Step-by-step: the path from COBOL to mobile

Creating an Air Miles Calculator demo in Visual COBOL

START

VISUAL COBOL

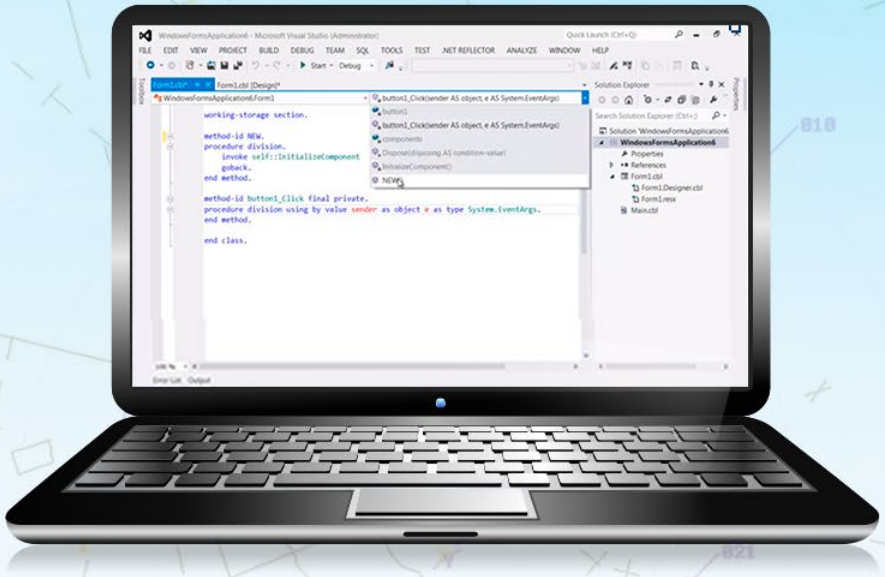
MICRO FOCUS

THE MOVE TO MOBILE STARTS HERE

A step-by-step guide to developing mobile apps using Visual COBOL

If you think mobile is beyond the capabilities of COBOL technology, think again – it's easy to embrace mobile technology with Visual COBOL using this simple step-by-step guide.

As developers, we have to act fast to keep up with end user demands, one being the growth in mobile use, and the corresponding demand for mobile access to applications. With Visual COBOL, delivering great mobile apps doesn't have to be complicated. Follow this guide, download the sample code provided, and see how you can develop a new mobile application based on your existing COBOL code.



We've documented the process
to show you just how easy it can be.

Get started now
See the step-by-step guide

1. Decide what you need to mobilize

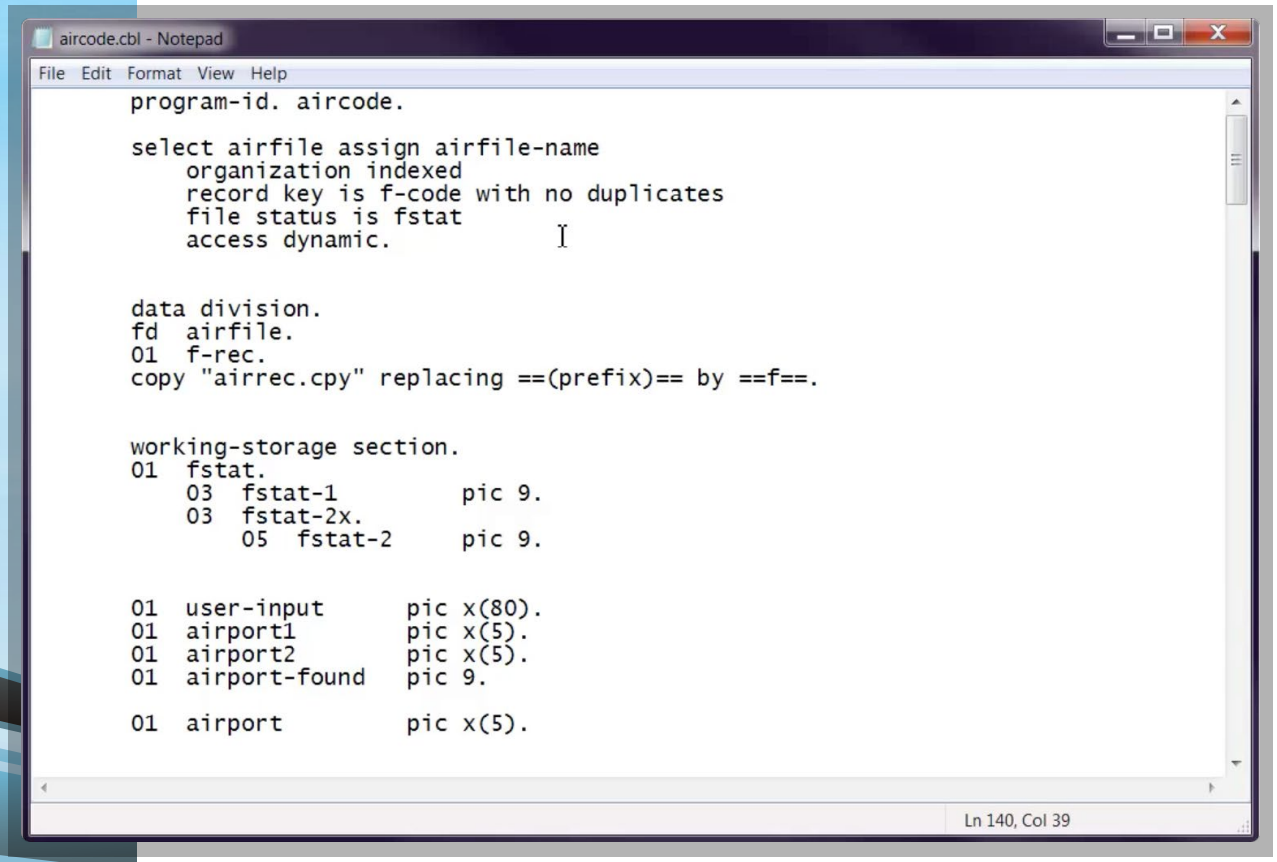
Mobile is now a must-have

It wasn't so long ago that mobile support was considered a 'nice-to-have' option for your application. Today, mobile is fast becoming a must-have for applications. Simply leaving it out of your next product release may not be something you can afford to do.

So where do you begin? It needn't be a daunting prospect – your existing COBOL applications contain vast amounts of functionality that you can expose as services for a mobile application. There's no need to develop a complete new set of capabilities – just use what already works well as the basis of your new mobile application.

