[ISSN: 2231-4946]

Handling Tombstoned in Windows Phone

Shivraj Bandi[1], Minal Moharir[2], Nagaraj G Cholli[3]

[1]*Department of Information Science & Engg, R V College of Engineering, Bangalore*

[1]shivrajbandi@live.in
[2]moharirminal@gmail.com
[3]nagaraj.cholli@gmail.com

***Abstract*— Nowadays, well turned-out phones permit us to carry our computers within our handsets with us and Mobile phone Applications became an ever-growing and very spirited field of software production. Microsoft also launched its Windows Phone7 [*1*] in India. There are lots of new features added to this windows phone. This windows phone 7.5 is code named as mango. In the earlier version of windows phone it doesn't contain the concept of multitasking which is the android's big feature. In Android the applications that stand on the background consumes the resources, where user thinks that they have quitted the application. In the same way our windows phone uses this with a concept named tombstoning [*2*]. In this paper we will discuss about tombstoning and how to handle this in windows phone.**

***Keywords*— Windows Phone, Tombstone, Multitasking, Mobile Applications, Mango.**

I. INTRODUCTION

Nowadays, well turned-out phones permit us to carry our computers within our handsets with us and Mobile phone Applications became an ever-growing and very spirited field of software production. In Android the applications that stand on the background consumes the resources, where user thinks that they have quitted the application. In the same way our windows phone [*1*] uses this with a concept named tombstoning.

In windows phone it is mainly intended to provide a quick to respond system all the time. The tactic behind this is not to run two or more applications in the background at the same time. By this strategy the applications will not fight for the resources, will not slow down the foreground application and challenging the battery.

To achieve this windows phone 7 will tolerate only one application to run on the foreground. When an application is sent to background and not closed, it is tombstoned [*2*]. Tombstoning means the application is static and lifeless, but the operating system will preserve the state information of that application. If the user comes back to the tombstoned application, the state information is passed to the application earlier than it is restarted. This allows the developer to bring back the application to a state it was in when it was tombstoned. This gives a false impression that the application was there all the time.

It ensures that when your application is activated by the user returning to a previously running application, the user is taken to an experience that is consistent with the one that was shown when the application was deactivated. It should not be evident to the user that the application was terminated and restarted.

II. TOMBSTONE PROCESS

There are certain events happening in the life cycle of an application. The application moves from one state to another state in answer to user events. Each transition generates actions to which we can respond. The states include activating, running and deactivating. These states are shown in the figure 1.

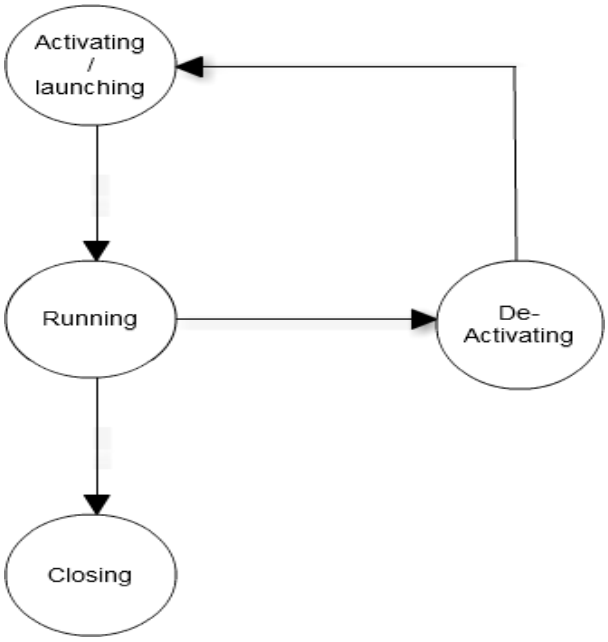

Figure 1: Tombstoning process

The activating/launching state is the one which raise a new request of the application. This always becomes visible as a new occurrence. Thus, in handling the launching event you do not use transient state to restore an on hand session, though you are free to check isolated storage for new-instance related information.

