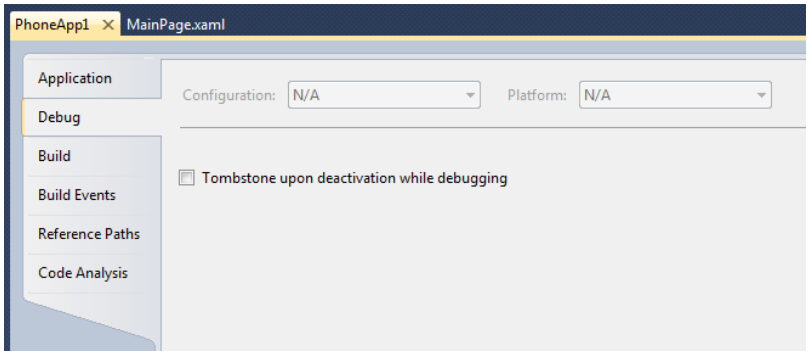
debugging. If you then press the Back button you will find that the program will resume execution after being reactivated.

Forcing a Program to be removed from memory

When we are testing your program we must make sure that it works correctly even when it is removed from memory and then reactivated. We know that the Windows Phone will only remove programs when memory is short. Rather than force us to try to load up memory and force our programs to be removed, Visual Studio provides a way that we can request programs to be removed from memory as soon as they are made dormant.



In the Debug tab of the properties for a project there is a textbox that, if checked, requests that a program be removed from memory when it is deactivated while debugging. This allows us to test for proper behaviour of a program even when it is completely removed from memory.

Fast Application Switching and Design

It is unfortunate that we have to do so many complicated things to give the user the impression that our application never actually stops. However, it is putting in the effort to make sure that your applications work correctly in this respect as it adds a great deal to the user experience. It is worth spending some time working with this as it is a good way to brush up on your understanding of events and dictionary storage, amongst other things. It is also an interesting way to discover that the road to producing a good user experience is not always a smooth one, particularly if you are writing programs in a constrained environment.

An XNA game can also connect to events that will be generated when it is stopped or deactivated and you can make games that store game state and high scores when the user moves away from them in exactly the same way.

The solution in *Demo 01 Complete Captains Log* contains a Windows Phone Silverlight log program that works correctly when deactivated.

9.3 Launchers and Choosers

The next part of our application development is concerned with how we use the Launchers and Choosers in the phone. These are the built in behaviours that are provided to allow our programs to perform standard actions. A Launcher is means by which our program can fire off a phone feature which does not return any data. A Chooser is where a phone feature is used to select something. Your application can then do something with the item returned.

Launchers and choosers make use of deactivation in that when a program activates launcher or chooser it is automatically deactivated. When the launcher or chooser finishes the program is reactivated.

Using a Launcher

These are all the launchers, along with a brief explanation of what each does:

`PhoneCallTask`	starts the Phone application with a particular phone number and display name selected. Note that the program cannot place the call, the user must initiate this
`EmailComposeTask`	our program can set properties on an email and then launch a task to allow the user to send the message.
`SmsComposeTask`	starts the Messaging application and display a new SMS message. Note that the message is not sent automatically.
`SearchTask`	starts the Search application using a query you provide
`WebBrowserTask`	starts the Web browser and displays the URL you provide.
`MediaPlayerLauncher`	starts the Media Player application and play a given media file
`MarketplaceReviewTask`	your program can start the review process of your application
`MarketplaceDetailTask`	your program can show details of a particular application in the market place
`MarketplaceHubTask`	starts the Windows Phone Marketplace client
`MarketplaceSearchTask`	your program can search the Windows Phone Marketplace for a particular application and then show search results
`BingMapsDirectionsTask`	your program can create a set of Bing Maps directions between to places, for driving or walking
`BingMapsTask`	your program can display a map centred on the current location of the phone or a specified location
`ConnectionSettingsTask`	your program can use this task to allow users to change the connection settings of the phone
`ShareStatusTask`	your program can fill in a status message that the user can then send to social networks of their choice
`ShareLinkTask`	your program can set up a url that can then be shared by the user over the social network of their choice

Each of these tasks can be started by your application and when the task is complete your application will be re-activated. Of course there is no guarantee that our application will regain control at any point in the future. The user may instead go off and do lots of other things instead of returning to the program.

