Walkthrough: Executing Code in the Interactive Window

With the new **C# Interactive window**, you get immediate feedback on what an expression will return or what an API call does. The Interactive window is much like the Immediate window, but the Interactive window has many improvements such as intellisense features and the ability to redefine functions and classes. After entering a code snippet at the REPL prompt, the code simply executes right away. You can enter statements and expressions, as well as class and function definitions. You do not need to create a project, define a class, define a Main, write your expression in a Console.WriteLine() call, and then enter the code for a bogus Console.ReadLine to keep the cmd.exe window alive. When you type each line of code that is a complete expression or construct and then press Enter, the code executes. If the entered code is incomplete when you press Enter, it does not execute, and you can continue entering code.

The Interactive window is a read-eval-print loop (or REPL), <http://en.wikipedia.org/wiki/REPL> . As you can see, REPLs are productivity enhancers that have long been the purview of dynamic and functional languages. Some (non-functional) static languages have had REPL offerings, such as SuperCede 2.0 for Java or Mono's C# REPL. The Roslyn REPL for C# brings all the goodness you associate with VS such as completion, code coloring, Quick Fixes, etc., to the REPL experience.

To use this walkthrough, you must first install Visual Studio 2012, and then install the Roslyn Community Technology Preview (CTP). The CTP is at <http://go.microsoft.com/fwlink/?LinkId=228400> .

This walkthrough demonstrates the following features or attributes of the C# Interactive window:

* Launching REPL with default execution context
* Using directives
* History commands
* Enter to execute if input is complete
* Ctrl-Enter to force execution of current input, or fetch previous input if the caret is in a previous submission
* Shift-Enter to insert a newline without executing the current input
* Var declarations
* Colorization, completion, param tips
* Quick fixes
* Expression evaluation
* Multi-line input
* Multi-line history with editing
* Shows redirecting Console I/O to REPL

**Walkthrough Steps …**

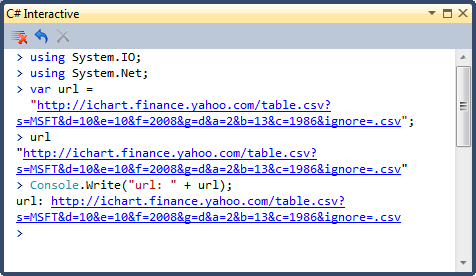
1. Open Visual Studio with the Roslyn CTP installed.
2. On the **View** menu, choose **Other Windows** and then **C# Interactive**. It is best to drag this to the document bay and dock the interactive window at the bottom of the document bay. It is common to work in an editor buffer and the REPL, switching back and forth for entering and saving code snippets, particularly if you are writing a script.

You may notice the C# Interactive window's toolbar has command icons for clearing the window's contents and resetting the execution context; there is no button to interrupt the currently executing code submission at this time. If you enter an infinite loop while playing around, you can use the reset button to reset the REPL's execution context. You shouldn't need this during the walkthrough, but if you do use it, you will need to restart the walkthrough.

After the banner for the execution environment, the REPL prompts you with "> ", which is where you provide input submissions for execution.

1. To honor our ancestry, let's enter the obligatory Hello World program in the REPL. Type the following, and press Enter.  
     
   > Console.Write("Hello, World!");
2. There are commands in the REPL that look like directives. The #help command describes common commands (input starting with #) and shortcuts for getting started:

> #help

1. Type the following input, and then press Enter. Notice that IntelliSense completion helps while you type.  
     
   > using System.IO;  
     
   Then input the following command, which you can do by using alt-uparrow to invoke history, backspace to delete the semicolon, ctrl-backspace to delete "io", then type "net;", and press Enter.  
     
   > using System.Net;
2. Enter the following partial statement without a semicolon at the end, and press Enter.  
     
   > var url =  
     
   Because the statement is not yet complete, the REPL does not execute it, and the code continues to the next line. Paste in the following code, and then press Enter. This code is a continuation of the previous line.   
     
   "http://ichart.finance.yahoo.com/table.csv?s=MSFT&d=10&e=10&f=2008&g=d&a=2&b=13&c=1986&ignore=.csv";  
     
   Because the line is complete, pressing enter at the end of the input executes the code.
3. Now view the contents of the variable by entering it as an expression. Type just the variable name without a semicolon at the end (note, in the current CTP, you may need to press Enter twice, once to dismiss the completion list and once to confirm the input).   
     
   > url  
     
   The value of the url variable shows on the next line.
4. Here's another way to view the results of an expression evaluation. Type the following, and press Enter.  
     
   > Console.Write("url: " + url);  
     
   Here's the Interactive window at this point:  
     
   
5. Enter the following lines into the Interactive window. The code uses a WebRequest instance to download data from the web site, and puts the result in the CSV string variable. For more information and a similar example, see [WebRequest Class](http://msdn.microsoft.com/en-us/library/system.net.webrequest.aspx).  
     