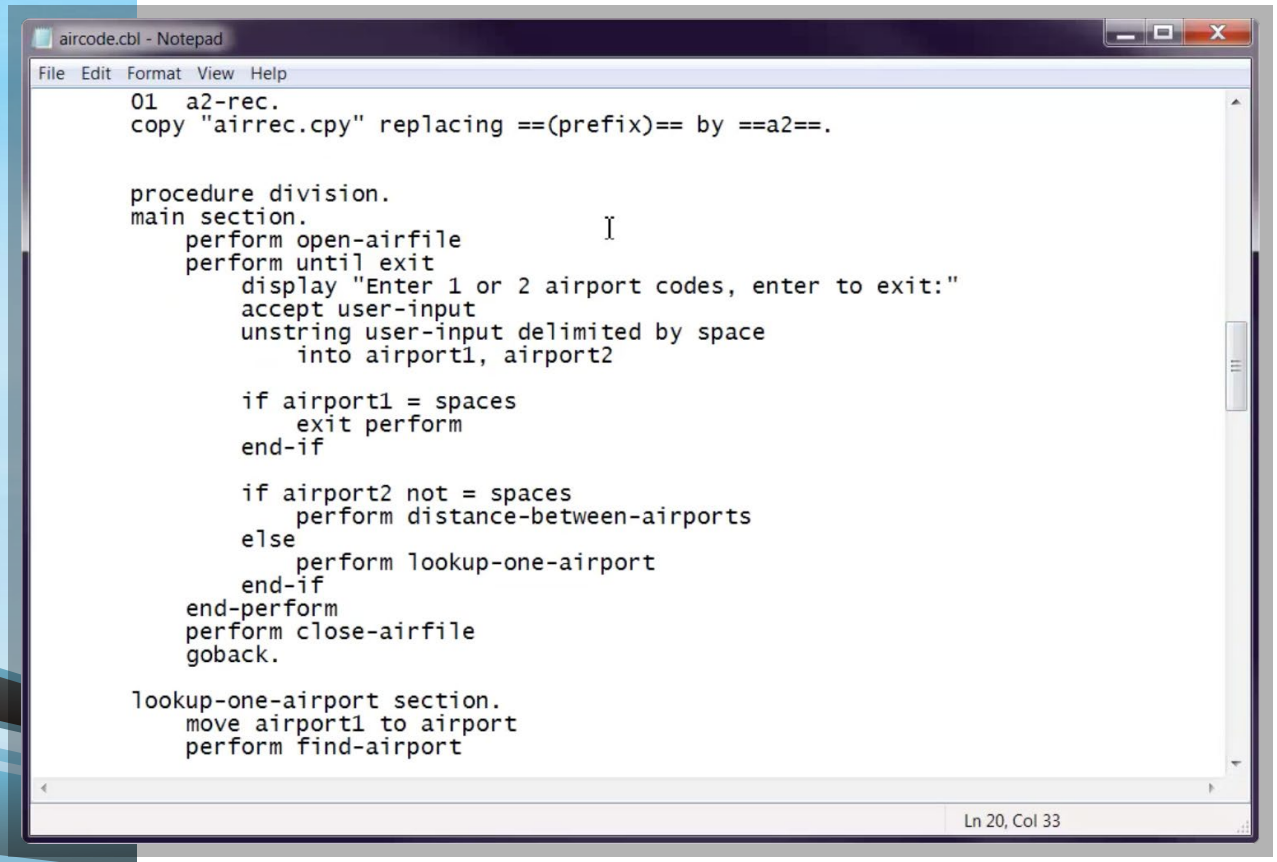
The toughest job is deciding which parts of your current application you want to give a mobile facelift. It's best to consider the overall application experience as it stands today, and look for functionality that will be of most value to a mobile user. That might be a reporting function, a booking system – whatever is most useful and works well within a typical mobile interface. Do start simply: a small but useful piece of code can be used to build a pattern for future expansion.

1. Decide what you need to mobilize



The first step in getting from COBOL to mobile is deciding which parts of your current app you want to give a mobile facelift. Here's a simple COBOL program to get us started. It uses an indexed data file for persistent storage.

1. Decide what you need to mobilize



And all the business logic is contained within the procedure division.

2. Decide if it's an HTML5 or native application

The pros and cons

There are several ways you can create a mobile application: two of the most common are writing a native app for the device (such as an iPhone app), or creating an HTML5 application that can be accessed using the device's web browser. There are pros and cons to either approach. Usually, native apps offer a richer user experience on the device, but while an HTML5 app might not be able to offer quite the same look and feel as a native app, it makes up for it in portability – HTML5 applications can be accessed from a wide variety of different devices allowing you to deliver a mobile user experience across multiple devices, much faster.

In our demo, we're going to use HTML5 to target multiple mobile devices. Since we're using web services to access the server-side COBOL applications, you can reuse the same web services to write a native app if you choose to.

3. Identify the COBOL code

Bring your application functionality together

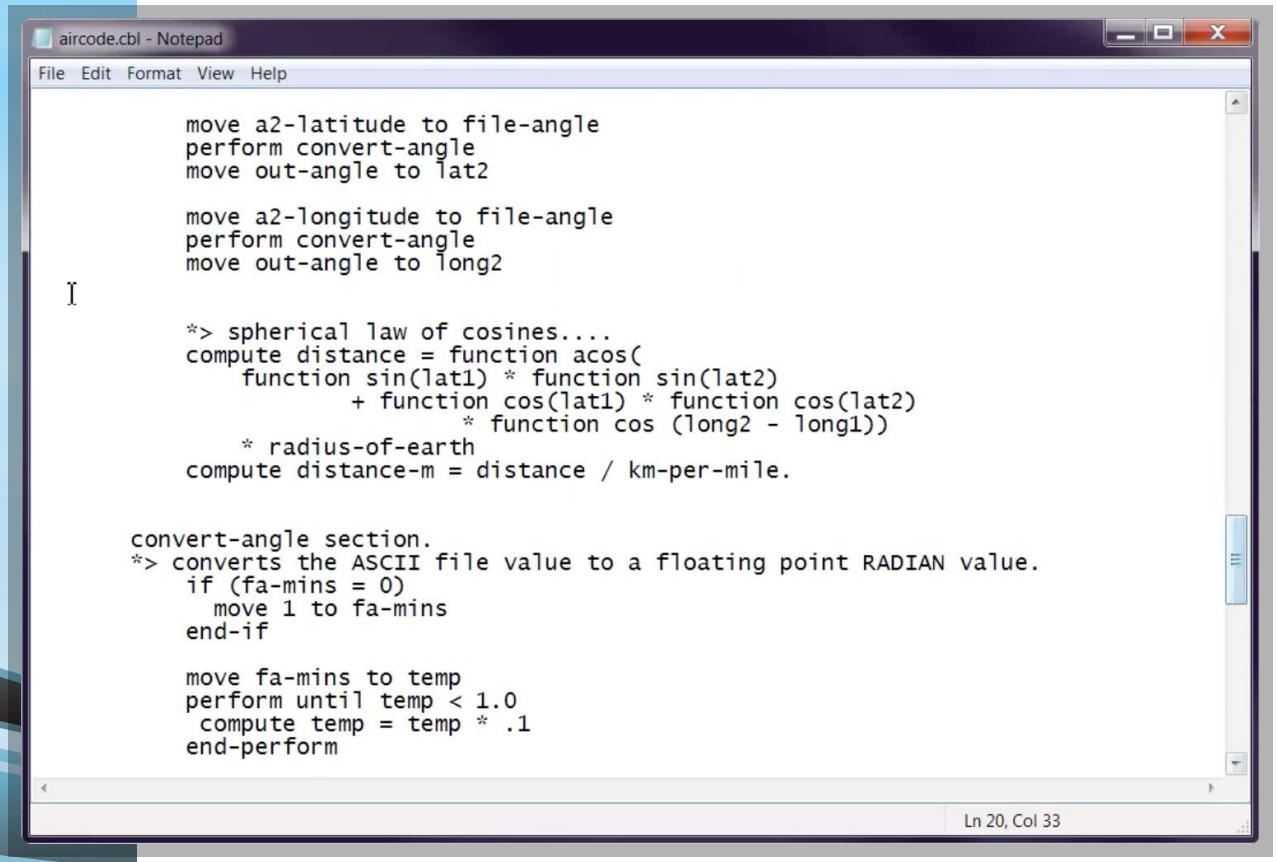
Once you've decided on the key functions, isolate this code from the rest of the main application. You might need to strip away code to achieve this, especially if there is an existing user interface in place. Now is also a good time to identify any application dependencies since these must be available too (or deleted out of the code if not required).

Your goal is to arrive at a callable API that provides access to discrete functions for your mobile application. Your existing application may already have this API architecture in place, in which case you won't need to worry about this. Otherwise, look to bring your application functionality together under a single program or series of programs that can be called directly to provide the results.

