# You’ve Been Tombstoned!

How to Handle Deactivation Events with XNA Game Studio 4.0 on Windows Phone

By Eron Hennessey, XNA Game Studio Documentation

## Introduction

In order to provide a fluid user experience and good performance while conserving valuable battery power, all applications on Windows Phone run exclusively in the foreground, while other applications that were running are either closed or deactivated. Because of this, it's important to know what types of events can cause your game to become deactivated or closed, and how to save and load game state so that you can provide a seamless and enjoyable experience for players.

This article briefly describes conditions that can cause game interruptions, and how you can detect and respond to each.

There are two primary ways that a game can be interrupted on Windows Phone:

1. The game is *deactivated*. This is also known as *tombstoning*, and signals a temporary interruption in game execution. Deactivation might result from top-level communications such as received calls, text messages, alarms, and calendar events. It also occurs if:
   * The user presses the Home or Search buttons while playing a game.
   * The screen is locked when the system times out.
   * The user presses and releases the Power button.
2. The game is *closed*. This signals a complete interruption in game execution, and occurs when:
   * The user has stopped game execution.
   * The game is left in the deactivated state for an extended time.
   * The phone is shut down or is rebooted.
   * Another event occurs that causes the game to stop executing and release its memory to the system.

Notes

Disabling application or user idle detection will cause the system timeout and lock events to *not* deactivate the application. For more information, see [Idle Detection for Windows Phone](http://msdn.microsoft.com/en-us/library/ff941090.aspx) on MSDN.

For a complete overview of the Windows Phone execution model, and for more detail about events that can cause a game to be deactivated or closed, see [Execution Model Overview for Windows Phone](http://msdn.microsoft.com/en-us/library/ff817008.aspx) on MSDN.

## Differences between XNA Game Studio and Silverlight Windows Phone Activation and Deactivation Events

XNA Game Studio and Windows Phone provide a variety of methods to detect and respond to game deactivation, activation, closing, and launching events:

* XNA Game Studio provides the **OnDeactivation**, **OnActivation**, **OnExiting**, and **Initialize** methods of the **Game** class to handle these events. The Silverlight API for Windows Phone provides the **Deactivated**, **Activated**, **Closing**, and **Launching** events of the **PhoneApplicationService** class.

While the XNA **OnDeactivation** and **OnActivation** methods have some correspondence to the phone's **Deactivated** and **Activated** methods, significant differences exist, as shown in the following tables.

### Table 1. Xna.Framework.Game methods

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Initialize | OnActivation | OnDeactivation | Exiting |
| Game launches | X | X |  |  |
| Game is deactivated (tombstoned) |  |  | X |  |
| Game is reactivated | X | X |  |  |
| Guide dialog is up |  |  | X |  |
| Guide dialog is dismissed |  | X |  |  |
| Game shuts down |  |  | X | X |

### Table 2. Microsoft.Phone.Shell.PhoneApplicationService events

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Launching | Activated | Deactivated | Closing |
| Game launches | X |  |  |  |
| Game is deactivated (tombstoned) |  |  | X |  |
| Game is reactivated |  | X |  |  |
| Guide dialog is up |  |  |  |  |
| Guide dialog is dismissed |  |  |  |  |
| Game shuts down |  |  |  | X |

Since **Game.OnActivation** and **OnDeactivation** are called even when the application is launching for the first time, these methods, by themselves, are not useful for detecting if the game has resumed from the tombstone state. For this, the **PhoneApplicationService** class is needed.

Note

Before you can use any **PhoneApplicationService** methods, events, or properties in your XNA Game Studio game, you need to add the Microsoft.Phone assembly to your application. You can do this in Visual Studio by right-clicking **References** in **Solution Explorer**, and then clicking **Add Reference**. You also need to either add a line, using Microsoft.Phone.Shell, to the *using* section of any source files that will be using **PhoneApplicationService**, or you need to specify the complete namespace every time you use a **PhoneApplicationService** member.

## Using PhoneApplicationService.Current.StartupMode to Detect Reactivation

One way to detect reactivation after a tombstoning event is to use the **PhoneApplicationService.StartupMode** property. This property is set before the game’s constructor is called, so you can check this property at any time to determine if it is called either when the game is being launched or when the game is resuming from a tombstoning event. **StartupMode** is set to **Activate** when the game is resuming from a tombstoning event, and it is set to **Launch** when the game is being launched.