   > var request = WebRequest.Create(url);  
   > var response = request.GetResponse();  
   > var dataStream = response.GetResponseStream();  
   > var reader = new StreamReader(dataStream);  
   > var csv = reader.ReadToEnd();  
   > reader.Close();  
   > dataStream.Close();  
   > response.Close();
6. Now that we have some data we pulled, let's inspect the data briefly before moving forward. Enter the following line without a semicolon at the end.   
     
   > csv.Length  
     
   The REPL displays the length of the CSV string on the next line.
7. Ok, there's a lot of data, so maybe we can break it up. Let's see how many lines there are:

> csv.Split('\n').Length

1. Still a lot of lines, so let's peak at the first couple of hundred characters and see if we can glean something of the string's structure or how long the lines are. Enter the following code.

> Console.Write(csv.Substring(0, 200));

Notice we reformat on as you type the close curly. We do not adhere to formatting options in CTP1, so that is a known issue.

We define 'count' outside of the loop, anticipating needing it again and then possibly saving some typing later. You'll see output similar to the following:

Date,Open,High,Low,Close,Volume,Adj Close

2010-11-10,27.01,27.08,26.81,26.94,52277300,26.28

2010-11-09,26.81,27.11,26.71,26.95,58538600,26.29

2010-11-08,26.68,28.87,26.58,26.81,71670800,26.15

2010-11-

1. Now we can see what the structure of the data is. Let's build a query to extract the adjusted closing price from the last column (Skip(1) skips header row). You can use Shift-Enter at the end of a line to avoid executing the input until you’ve entered everything; enter only evaluates if the expression looks complete:

> var prices = csv.Split('\n').Skip(1)

.Select(line => line.Split(','))

.Where(values => values.Length == 7)

.Select(values => new { date = DateTime.Parse(values[0]),

                                       price = float.Parse(values[6]) });

1. Let's print out a bit of the prices from the query. You can use Shift-Enter after the first to lines to avoid executing them immediately. If you use Enter inside the 'foreach' loop, the code won't execute until you type the final curly brace and then press Enter.

> foreach (var p in prices.Take(10))

      Console.WriteLine(p);

You're done. Enjoy using the REPL and please provide feedback!

**Walkthrough Log …**

Microsoft (R) Roslyn C# Compiler version 1.0.0.0

Loading context from 'CSharpInteractive.rsp'.

Type "#help" for more information.

> #prompt inline "> " " "

> using System.IO;

> using System.Net;

> var url =

"http://ichart.finance.yahoo.com/table.csv?s=MSFT&d=10&e=10&f=2008&g=d&a=2&b=13&c=1986&ignore=.csv";

> url

"http://ichart.finance.yahoo.com/table.csv?s=MSFT&d=10&e=10&f=2008&g=d&a=2&b=13&c=1986&ignore=.csv"

> Console.Write("url: " + url);

url: http://ichart.finance.yahoo.com/table.csv?s=MSFT&d=10&e=10&f=2008&g=d&a=2&b=13&c=1986&ignore=.csv

> var request = WebRequest.Create(url);

var response = request.GetResponse();

var dataStream = response.GetResponseStream();

var reader = new StreamReader(dataStream);

var csv = reader.ReadToEnd();

reader.Close();

dataStream.Close();

response.Close();

> csv.Length

287803

> csv.Split('\n').Length

5720

> Console.Write(csv.Substring(0, 200));

Date,Open,High,Low,Close,Volume,Adj Close

2008-11-10,21.85,21.97,21.19,21.30,67106800,19.87

2008-11-07,21.32,21.54,21.00,21.50,71256300,20.06

2008-11-06,21.87,22.08,20.86,20.88,95509700,19.48

2008-11-

> var prices = csv.Split('\n').Skip(1)

.Select(line => line.Split(','))

.Where(values => values.Length == 7)

.Select(values => new { date = DateTime.Parse(values[0]),

                                       price = float.Parse(values[6]) });

> foreach (var p in prices.Take(10))

  Console.WriteLine(p);

{ date = 11/10/2008 12:00:00 AM, price = 19.87 }

{ date = 11/7/2008 12:00:00 AM, price = 20.06 }

{ date = 11/6/2008 12:00:00 AM, price = 19.48 }

{ date = 11/5/2008 12:00:00 AM, price = 20.05 }

{ date = 11/4/2008 12:00:00 AM, price = 21.37 }

{ date = 11/3/2008 12:00:00 AM, price = 20.54 }

{ date = 10/31/2008 12:00:00 AM, price = 20.28 }

{ date = 10/30/2008 12:00:00 AM, price = 20.55 }

{ date = 10/29/2008 12:00:00 AM, price = 20.88 }

{ date = 10/28/2008 12:00:00 AM, price = 20.98 }

>