Remember, web services are generally stateless architectures – so you should treat every invocation as the initial program call and initialize elements, such as data items and opening files, as you need them. SOA approaches like REST pass state between client and server within the HTTP request, allowing you to pick up from where you last left off for any given user request. You can also take care of these aspects later on from the comfort of an IDE.

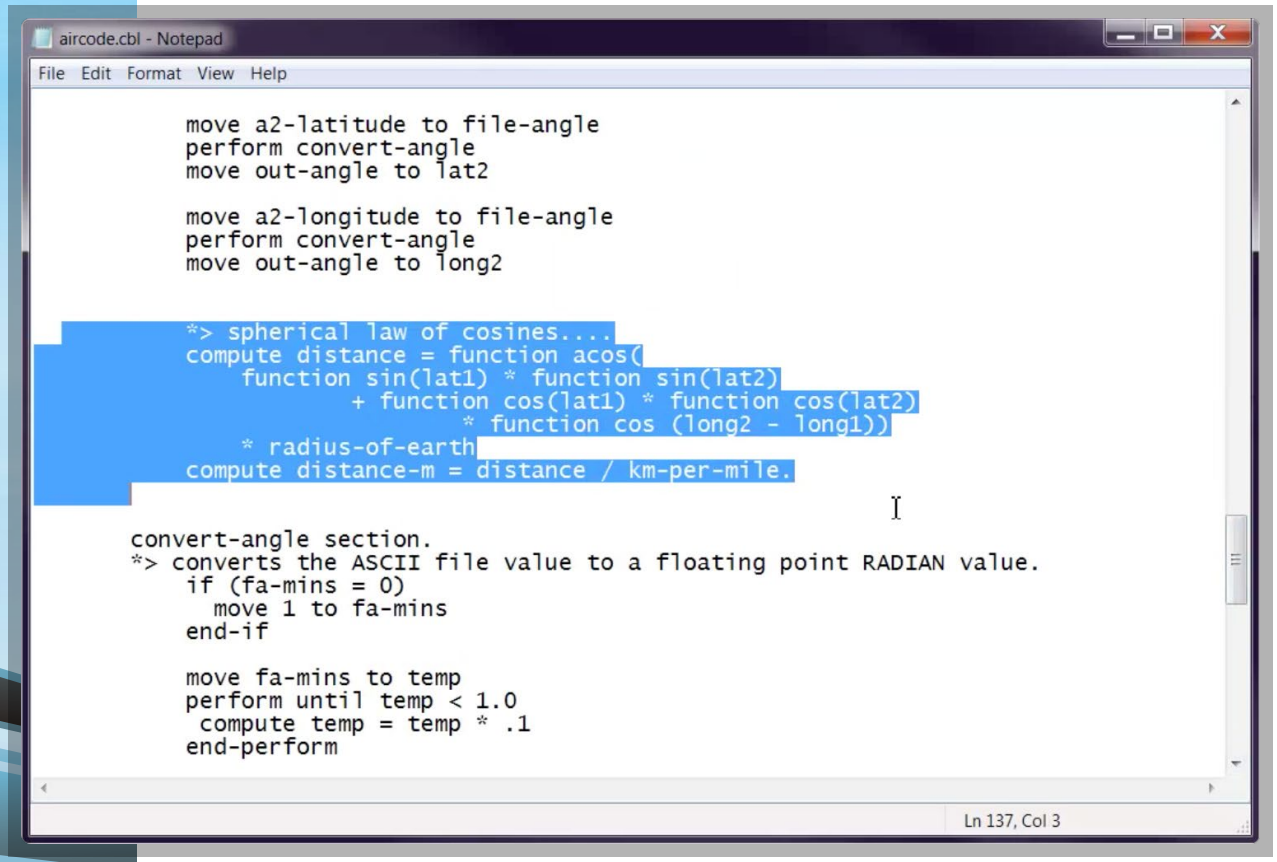
You may also be wondering how to keep data and data files separate from another user request. Visual COBOL actually provides call containers (we call them run units) which run your procedural COBOL programs in isolation from the same set of programs that may be supporting another user's request.

3. Identify the COBOL code



You need to identify the core COBOL business logic which represents the underlying functional capability you wish to make accessible to mobile user – then isolate this business logic from the rest of the main COBOL application.

3. Identify the COBOL code



This is a formula that calculates two points on the earth's surface. The data file used by the program contains a list of airports and their locations. So it can tell you how far apart one airport is from another.

4. Select the right tools for your architecture

Java or .NET based infrastructure

The next stage is thinking about the architecture of your application and the tools you'll need to do the job. First you need to decide whether to base the application on a Java or .NET based infrastructure (you may already have an in-house standard defined). Either way, Visual COBOL enables you to compile your COBOL API directly to Java byte code or Microsoft Intermediate Language so it can run directly inside either the Java Virtual Machine or the .NET CLR.

This means you can draw on tools in JVM and .NET to enhance your mobile application – here is what you'll need:

A web server or application server

The mobile client will be sending HTTP requests to the COBOL system and these need to be processed and returned as HTML web pages for the client's mobile browser. If you're a Java-based company and already have a Java App Server available, use the Java App Server, or consider an open source solution such as Tomcat. [Link to: http://tomcat.apache.org] If your architecture preference is .NET you'll need Microsoft IIS installed. If you don't have IIS installed, Visual Studio includes a limited version you can use to get started.

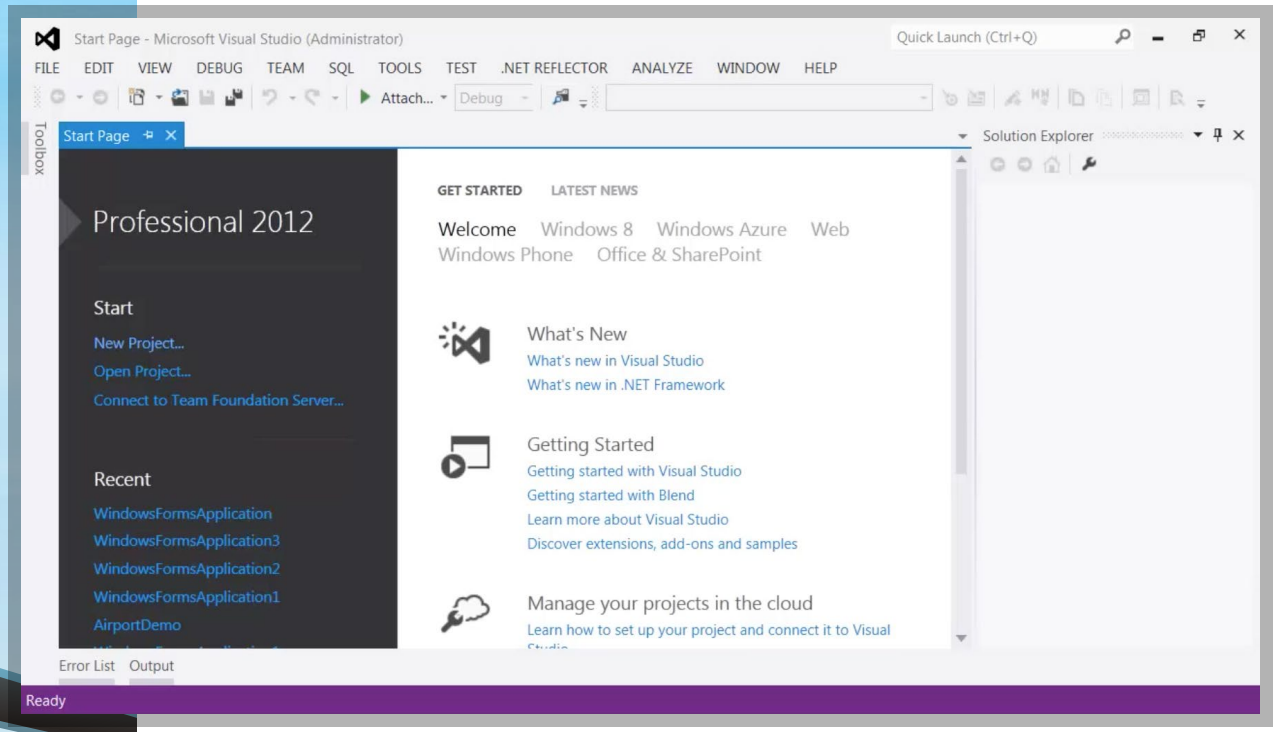
Portability

By choosing to build an HTML5 application, the challenge of portability moves from the device into the browser. While HTML5 offers good coverage, we still need to think about JavaScript for web programming. Fortunately, it's a problem the software industry has addressed – there are now plenty of portable JavaScript libraries you can use. In this example, we're utilizing a popular framework called jQuery. The jQuery Mobile extensions will provide us not only with a portable JavaScript library, but also a rich, touch-responsive user experience for the end users.

