerStatus(&sps);

             // This flag will be examined beneath a call to the tile's
             // GetStatusText() method.
             m_fOnBattery = (sps.ACLineStatus == 0);
         }
         else
         {
             hr = E_FAIL;
         }
     }
     else
     {
         hr = E_FAIL;
     }

     return hr;
 }

 static LRESULT CALLBACK TileTestWndProc(HWND hWnd,
```

```
                                        UINT message,
                                        WPARAM wParam,
                                        LPARAM lParam)
{
    switch (message)
    {
        case WM_CREATE:
        {
            // This WndProc is static, so store the pointer to the
            // CTileButton1 object for retrieval later when the window
            // gets a power-related notification.
            CREATESTRUCT *pcs = (CREATESTRUCT*)lParam;
            if(NULL != pcs)
            {
                SetWindowLong(hWnd, GWL_USERDATA, (LONG)pcs->lpCreateParams);
            }

            break;
        }
        case WM_POWERBROADCAST:
        {
            if(PBT_APMPOWERSTATUSCHANGE == wParam)
            {
                CTileButton1* pTile = NULL;
                pTile = (CTileButton1*)GetWindowLong(hWnd, GWL_USERDATA);

                SYSTEM_POWER_STATUS sps;
                GetSystemPowerStatus(&sps);

                // Store the current state for use later.
                pTile->m_fOnBattery = (sps.ACLineStatus == 0);

                // Request a refresh of the tile UI. (Note that a genuine
                // tile would only do this if it knew the UI had actually
                // changed.
                MobilityCenterRefresh(L"Button1");
            }

            break;
        }
        default:
        {
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
    }

    return 0;
}

void FinalRelease()
{
    if(NULL != m_hWnd)
    {
        DestroyWindow(m_hWnd);
    }
}
```

You then replace the existing **GetStatusText** implementation in TileButton1.cpp with the following code:

```
STDMETHODIMP CTileButton1::GetStatusText(BSTR *pbstrStatusText)
{
    HRESULT hr = S_OK;

    if(NULL == pbstrStatusText)
    {
        return E_INVALIDARG;
```

```
        }

        // Allocate a bstr to store the string.
        *pbstrStatusText = SysAllocString(
                                m_fOnBattery ?
                                    L"Power source is Battery" :
                                    L"Power source is AC");
        if(NULL == pbstrStatusText)
        {
            hr = E_OUTOFMEMORY;
        }

        return hr;
    }
```

After the sample solution is rebuilt with these changes, the status text on button tile 1 dynamically reflects the current power source. Figure 11 shows the different UI presented by the tile depending on the power source.

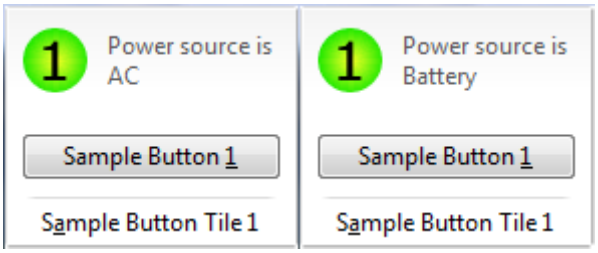


Figure 11: UI presented by Windows Mobility Center that shows an active tile

Deploying the Sample Tiles

To run the sample tiles on any computer, first copy the following files to the target computer:

- `MobilityCenterSample.exe`
- `MobilityCenterSample.reg`

You must complete the following registration steps while running with elevated privileges. Otherwise the COM registration silently fails and the registry settings apply only to the current user. On the target computer, perform the following registration steps from the command prompt:

- `MobilityCenterSample.exe /regserver`
- `regedit MobilityCenterSample.reg`

Finally, change the paths in the registry used to locate resources. For example, change the path held in the `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MobilePC\MobilityCenter\OEMImage` value to:

```
%ProgramFiles%\OEMInstallPoint\OEMCenter.cpl,101
```

Troubleshooting the Sample OEM Tiles

This section is intended to help troubleshoot unexpected behavior when building the OEM tiles that are provided in this sample.

Note that if Windows Mobility Center cannot load any of the resources specified in the registry relating to the OEM extensible tiles or branding data, Windows Mobility Center does not present any extensibility data to the user.

No OEM tiles appear in the UI

- The registration configuration file (MobilityCenterSample.reg) that is provided does not correctly set the resource locations in the registry. It exists to create placeholder entries of all the fields that the OEM must then specify.
- Ensure that the **OEMName** registry references an executable file or DLL that contains the required resource.
- Verify that if an OEM Image has been supplied, it is no more than 120 pixels wide and 36 pixels high. Also verify that the image type is PNG. The **OEMImage** field is not required, but if it is included, it must be valid.
- Verify that the **TileOrder** registry key exists and that it contains strings identical to those used to create the tile-specific registry keys. Try including only a single static tile to verify the path and OEM information is correct.
- Verify that COM registration has been performed while running with elevated privileges with the following command:

  ```
  MobilityCenterSample.exe /regserver
  ```

  Note Registry settings must be made while running as an Administrator. Running as another user results in those registry settings being available for that user only.

- Check that the extensible tile local COM server is running. In the case of the sample solution, when Windows Mobility Center is running, Windows Task Manager lists the following two processes:

  ```
  MblCtr.exe
  MobilityCenterSample.exe
  ```

- Note that if the `MobilityCenterSample.exe` process is still running, then a build of the sample in Visual Studio cannot generate a new MobilityCenterSample.exe file.

A specific tile does not appear in the UI

- For static tiles, verify that the path specified in the registry for the tile resources is correct.
- Ensure that a tile with an Icon Action also has an Icon ToolTip.
- If a dynamic tile returns an error while it is being loaded when Windows Mobility Center is started, the tile will not appear in the UI.

The branding image does not appear

- Check that a registry value named **OEMImage** is in the same registry key as the **OEMName** registry value.

One of the tiles is disabled

- If a tile is successfully loaded when Windows Mobility Center is started, but the tile later returns an error when its UI is refreshed, then the tile will be disabled.

The slider tooltip updates slowly

- Too much time is being spent beneath the **PerformSliderAction()** function while the slider is being moved. Consider taking less action while the slider is being moved, and instead take the more comprehensive action when the slider movement is complete.

The list of items in a dropdown tile is re-populated unnecessarily

- When the dropdown tile UI is refreshed, Windows Mobility Center will call the tile's **GetItems()** function. If **GetItems()** returns S_OK, the list of items will be re-populated, which may be noticeable in the UI if the number of items in the list is large. **GetItems()** should always return S_FALSE if the current list of items in the dropdown is unchanged from the previous set of items returned from **GetItems()**.

Visual Studio fails to save modified source files, or to create the output executable file, or to convert the sample solution from Visual Studio 2003 to Visual Studio 2005 format.

- Ensure that your Visual Studio session has rights to modify or create files in the target folder. Remember that permissions for modifying and creating files in a folder may be treated separately. If the sample is copied to a folder beneath your Users folder, then permissions should not prevent the files from being modified or saved by Visual Studio.

Make use of Trace Viewing Techniques

- The Windows Mobility Center, thus built, supports Trace Viewing. This may yield additional clues, in support of debugging.

FAQ

This section is intended to answer some of the most frequently asked questions about Windows Mobility Center.

Q: What is Windows Mobility Center?

A: Windows Mobility Center provides a discoverable, consolidated UI of frequently modified system settings. Windows Mobility Center gives OEMs a place to plug in additional hardware-specific settings for their computers, and enables OEMs to distinguish themselves with a branded mobility experience.

Q: What are Windows Mobility Center's entry points?

A:

- Windows (Start) Menu/Accessories
- Control Panels/Mobile PC (and Classic)
- Battery Meter (flyout & context menu)
- Personalization CPL (quick link)
- Win+x key binding
- *Recommendation:* OEM Hardware button
    - Tablet hardware button default

Q: What versions of Windows Vista will Windows Mobility Center be available in?

A: Windows Mobility Center will be available in Windows® Vista™ Home Basic through Windows® Vista™ Ultimate Edition, but will only appear on mobile PCs (such as laptops, Tablet PCs, and Ultra-Mobile PCs).

Q: Can I extend the Presentation Settings application?

A: No, the Presentation Settings application cannot be extended by OEMs. If you want to suppress custom notifications during presentations (to achieve parity with system alert suppression), use the **SHQueryNotificationState** API for QUNS_BUSY.

Q: What is the order of events that transpires when Windows Mobility Center loads?

A: Using tracing while debugging can give this information. For reference, Windows Mobility Center loads resources in the following order:

1.) Read the **TileOrder** registry key.

2.) Read the **OEMName** registry key.

3.) If an OEM image is provided, load that **OEMimage.**

4.) For static tiles:

- Load **ButtonText .**
- Load **ButtonAction .**
- Load **TileName .**
- Load **StatusText .**

- Load **Icon.**

5.) For Dynamic tiles (on refresh):

For slider tiles:

- Load **slider range.**
- Load **current position.**
- Load **slider tooltip**.

For dropdown tiles:

- Load **items.**
- Load **selected item index**.

For all dynamic tiles:

- Load **TileName**.
- Load **StatusText**.
- Load **ControlInfo**.
- Load **Icon**.

Q: How are resources referenced in the registry?

A: Resources are referenced through their IDs, appearing with the format: `resource_location <comma> resource_id`. While registry settings are often referenced with offsets rather than IDs, Windows Mobility Center references resources with positive resource IDs (as they appear in the .rc file)

Q: What kinds of controls can OEMs implement in Windows Mobility Center?