Example

protected override void Initialize()

{

if (PhoneApplicationService.Current.StartupMode == StartupMode.Activate)

{

// game is resuming from tombstoning. Restore any transient data that was saved.  
 }

}

Note

This technique still does not solve the problem of detecting if the game is being deactivated because the game is exiting or because of a tombstoning event. To determine the cause, you need to add an event handler for the **PhoneApplicationService.Deactivated** event.

## Using PhoneApplicationService Events to Detect Deactivation and Reactivation

You can use the **PhoneApplicationService.Deactivated** and **Activated** events to determine if the game is being deactivated and reactivated due to a tombstoning event. Checking these properties avoids the issues, described earlier, with trying to use **Game.OnActivation** and **OnDeactivation** to handle tombstoning events.

**To use PhoneApplicationService events to respond to tombstoning**

1. Create event handlers for the **PhoneApplicationService.Deactivated** and **PhoneApplicationService.Activated** events and assign them in your game class constructor.
2. Use the event handlers to store and restore transient and persistent data that your game needs to restore the game's state after a tombstoning event.

More information about storing transient and persistent data is provided in the next section.

## Transient and Persistent Data

To correctly handle deactivation and closed events, you should determine what game data is transient and what data is persistent.

*Transient* data is defined as any data that is necessary to restore the game to the state it was in when the game was deactivated. If, for example, a menu was displayed when the game was deactivated, the same menu should still be displayed when the user returns to the game, including restoring the state of the menu itself (any selections made or data typed). If the user was actively playing the game, the game should appear just as the user left it⎯though it is customary to return to the game in the paused state, if your game supports such a feature.

*Persistent* data is defined as data necessary to save the player's progress, without necessarily restoring the game to the exact state in which it was left. If the game was closed, it is sufficient to save the user's progress in the game. When the user returns to the game, you can return the player to the game's start menu with an option to resume gameplay at the point where the user left the game. There is no need to return to the exact game state, menu, or screen that was displayed when the game was closed.

## Saving and Restoring Transient Data

The game's **PhoneApplicationService.Current** object is maintained when an application has been deactivated. Because of this, the **PhoneApplicationService.State** property can be used to save and restore transient data associated with your game. This property is a **System.Collections.Generic.IDictionary** object that can store key and value pairs of arbitrary data.

For example, in your **PhoneApplicationService.Deactivated** event handler, you can store your transient data as shown in the following code:

void GameDeactivated(object sender, DeactivatedEventArgs e)

{

PhoneApplicationService.Current.State["BugPos"] = bug.Position;

PhoneApplicationService.Current.State["BugRot"] = bug.Rotation;

PhoneApplicationService.Current.State["BugTarget"] = bug.Target;

PhoneApplicationService.Current.State["BugMoving"] = bug.Moving;

PhoneApplicationService.Current.State["foodLocations"] = foodLocations;

}

The same data can be restored in a similar way, in your **PhoneApplicationService.Activated** event handler:

void GameActivated(object sender, ActivatedEventArgs e)

{

bug.Position = (Vector2)(PhoneApplicationService.Current.State["BugPos"]);

bug.Rotation = (float)(PhoneApplicationService.Current.State["BugRot"]);

bug.Target = (Vector2)(PhoneApplicationService.Current.State["BugTarget"]);

bug.Moving = (bool)(PhoneApplicationService.Current.State["BugMoving"]);

foodLocations =

(List<Vector2>)(PhoneApplicationService.Current.State["foodLocations"]);

gameIsPaused = true;

}

Note

The **PhoneApplicationService.State** dictionary cannot be modified after the **Deactivated** event is handled or before the **Activated** event is raised, so it is a good idea to use the **Activated** and **Deactivated** event handlers to modify the game's state data.

## Saving and Restoring Persistent Data

When your game closes, the **PhoneApplicationService** object associated with the game is also released, so you cannot use it to save and restore any persistent data for your game. Instead, you must use the title-specific isolated storage that Windows Phone provides for your game. This storage space persists between launches of your game, so it is a natural place to store persistent game data, such as player progress and high scores.

To use isolated storage, you need to add a line, using System.IO.IsolatedStorage, to the *using* block in any source files that will use any isolated storage classes. Visual Studio includes the assembly necessary to use isolated storage in all XNA Windows Phone Game projects by default, so there is no need to specify additional assemblies when using this namespace.