AJAX & JSON

The HTML5 application will need to communicate with a server-side COBOL service which we'll enable using web services. Web service requests are achieved using Asynchronous JavaScript and XML (or AJAX for short). The content of the request will be transmitted using JavaScript Object Notation (or JSON for short). APIs for both of these capabilities are available within the jQuery framework.

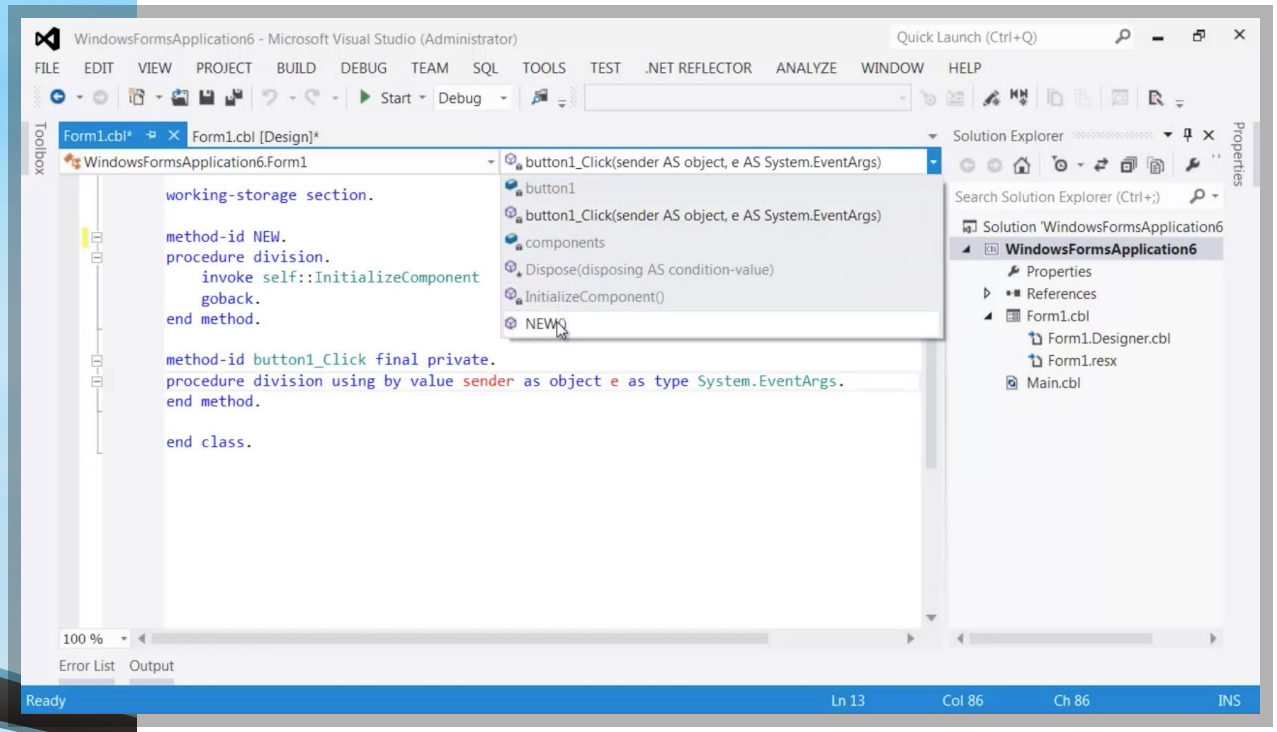
5. Fire up your IDE of choice



Now you can take advantage of the new features .NET and JVM offer. This is Microsoft's Visual Studio IDE which is used by millions of developers worldwide – and you can use it for COBOL development too. Alternatively, you can choose Eclipse as your IDE.

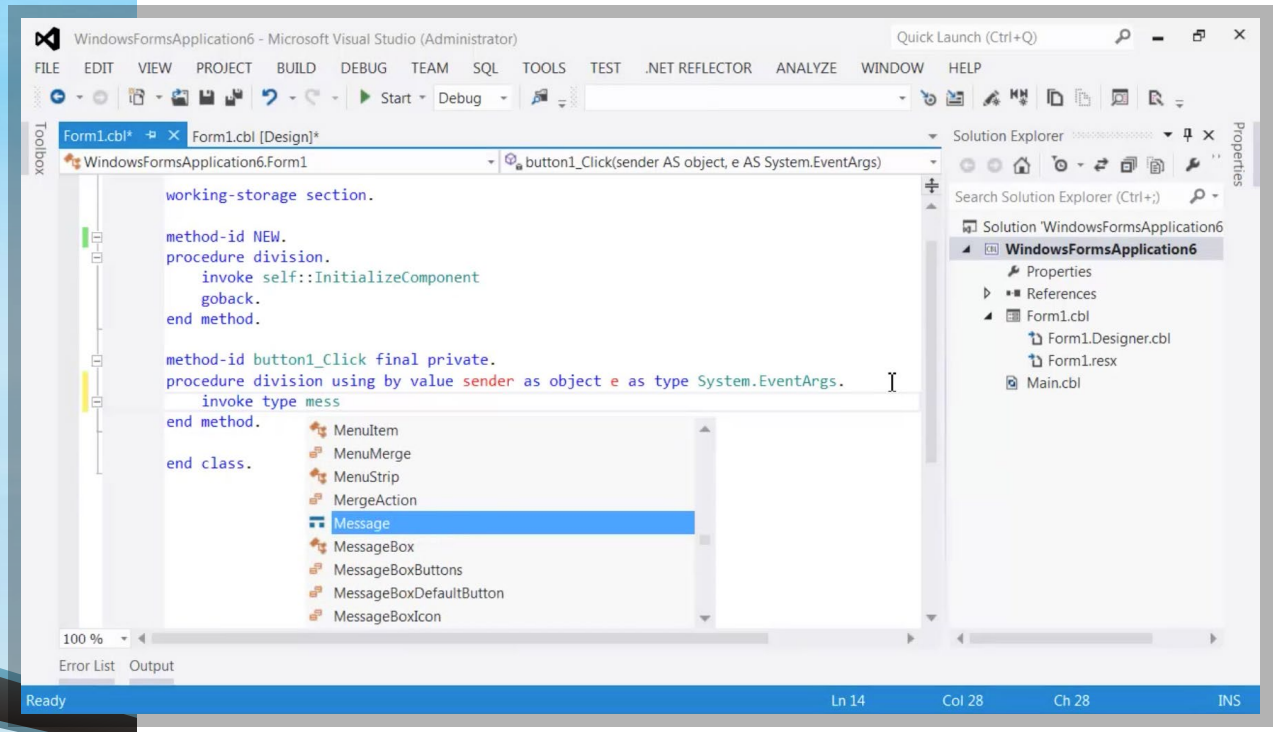
You can expect efficient development features like IntelliSense or Content Assist, error markers, fast navigation and a wealth of other tools customized for COBOL development – all inside what has become the industry standard for software development on Windows, Unix and Linux.