A: Static tiles only support buttons. Dynamic tiles support buttons, sliders, or dropdowns. Each extensible tile can only have one of these controls.

Q: Can OEMs modify the Windows tiles?

A: No, OEMs cannot change the tiles provided by Windows Mobility Center. Additional functionality can be added by using the extensibility features provided by Windows Mobility Center.

Q: Will Windows Mobility Center support Themes?

A: Yes, Windows Mobility Center fully supports Windows Vista Themes.

Q: Where are the sample code and library files available?

A: The sample code, header and binary library files that accompany this white paper are available from the OEM channel Web site and are provided on the OPK CD within the Mobility Center folder.

Best Practices

This section covers actions that we recommend OEMs take to make Windows Mobility Center's user experience as positive as possible.

General

- Provide a hardware button to launch Windows Mobility Center.

  Note Windows Mobility Center can be programmatically launched by calling **%windir%\system32\MblCtr.exe**. If Windows Mobility Center is not already running when this command is executed, then Windows Mobility Center will be launched and brought into the foreground. If Windows Mobility Center is already running when the command is executed, but it is not in the foreground, then the running instance will be brought into the foreground. If Windows Mobility Center is already running when the command is executed, and it is already in the foreground, then it will be closed. As such, this command is suitable for executing when the user presses the hardware button, where the user wishes to quickly make an adjustment to some mobile setting and then press the hardware button again to dismiss Windows Mobility Center.

  OEMs can activate Windows Mobility Center by mapping a hardware button to the Windows Mobility Center keyboard shortcut (WINDOWS LOGO KEY + X).

  To programmatically launch Windows Mobility Center such that it is brought to the foreground regardless of whether it was already running, execute the command **%windir%\system32\MblCtr.exe /open**

- Provide a WDDM driver that implements destructive polling through WMI.
- Provide Native Wi-Fi drivers that implement the "wireless on/off" API for wireless networking devices.

User Experience

- Include only those settings in the Windows Mobility Center that are **frequently** used.
- Avoid putting these same settings into the notification area, at the far right of the taskbar. The taskbar notification area is meant purely for persistent, relevant, dynamic status information and user notifications (not launching applications or utilities).
  - Provide UI feedback when settings are adjusted (update status text and button action, if applicable).
  - Provide links to settings that are different from those provided by Windows Vista.
  - Use localized text strings.
  - Avoid using conflicting keyboard shortcuts.
  - Use an OEM logo image with a transparent background. The image resource shown by Windows Mobility Center is PNG data provided by the OEM.

Developer

- Avoid using CPU-intensive COM logic. It causes tiles to load slowly.
- Use Windows messages or notifications for dynamic updates to settings. Avoid the use of polling mechanisms.
- Use tracing for debugging tiles. Messages are logged when the tiles load.
- Provide both 32X32 and 48X48 pixel icons with a 32-bit color depth.
- Do not use per-process variables which assume that the local server process starts when Windows Mobility Center is invoked. When Windows Mobility Center closes, COM will not

close down the local server until a few seconds have elapsed. During this time the user can invoke Windows Mobility Center again, and the extensible tiles will be generated by the same local server process that was running previously. As such, any per-process variables in the local server retain their values between Windows Mobility Center sessions in this case. As part of testing the extensible tiles, it is important that Windows Mobility Center is closed and restarted such that the same instance of the local server provides tiles for both instances of Windows Mobility Center.

Updating Button Tiles from the Beta2 to RC1 Release of Windows Mobility Center

As part of enhancing the Windows Mobility Center extensibility model to support dropdown and slider controls on tiles, the interface definition for tiles with buttons also changed. Any tiles with buttons implemented to be hosted by the Beta2 release of Windows Mobility Center need to be changed to support the RC1 extensibility model. The changes required are listed below for both static tiles, (those with string and icon resources referenced in the registry), and for dynamic tiles, (those implemented as local COM servers).

Static tiles

- Change any **IconLocation** registry values to be **Icon** values.
- Change any **FriendlyName** registry values to be **TileName** values.

Dynamic tiles

- Change the interface from which the tile class is derived in the tile header file from **IMobilityCenterTile** to **IMobilityCenterButtonTile**.
- Change the tile's **GetFriendlyName()** method to be **GetTileName()** in the .h and .cpp files.
- Change the tile's **GetButtonText()** method to be **GetControlInfo()** in the .h and .cpp files.
- In the tile's header files containing the class definitions, change:

  ```
  STDMETHOD(GetIcon) (HICON * phIcon);

  STDMETHOD(GetIconTooltip) (BSTR * bstrIconTooltip);
  ```

  to:

  ```
  STDMETHOD(GetIcon(int nWidth, int nHeight, HICON *phIcon,
                    BOOL *pfActionable, BSTR *pbstrIconToolTip);
  ```

- Merge the tile's **GetIcon()** and **GetIconTooltip()** functions into a single **GetIcon()** function. This step includes:
  - Checking the input icon width and height in order to return the icon that most closely matches the size of the icon when rendered in the UI.
  - Returning the same tooltip as **GetIconTooltip()** previously did.
  - Explicitly setting ***pfActionable** to specify whether the icon is actionable.

Summary of Sample Use

The following steps are required in order for Windows Mobility Center to use tiles included in the Visual Studio sample.

- Install the Windows Mobility Center OPK content in a location where you have access to create and modify content.
- In Visual Studio, open the **MobilityCenterSample.sln** solution.
- If opening the solution in Visual Studio 2005, convert to Visual Studio 2005 format.
- Build the **MobilityCenterSample.sln** and verify that the output status of the build is **1 succeeded, 0 failed, 0 skipped.**
- Copy the following files to your target computer:
    - ..\output\MobilityCenterSample.exe
    - ..\MobilityCenter\MobilityCenterSample.reg
- On the target computer, perform the following tile registration steps:
    - MobilityCenterSample.exe /regserver
    - regedit MobilityCenterSample.reg
- Edit the placeholder registration information beneath the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MobilePC\MobilityCenter registry key, such that the following entries reference the actual location of MobilityCenterSample.exe on the target computer:
    - **OEMName**
    - **OEMImage**
    - **Tiles\StaticButton\StatusText**
    - **Tiles\StaticButton\ButtonLabel**
    - **Tiles\StaticButton\TileName**
    - **Tiles\StaticButton\IconToolTip**
    - **Tiles\StaticButton\Icon**

    Note Do not to leave mismatched double quotes during this edit.
- Invoke Windows Mobility Center and view the eight sample tiles.

Presentation Settings

The Windows Vista Presentation Settings feature available on mobile PCs enables users to configure settings to be applied while a presentation is in progress. By applying these settings, fewer interruptions from the computer occur during the presentation. By default, the settings:

- Prevent a screen saver from turning on during the presentation (assuming group policy allows screen savers to be disabled).
- Prevent the display or CPU from turning off due to lack of user input to the computer.
- Prevent system notifications from appearing near the taskbar notification area.

Other presentation settings include a user-specified desktop wallpaper to be shown while presenting, and a user-specified volume to be applied to the computer. Those settings are not applied by default.

Presentation settings can be turned on or off manually by the user via Window Mobility Center, or automatically when an external monitor or network projector is connected.

OEMs can also turn presentation settings on or off, or invoke the presentation settings configuration window.

Turning presentation settings on

To turn presentation settings on, run the following command:

```
%windir%\system32\PresentationSettings.exe /start
```

After presentation settings are turned on, the icon shown in figure 12 appears in the taskbar notification area. If presentation settings are already turned on before running this command, then running the command has no effect.

Figure 12: Presentation settings icon in the system tray

Turning presentation settings off

To turn presentation settings off, run the following command:

```
%windir%\system32\PresentationSettings.exe /stop
```

Once presentation settings are turned off, the icon shown in figure 12 no longer appears in the taskbar notification area. If presentation settings are not turned on before running this command, then running the command has no effect.

Invoking the presentation settings configuration window

To invoke the presentation settings configuration window, run the following command:

```
%windir%\system32\PresentationSettings.exe
```

The configuration window appears regardless of whether presentation settings were turned on before running the command.

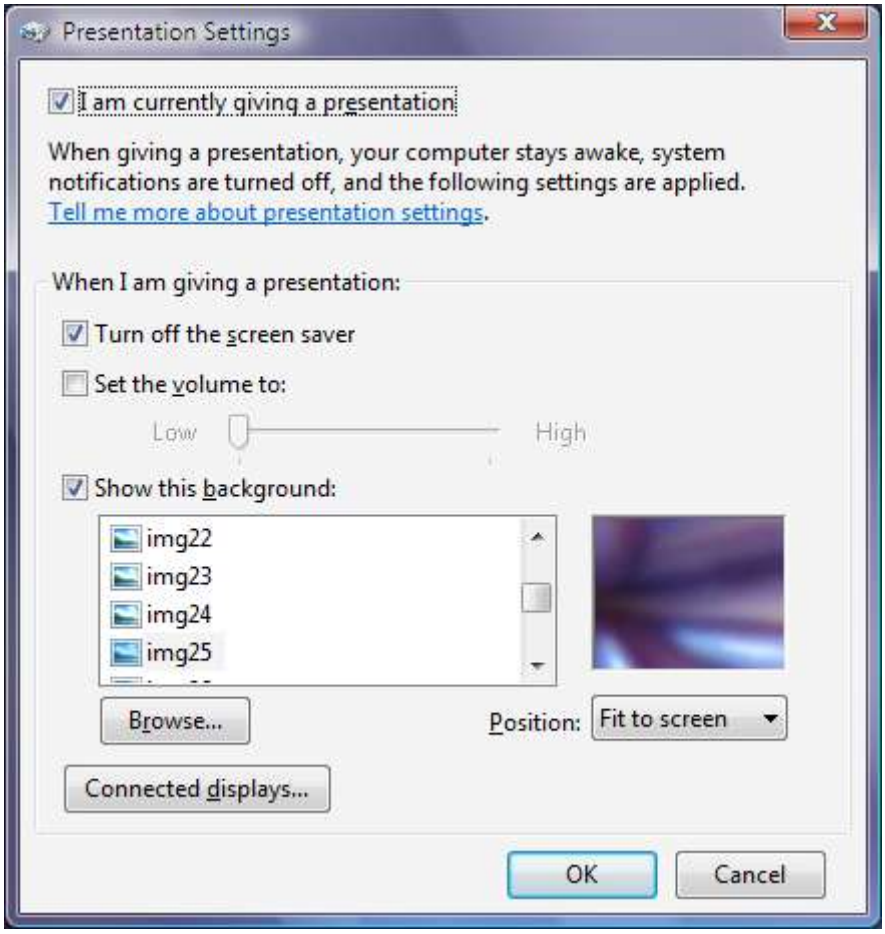


Figure 13: Presentation settings configuration window

Other optional command line arguments for presentation settings

If either of the /start or /stop arguments described above are used, then it must be the first argument supplied. Other available arguments are described in this section.

The /silent command line argument

This command line argument suppresses the message presented to the user when presentationsettings.exe is run on a non-mobile PC.

The /nowallpaper command line argument

If this command line argument is used then presentationsettings.exe does not change the desktop wallpaper, (regardless of the user's current presentation settings configuration settings.)

If presentationsettings.exe is run with the /nowallpaper command line argument and presentationsettings.exe is already running, then the /nowallpaper command line argument has no effect.

Determining the state of presentation settings

In order to determine whether presentation settings are currently turned on, use the Windows Vista API **ShQueryUserNotificationState()**. If presentation settings are turned on, the API returns a value of QUNS_PRESENTATION_MODE, (defined in the shellapi.h header file).

The following code shows how an OEM can determine the state of presentation settings.

```
QUERY_USER_NOTIFICATION_STATE state;

HRESULT hr = SHQueryUserNotificationState(&state);
if(SUCCEEDED(hr) && (QUNS_PRESENTATION_MODE == state))
{
    // Presentation settings are turned on.
}
```

Notification of a change in state of presentation settings

When presentation settings are turned on or off, all top level windows receive a WM_SETTINGCHANGE message. The *lParam* parameter sent with the message is the string "PresentationMode". An application that handles this notification should spend minimal time actually processing the WM_SETTINGCHANGE message itself.