**To save persistent game data using isolated storage**

1. Call **IsolatedStorageFile.GetUserStoreForApplication** to get an **IsolatedStorageFile** object.
2. Open a file by using **IsolatedStorageFile.OpenFile** with **FileMode.Create**, which returns an **IsolatedStorageFileStream** object that can be used to write to the file.
3. Use **IsolatedStorageFileStream.Write** to write bytes of data to the file.

**Example**

// Save the game state (in this case, the high score).

IsolatedStorageFile savegameStorage = IsolatedStorageFile.GetUserStoreForApplication();

// open isolated storage, and write to the file.

IsolatedStorageFileStream fs = null;

using (fs = savegameStorage.CreateFile(SAVEFILENAME))

{

if (fs != null)

{

// just overwrite the existing info for this example.

byte[] bytes = System.BitConverter.GetBytes(highScore);

fs.Write(bytes, 0, bytes.Length);

}

}

To reload the data when your game is relaunching, a similar procedure can be followed.

**To load persistent game data using isolated storage**

1. Call **IsolatedStorageFile.GetUserStoreForApplication** to get an **IsolatedStorageFile** object.
2. Open the file by using **IsolatedStorageFile.OpenFile** with **FileMode.Open**, which returns an **IsolatedStorageFileStream** object that can be used to read from the file.
3. Use **IsolatedStorageFileStream.Read** to read bytes of data from the file.

Example

using (IsolatedStorageFile savegameStorage =

IsolatedStorageFile.GetUserStoreForApplication())

{

if (savegameStorage.FileExists(SAVEFILENAME))

{

using (IsolatedStorageFileStream fs =

savegameStorage.OpenFile(SAVEFILENAME, System.IO.FileMode.Open))

{

if (fs != null)

{

// Reload the saved high-score data.

byte[] saveBytes = new byte[4];

int count = fs.Read(saveBytes, 0, 4);

if (count > 0)

{

highScore = System.BitConverter.ToInt32(saveBytes, 0);

}

}

}

}

}

Note

For more information about using isolated storage to write and read data associated with your game, see [Writing Data (Windows Phone)](http://msdn.microsoft.com/en-us/library/ff604992.aspx), in the XNA Game Studio documentation, and [Isolated Storage for Windows Phone](http://msdn.microsoft.com/en-us/library/ff626522.aspx), in the Windows Phone Development documentation.

## Testing Interruptions with the Windows Phone Emulator

After you have implemented code for handling deactivation, reactivation, closing, and relaunching of your game, you can use the Windows Phone Emulator to test your code while it’s running with the Visual Studio debugger.

The following procedure takes your game through all of the states described earlier, allowing you to test your code to determine if it is functioning correctly.

**To test launching, deactivation, reactivation, and closing of your game**

1. Debug your game with Windows Phone 7 Emulator selected in the **XNA Game Studio Deployment Device** item in Visual Studio’s toolbar.  
     
     
     
   The emulator starts and your game will launch.
2. Click the **Home** button (**Start** in the emulator) on the emulator display.

The game is deactivated and you are shown the emulator’s Home screen.

1. Click the **Back** button on the emulator display.

The game is reactivated.

1. Stop debugging in Visual Studio while the emulator is still running.

Doing so allows you to test game launching after a full closing event.

1. Restarting debugging in Visual Studio without restarting the emulator.

Doing so launches your game.

## Additional Resources

This article has covered the basics of handling interruptions on Windows Phone. For more information, refer to the *XNA Game Studio 4.0* and *Windows Phone Development* documentation sets, which are included with the Windows Phone Developer Tools and are also available online on MSDN.

Documentation that directly pertains to this subject matter is provided by the following selection of links.

### XNA Game Studio 4.0 (MSDN)

[Handling Interruptions on Windows Phone](http://msdn.microsoft.com/en-us/library/ff827866.aspx)

[Writing Data (Windows Phone)](http://msdn.microsoft.com/en-us/library/ff604992.aspx)

### Windows Phone Development (MSDN)

[Execution Model Overview for Windows Phone](http://msdn.microsoft.com/en-us/library/ff817008.aspx)

[How to: Preserve and Restore Application State for Windows Phone](http://msdn.microsoft.com/en-us/library/ff967547.aspx)

[Idle Detection for Windows Phone](http://msdn.microsoft.com/en-us/library/ff941090.aspx)

[Isolated Storage for Windows Phone](http://msdn.microsoft.com/en-us/library/ff626522.aspx)