5. Fire up your IDE of choice



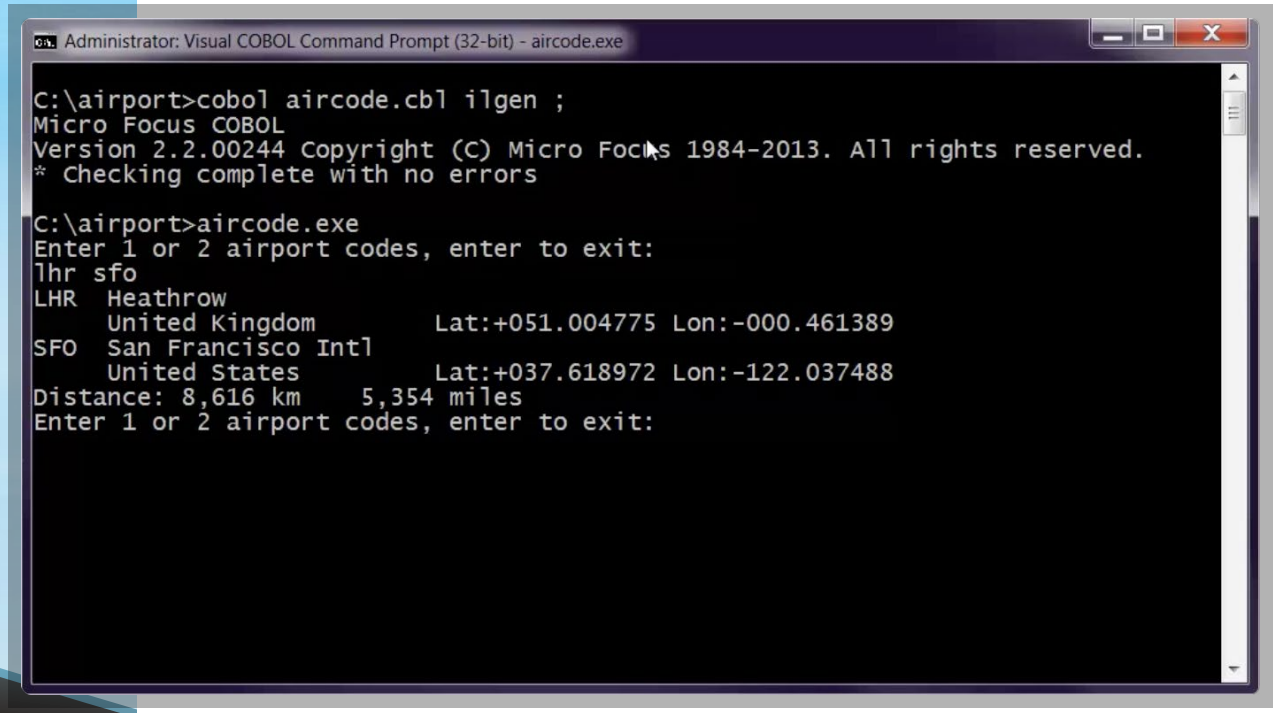
You can see the Editor is specifically designed for COBOL development, with syntax colorization, the ability to jump to locations in the code base using a mouse click, and dozens of other features which bring COBOL development into the modern age.

5. Fire up your IDE of choice



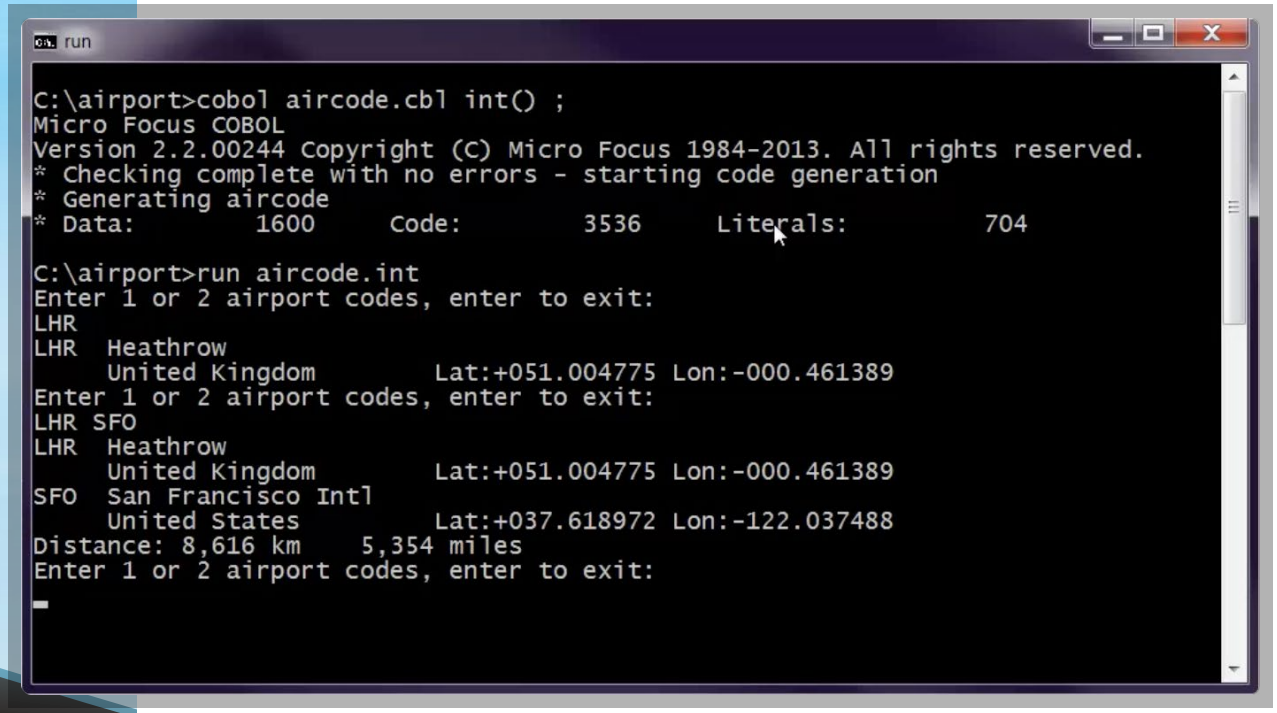
The IDE also prompts you with coding suggestions that help you write code faster and with greater accuracy.