The isolated storage is nothing but the persistent storage. With isolated storage, data is always isolated by user and by assembly. The data saved in the store can be any kind of data, from user preference information to application state.

The running state appears after the launching or the activating event is handled. Later the running state can either go to the closing or deactivating state. So before it goes to deactivating state we store the data at the deactivation event handler [*5*]. While deactivating the application can have only 10 seconds to store all the data, so it is healthier to store the data incrementally. Rather than storing all the data at a once it is better to have some data stored before the deactivation event handler is triggered.

As it is said the running state can end at closing stage or at the deactivating state. In first case when the user press back button till the application start screen appears, this will be a sign of closing event and signifies that the application is terminated. So as this is not the suspension, we should store the persistent data in the isolated storage and will not store the transient data.

In the later case of running state to deactivating state take place when the user changes your foreground application with another application. We will receive a deactivating even when this situation happens. This definitely does the same as it did with the closing event. Now here we store not only the permanent data but also the transient data [*4*]. This is because when you return back to your application from deactivating the same state should be visible when resumed.

We store the transient data in the state dictionary (PhoneApplicationService). As we cannot know whether the application will reactivate again or not, so we store the transient data in the state dictionary, this helps when we come back to application again. When it receives a deactivating event it has only 10 seconds to do all the actions and store all the data. If not done within those 10 seconds the application can get terminated.

III. TOMBSTONE SCENARIOS

There are several cases where the application can get tombstoned. Here are some of the cases .The application will get tombstoned immediately when the user navigates to a new application by pressing the start key, when pressing search button, when user responds to a toast notifications, when interrupt activity occurs, when application uses a launcher or chooser to start a process.

There are some cases where if you launch one of partial number of local services on behalf of application, the application will not automatically be tombstoned. The services that can be launched without being tombstoned are photochooserTask, CameraCaputreTask, PhoneNumberChooserTask, MediaPlayerLauncher, etc.

Closing your application through going back from the first screen causes your application to shut down. The rest of the ways application stops running causes something informally called tombstoning.

IV. HANDLE TOMBSTONE

As we know that there are four events in the application launching, closing, activated and deactivated. When anyone opens the application in the visual studio 2010 express edition they can find the App.xaml class file. In this class we have four event handlers Application_Launching, Application_Closing, Application_Activated, and Application_Deactivated [*6*]. The first two are for normal application set up and shutdown. Then later two are for tombstoning cases.

```
Private void SavePersistentData ()
{
        var store =
IsolatedStorageSettings.ApplicationSettings;
        store.Save ();
}

Private void RestorePersistentData ()
{
        var store =
IsolatedStorageSettings.ApplicationSettings;
}
```

Initially when the application is launched from the start screen the application gets activated. Many operations can be done after the application is activated. When an application is deactivated at first the process running is sent to the blocked state. Then the application can go to deactivating state or closing state. If the application goes in to closing state the process in the blocked state is also stopped and never recovers.

```
void Application_Closing (object sender,
ClosingEventArgs e)
{
         SavePersistentData ();
}
```

If the application goes to the deactivating state then the process which is in the blocked state is dispatched to the running state. And this is a typical tombstone case [*3*]. The process which is blocked is then started and goes on with the operations. When the process, starts the application is constructed as same way when process is stopped before. If Application is constructed the state changes from deactivated to the activated state and then to the running state. Generally the App can go to the closing state or deactivating state from the running state. This handling is shown in the Figure 2 below.