Adding an email feature to JotPad

We might decide to add a Mail feature to the Jotpad program. By pressing the Mail button a user can send the contents of the jotpad as an email. The image above shows how this might work.

```
private void mailButton_Click(object sender, RoutedEventArgs e)
{
    sendMail("From JotPad", jotTextBox.Text);
}
```

When the mail button is clicked the event handler takes the text out of the textbox and calls the `sendMail` method to send this. It also adds a `From` message as well. The `sendMail` method is very simple:

```
private void sendMail(string subject, string body)
{
    EmailComposeTask email = new EmailComposeTask();

    email.Body = body;
    email.Subject = subject;
    email.Show();
}
```

This creates an instance of the `EmailComposeTask` class and then sets the `Body` and `Subject` properties to the parameters that were provided. It then calls the `Show` method on the task instance. This is the point at which the JotPad application is tombstoned to make way for the email application. When the user has finished sending the email the JotPad program will resume.

To make use of the tasks a program must include the Tasks namespace:

```
using Microsoft.Phone.Tasks;
```

The solution in *Demo 02 Email JotPad* contains a Windows Phone Silverlight jotter pad that can send the jotting as an email. Note that this will not work properly in the Windows Phone emulator as this does not have an email client set up. However, you can see the deactivated behaviour in action as the program tries to send an email and displays a warning instead.

Using a Chooser

These are all the choosers that are available:

`CameraCaptureTask`	starts the Camera application for the user to take a photo
`EmailAddressChooserTask`	starts the Contacts application and allows the user to select a contact's email address
`PhoneNumberChooserTask`	starts the Contacts application and allows the user to select a contact's phone number.
`PhotoChooserTask`	starts the Photo Picker application for the user to choose a photo.
`SaveEmailAddressTask`	saves the provided email address to the Contacts list Returns whether or not the save was completed.
`SavePhoneNumberTask`	saves the provided phone number to the Contacts list. Returns whether or not the save was completed
`AddressChooserTask`	your program can request that the user select an address from the contact the phone
`GameInviteTask`	your program can invite another phone user to a multiplayer gaming session
`SaveContactTask`	your program can populate a contacts entry and allow the user to save this to a contact list on the phone. Returns whether or not the save was completed.
`SaveRingtoneTask`	your program can give the user the option to save an audio file in an appropriate format as a ringtone for the phone

A chooser works in exactly the same way as a launcher with just one difference. A chooser can return a result to the application that invoked it. This is done in a manner we are very used to by now; our application binds a handler method to the completed event that the chooser provides.

Picture display

We can explore how this works by creating a simple picture display application this will allow the user to choose a picture from the media library on the phone and then display this on the screen. We must create the chooser in the constructor for the `MainPage` of our application:

```
PhotoChooserTask photoChooser;

// Constructor
public MainPage()
{
    InitializeComponent();

    photoChooser = new PhotoChooserTask();

    photoChooser.Completed +=
          new EventHandler<PhotoResult>(photoChooser_Completed);
}
```

This constructor creates a new `PhotoChooser` value and binds a handler to the completed event that it generates. The completed event handler uses the result returned by the chooser and displays this in an image.

```
void photoChooser_Completed(object sender, PhotoResult e)
{
    if (e.TaskResult == TaskResult.OK)
    {
        selectedImage.Source =
                  new BitmapImage(new Uri(e.OriginalFileName));
    }
}
```

This method sets the source of the image to a new `BitMapImage` that is created using the address of the photo result. Note that the `TaskResult` property of the result allows our program to determine whether or not the user chose something to return.

The final link we need to provide is the event handler for the Load button that starts the process off.

```
private void loadButton_Click(object sender, RoutedEventArgs e)
{
    photoChooser.Show();
}
```

This method just calls the `Show` method on the chooser to start the choosing process running.

The solution in *Demo 03 PictureDisplay* contains a Windows Phone program that you can use to select an image from the media storage. This program will not work on the Windows Phone emulator which does not have any images in is media store. It will also not work on the Windows Phone device if you use the Zune software to connect the phone to Visual Studio. This is because when the phone is connected to Zune the media content is not available. However, if you disconnect the phone and then run the application it will work correctly.

9.4 Background Processing

From the previous sections we know that a Windows Phone will only allow one program at a time to be active on the phone. However, there are occasions when it would be very useful for some part of an application to be active when it is not visible to the user. The Windows Phone operating system provides this in the form of "background processing" features, where a developer can create code which will run when their application is not active.

Note that this is **not** the same as allowing a given program to run in the background. The background components are entirely separate, have no user interface and have strict limitations placed on what they can and can't do. They only run in specific situations so that the phone operating system can ensure that they will have an impact on the user experience. The phone operating system will shut down background tasks if