6. Compile the code in your IDE



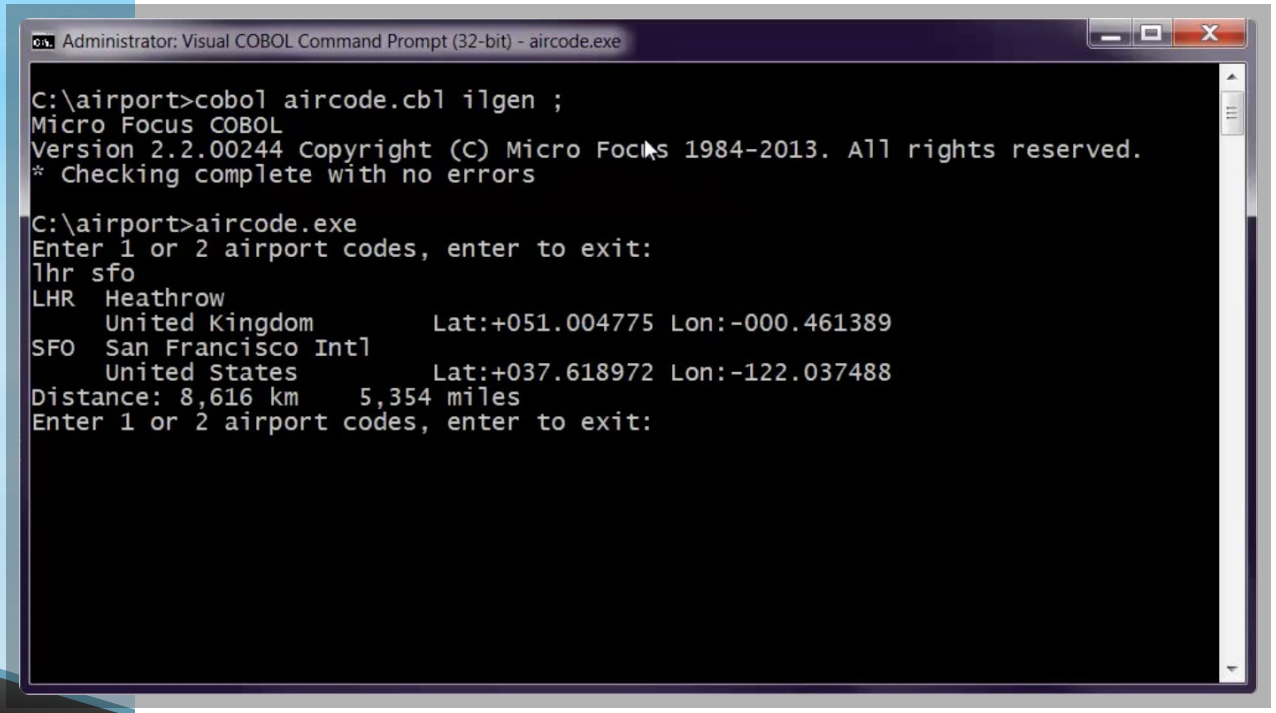
Using the COBOL application code you previously identified to provide the API, you can now create a project in the IDE, and compile your code for .NET using Visual Studio or JVM in Eclipse. In most cases, your procedural COBOL code can compile quite simply without any changes.

6. Compile the code in your IDE



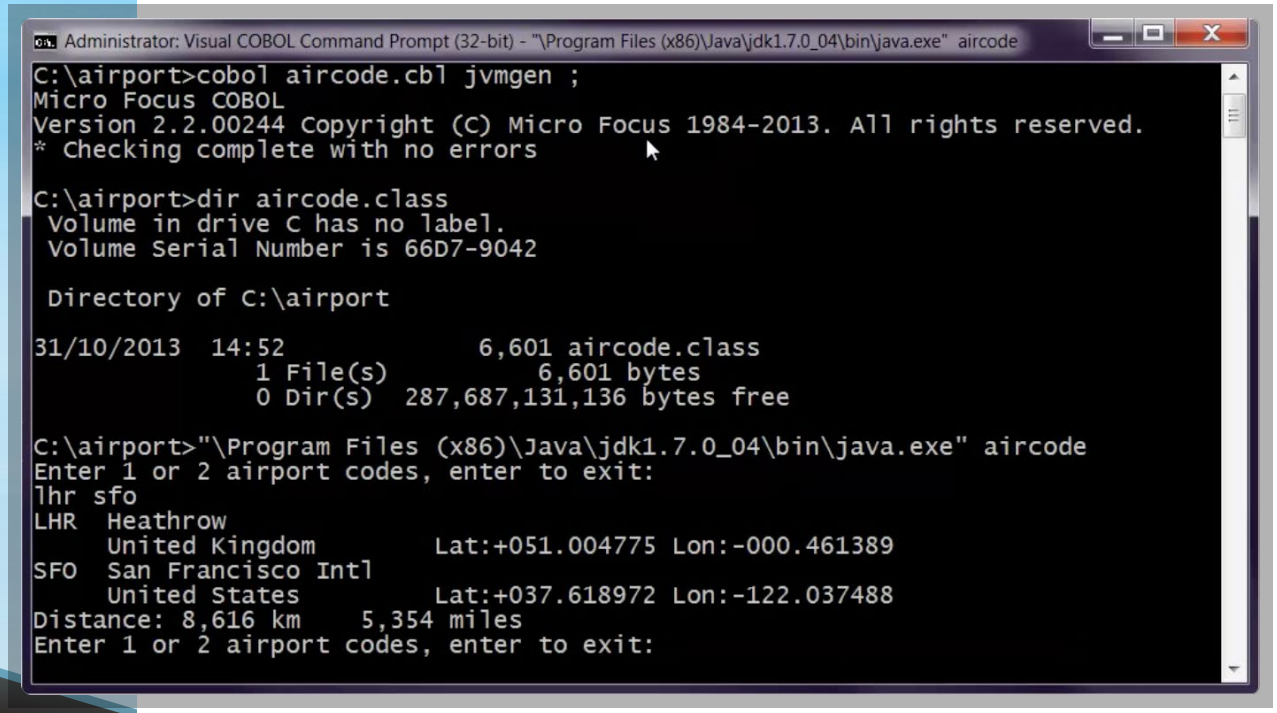
Our application uses a primitive character-based user interface, which is very common for COBOL systems. It will be revamped later in this process. By typing in LHR for Heathrow and SFO for San Francisco, the program calculates the distance between these two airports.

6. Compile the code in your IDE



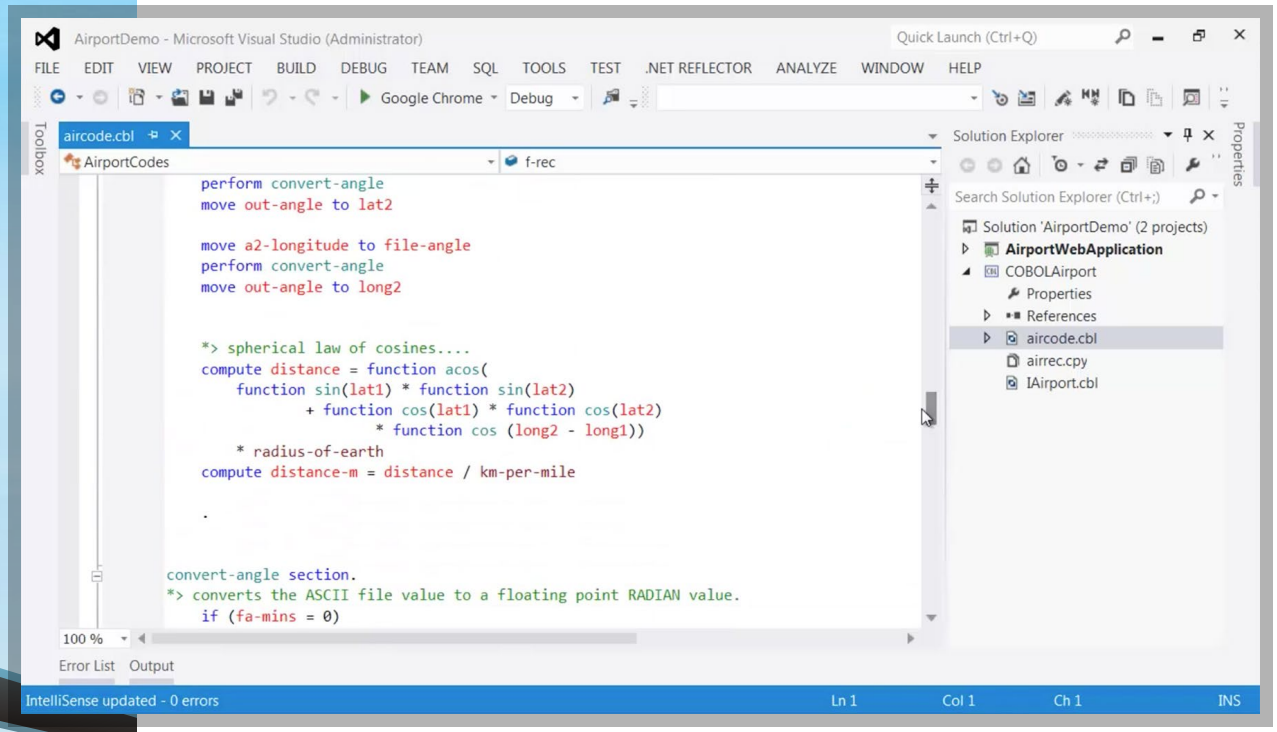
Now we want to tidy up the interface and access the application from a mobile device. To do that you would usually have to write a new program in C+ or VB for .NET or Java for JVM. But even though our program is written in COBOL, we can compile it for .NET just by changing a compiler directive to ilgen – which generates Microsoft Intermediate Language. So we now have a .NET assembly which we can run and check.