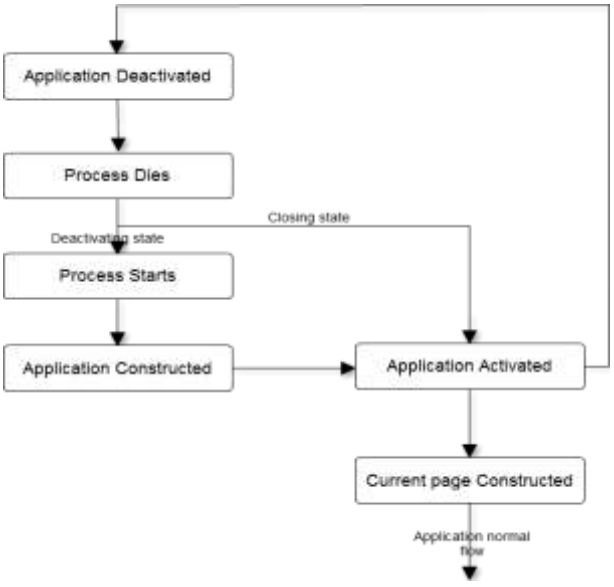

Figure 2: State flow while tombstoning

In the Launching and Closing event handlers we typically save data to an isolated storage to store persistent data. Whereas in the activated and deactivated event handlers we use Current state to store transient data.

```
Private void Application_Launching (object sender,
ClosingEventArgs e)
{
        RestorePersistentData ();
}
Private void Application_Deactivated (object sender,
ClosingEventArgs e)
{
        SavePersistentData ();
        Var store = PhoneApplicationService.
Current.state
}
```

The small pattern for structuring the code that cover all the possibilities is to make sure that code for saving persistent data is called in deactivating and closing event handlers. The code for restoring the persistent data is called in activated and launching event handlers.

```
private void Application_Activated (object sender,
ClosingEventArgs e)
{
        RestorePersistentData ();
        var store = PhoneApplicationService.
Current.state;
}
```

To maintain the state of a page in the app, if it gets tombstoned there is a library called Tombstone Helper [*6*] which adds extension methods to PhoneApplicationPage.

```
protected override void OnNavigatingFrom
(NavigatingCancelEventArgs e)
{
  this.SaveState (e);
}

protected override void OnNavigatedTo
(NavigationEventArgs e)
{
  this.RestoreState ();
}
```

If we use this code then the contents, checked states, positions of textboxes, radio buttons, sliders, etc. will be preserved.

The proper place for the code that saves the data resides in the OnNavigatingFrom method. In this method we are calling the helper extension method to save state for each member of a page that want to save.

The proper place for the code that loads the data when returning from tombstoned state resides in OnNavigatedTo method. In this method we are calling the helper extension method to load state for each member of a page that want to load.

V. Conclusion

The tombstoning mechanism is not a replacement for multitasking, but it can be used to provide an excellent user experience that is nearly indistinguishable from multitasking in many cases, with relatively little effort on your part. This tombstoning is not the real multitasking [*7*]. It is user's responsibility to save all the application and page state, so that when the page gets reloaded it can recover properly and reload the previous data.

VI. References

[1] Microsoft (2011), "*Windows Phone Page*". Available: http://www.microsoft.com/windowsphone/en-us/default.aspx.

[2] MSDN (2011), "*Execution cycle: Tombstone*". http://msdn.microsoft.com/en-us/library/ff817008 v=vs.92).aspx

[3] Chia-Chi Teng, Richard Helps, "*Mobile Application Development: Essential New Directions for IT*".2010 Seventh International Conference on Information Technology.

[4] Roger Ouch and Blake Rouse, *"Developing a Driving Training Game on Windows Mobile Phone Using C# and XNA*". The 16th International Conference on Computer Games

[5] Nicks.NET Travels (2011), "*Tombstone Cases*".http://nicksnettravels.builttoroam.com/post/2011/03/07/ Windows-Phone-7-Tombstone-Frustration.aspx

[6] Codeplex (2011), "*Tombstone Helper*" http://tombstonehelper.codeplex.com/

[7] Fragile development (2010), "*Multitasking*" http://fragiledevelopment.wordpress.com/2010/10/12/multitasking-on-windows-phone-7/