6. Compile the code in your IDE



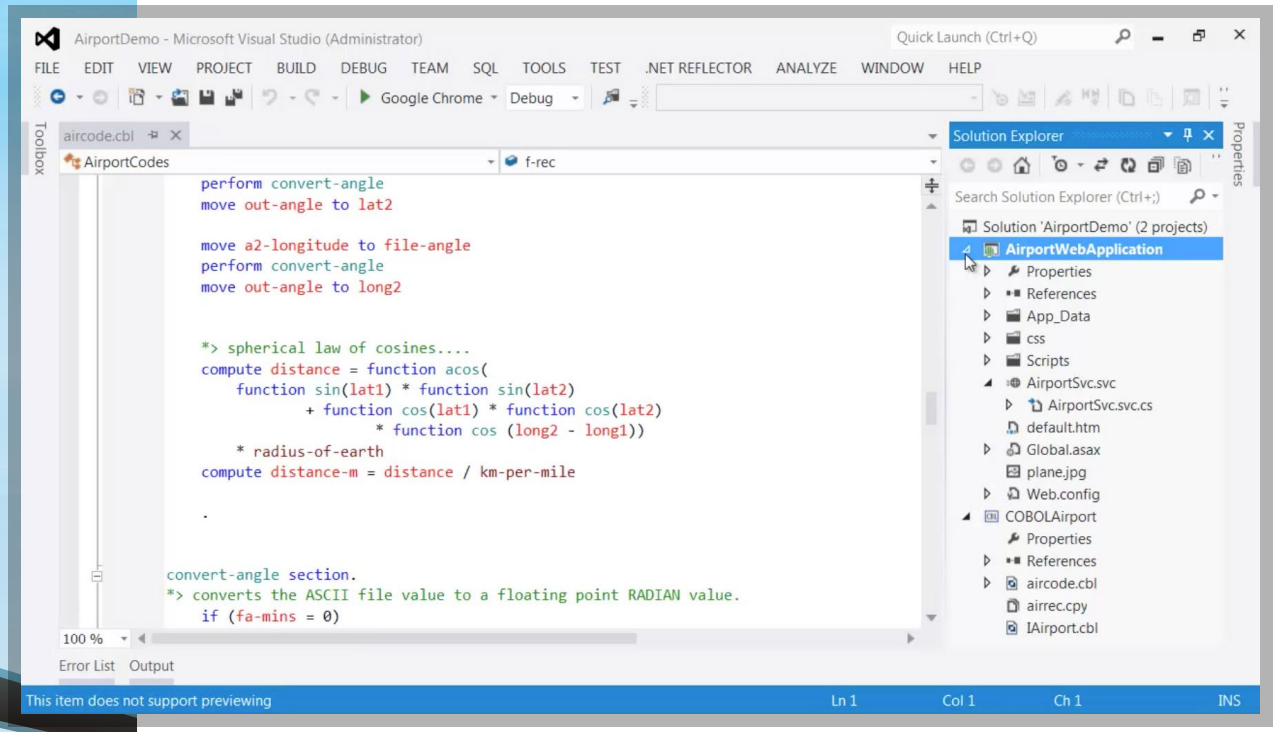
You can do the same for Java too, just by changing a compiler directive to jvmgen which generates a Java class which we can run using JVM. So Micro Focus COBOL compiler technology allows deployment to .NET or JVM platforms, without any rewrite to the original COBOL code.

7. Create the airport application user interface



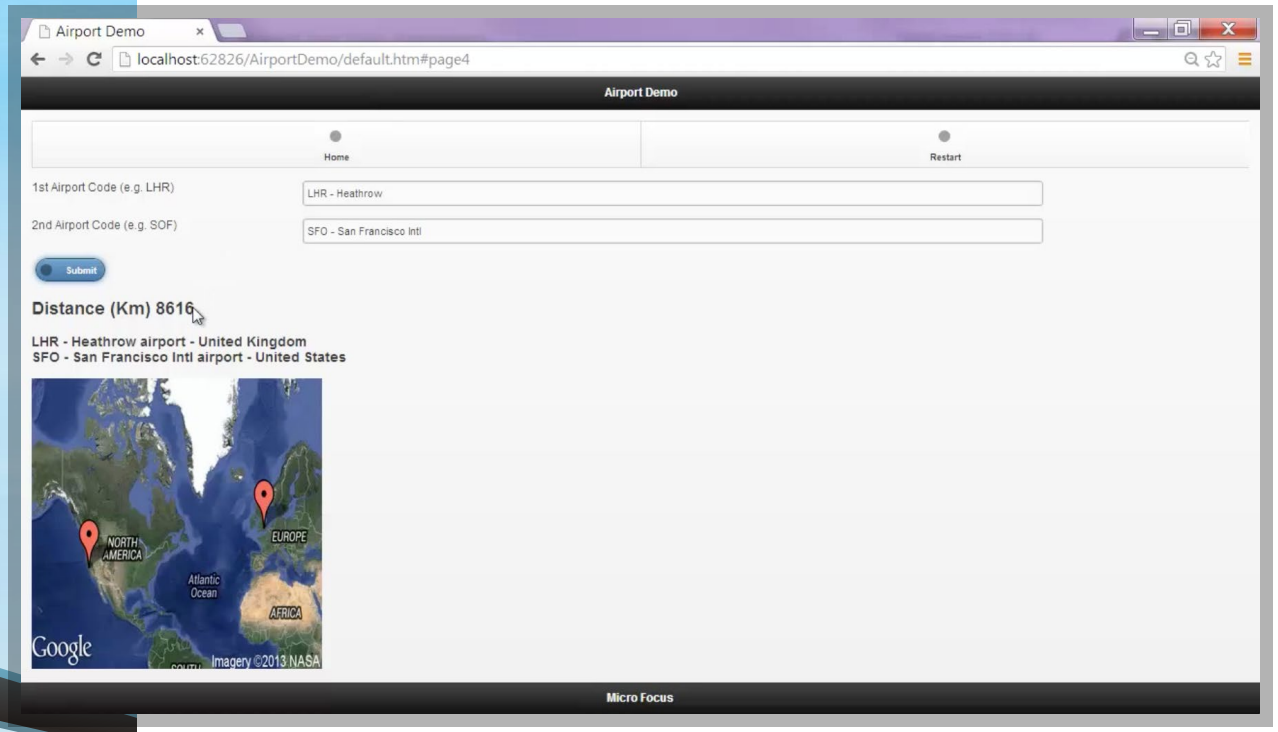
This project in Visual Studio includes our airport program. The user interface will be built using HTML5. Both Eclipse and Visual Studio provide HTML design tools to help you build your application UI. There are several alternatives to the standard designers available that allow you to build a UI based on a specific device form factor (the form factor of a mobile refers to its size, shape, input mechanism, screen resolution, colour palette and much more). The decision is yours as to which developer toolset you choose – once you have decided you need to export the HTML into your Eclipse or Visual Studio project.

7. Create the airport application user interface



This is another project in Visual Studio which contains a C# program and HTML5. This uses the COBOL program to provide all the backend functionality. The HTML 5 pages also use a JavaScript library (jQuery) to create a web app that runs in a browser and gives access to the application from mobile devices that support HTML5.

7. Create the airport application user interface



This is the completed interface with two new entry fields. As you type into them, the C# web service is being called by the browser to retrieve the list of matching airport codes. Since we know longitude and latitude, we can also provide a map of the two points.

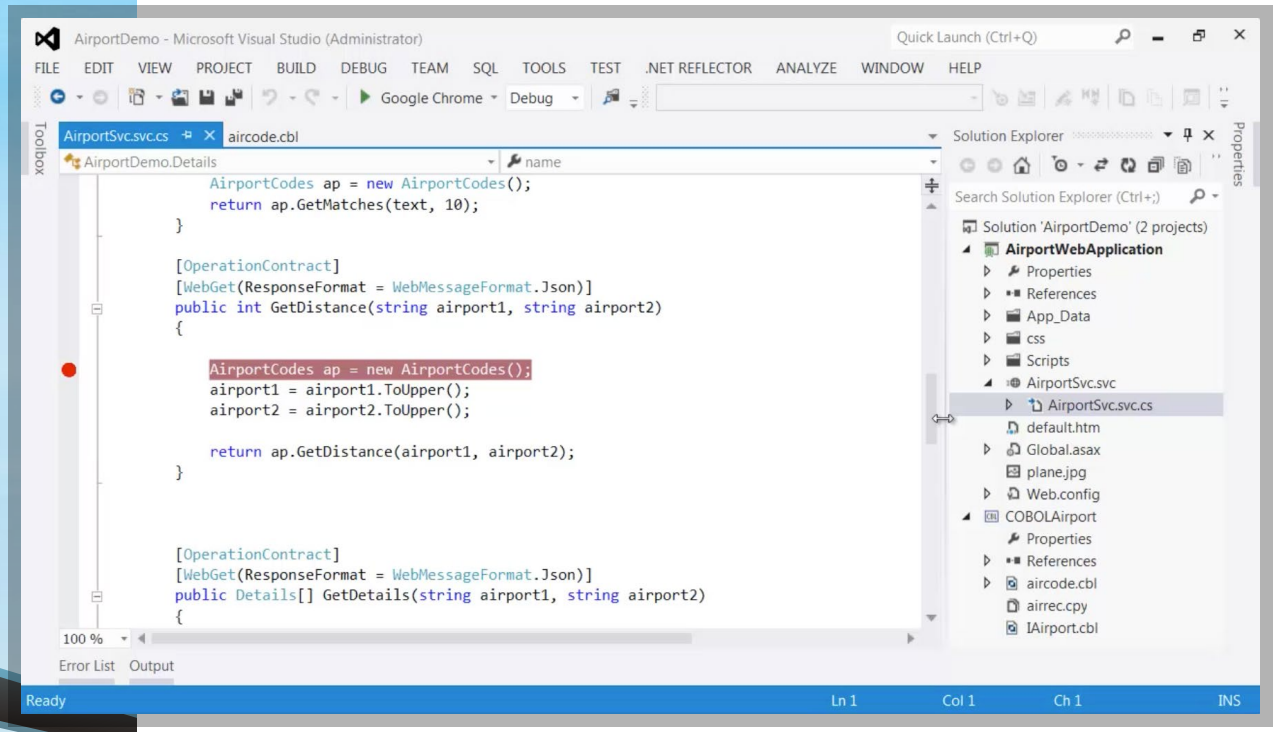
8. Create a simple COBOL web service and deploy

Create a simple COBOL web service

It's now time to build the web service itself. This is a thin layer of code that will receive an incoming request and invoke the COBOL program to perform the real business processing. You can use Java or a .NET language such as C# to implement the web service API. Calling the COBOL code from Java or C# is easy now that COBOL is compiled using the same basic building blocks as used in other languages. It's easy to integrate the two together – but you'll need to think about multi-user access.

COBOL programs are still procedural code, so we need to make sure that they can handle simultaneous access by many users at once. Fortunately, there's no need to re-architect your code, you can use the run unit container support within Visual COBOL to isolate one user web service call from another.

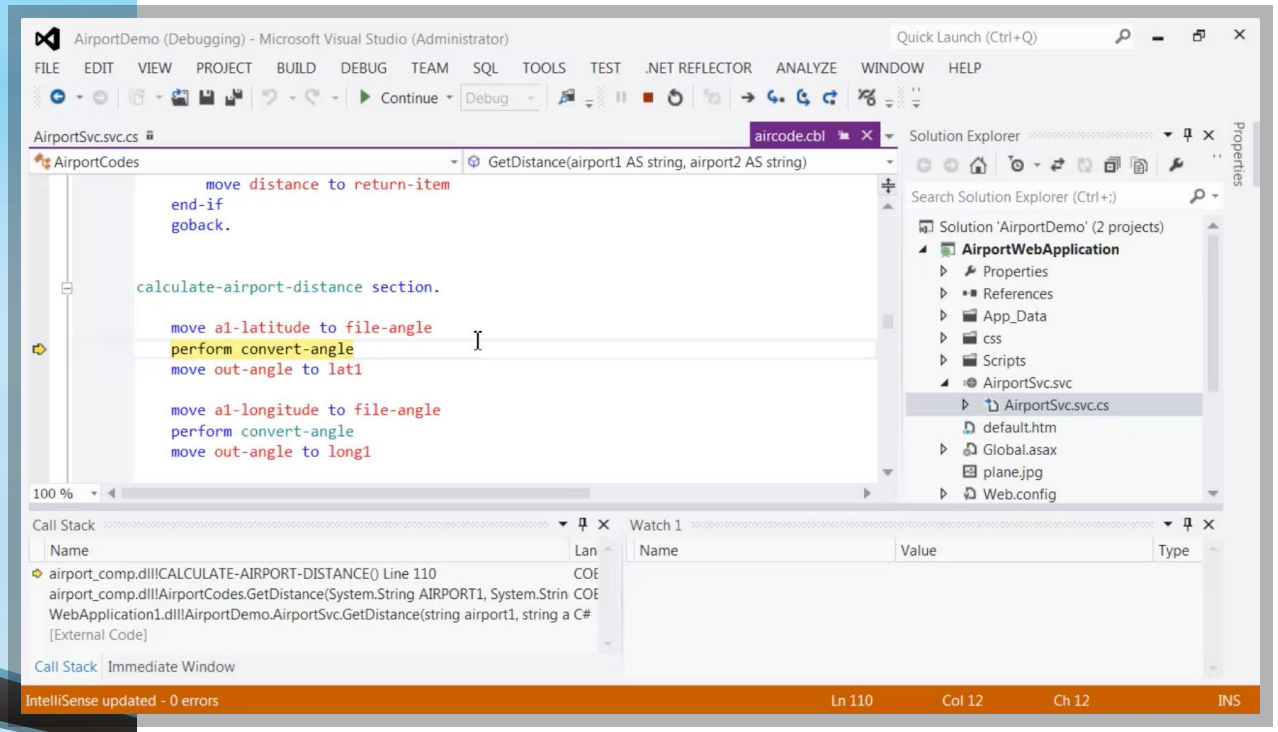
8. Create a simple COBOL web service and deploy



The C# code provides a simple web service that allows the browser running on the device to dynamically update the page with data from the COBOL program running on the web server.

To deploy, you connect the user interface with the web service using AJAX calls. Whenever the user clicks a button or types a character into an input field, AJAX calls are being sent from the client to the server-side COBOL program to populate the UI with results received from the COBOL application.

9. Test the mobile app



When it's time to debug the application, a breakpoint can be set in the IDE and you can step through the code from C# into the backend COBOL program.

WHAT NEXT?

Watch the video showing how we built the app

Get started! **Take a free Visual COBOL trial** and download the application code