Fiddler
Web Debugging Proxy

Extending Fiddler with .NET Code

Introduction

Fiddler2 is a highly versatile platform that offers extensibility via both script and .NET code. Using the extensibility mechanisms, you can add to Fiddler's UI, automatically modify requests or responses, and custom Inspectors that enable scenario-specific display and manual-modification of requests and responses.

Prerequisites

Writing extensions for Fiddler2 requires Visual Studio .NET 2005+ or the free .NET Framework v2 command-line compilers.

Fiddler v2.x loads only **.NET CLR v2.0** assemblies; use Visual Studio 2005+ to compile your extension. Fiddler itself requires only that the user have **.NET Framework 2.0 SP1** installed. You *may* have your extensions target the .NET Framework 3.5 (which includes Linq) since that framework also (confusingly) runs on the v2.0 CLR but you MUST yourself ensure that the user has the required Framework version installed BEFORE you install your extension, or a user with only the older Framework will crash on boot.

You should also ensure your project targets **AnyCPU** to ensure that it works properly with 64bit Fiddler.

You will also need to have the most recent version of Fiddler2 installed in order to develop Fiddler extensions.

If you've previously developed Fiddler extensions, you may want to read the article Updating Fiddler Extensions for v2.1.

Debugging your Extensions

- Extension developers should set the `fiddler.debug.extensions.showerrors` preference to **True** ensure that exceptions and other extension-related errors are not silently caught.
- Extension developers should set the `fiddler.debug.extensions.verbose` preference to **True** to spew logging information to the Log tab.

Direct Fiddler2 to load your extension assemblies

Fiddler loads extension assembly DLLs from the `%Program Files%\Fiddler2\Scripts` and `%USERPROFILE%\My Documents\Fiddler2\Scripts` folders. Install to the %Program Files% location to make your extensions available to all users on the machine, or the %UserProfile% folder to make the extension available only to the current user.

In addition to placing your extension DLLs in the appropriate folder, you must also mark your assembly to indicate the minimum version of Fiddler required for your Assembly to load correctly.

Set the `Fiddler.RequiredVersion` attribute in your `AssemblyInfo.cs` file (or elsewhere in your code), as follows:

```
using Fiddler;

// Extension requires Fiddler 2.2.8.6+ because it uses types introduced in v2.2.8...
[assembly: Fiddler.RequiredVersion("2.2.8.6")]
```

If Fiddler finds a `RequiredVersion` attribute that indicates a later version of Fiddler is required, the user will be notified that a later version of Fiddler is required to use your extension. Assemblies which do not contain a `RequiredVersion` attribute are silently ignored.

The IFiddlerExtension Interface

Public classes in your assembly that implement the `IFiddlerExtension` interface will be loaded by Fiddler during startup.

```
public interface IFiddlerExtension
{
  // Called when Fiddler User Interface is fully available
  void OnLoad();

  // Called when Fiddler is shutting down
```

```
  // Called when Fiddler is shutting down
  void OnBeforeUnload();
}
```

The OnLoad function will be called when Fiddler has finished loading and its UI is fully available. At this point, you can safely add menu items, tabbed pages, or other elements to the Fiddler UI.

The OnBeforeUnload function will be called when Fiddler is shutting down and unloading all extensions.

The IAutoTamper Interface (extends IFiddlerExtension)

Extensions that implement the IAutoTamper interface (which descends from the IFiddlerExtension interface) are called for each HTTP/HTTPS request and response, enabling modifications, logging, or other operations.

WARNING: Functions in this interface are called on background, non-UI threads. If you wish to update UI, you **must** use **Invoke** or **BeginInvoke** to update the UI. Also, note that the IAutoTamper:* functions may be called *before* the **OnLoad** event is called-- Fiddler allows traffic to flow *before* the UI is fully available.

```
public interface IAutoTamper : IFiddlerExtension
{
  // Called before the user can edit a request using the Fiddler Inspectors
  void AutoTamperRequestBefore(Session oSession);

  // Called after the user has had the chance to edit the request using the Fiddler Inspectors, but before
the request is sent
  void AutoTamperRequestAfter(Session oSession);

  // Called before the user can edit a response using the Fiddler Inspectors, unless streaming.
  void AutoTamperResponseBefore(Session oSession);

  // Called after the user edited a response using the Fiddler Inspectors.  Not called when streaming.
  void AutoTamperResponseAfter(Session oSession);

  // Called Fiddler returns a self-generated HTTP error (for instance DNS lookup failed, etc)
  void OnBeforeReturningError(Session oSession);
}
```

The IAutoTamper2 Interface (Extends IAutoTamper)

```
 /// <summary>
/// Interface for AutoTamper extensions that want to "peek" at response headers
/// </summary>
public interface IAutoTamper2 : IAutoTamper
{
/// <summary>
/// Called when the response headers become available
/// </summary>
/// <param name="oSession">The Session object for which the response headers are available</param>
void OnPeekAtResponseHeaders(Session oSession);
}
```

The IAutoTamper3 Interface (Extends IAutoTamper2)

```
/// <summary>
/// Interface for AutoTamper extensions that want to "peek" at request headers
/// </summary>
public interface IAutoTamper3 : IAutoTamper2
{
/// <summary>
/// Called when the request headers become available
/// </summary>
/// <param name="oSession">The Session object for which the request headers are available</param>
void OnPeekAtRequestHeaders(Session oSession);
}
```

The IHandleExecAction Interface

Extensions that implement the IHandleExecAction interface are called when the user has entered a command into the QuickExec box. If your extension would like to react to the command (and prevent further processing by other extensions and Fiddler itself) return **true** from this method.

```
public interface IHandleExecAction
{
  // return TRUE if handled.
  bool OnExecAction(string sCommand);
}
```

Note that the Fiddler.Utilities class includes a helper function which you may find useful when interpreting the sCommand parameter.

```
[CodeDescription("Tokenize a string into tokens. Delimits on whitespace; Quotation marks are dropped unless
preceded by a \ character.")]
public static string[] Parameterize(string sCommand)
```

Step-by-step Sample Extension

Here's a trivial extension which modifies the User-Agent string of all outbound requests.

1. Start Visual Studio 2005 or later.
2. Create a new Project of type **Visual C# Class Library**
3. Right-click the project's **References** folder in the **Solution Explorer**
4. Choose the **Browse** tab and find **Fiddler.exe** in the **C:\Program Files\Fiddler2** folder.
5. Click **Ok** to add the reference.
6. If you're planning to add to Fiddler's UI
    1. Right-click the project's **References** folder in the **Solution Explorer** again
    2. On the **.NET** tab, choose **System.Windows.Forms**.
    3. Click **Ok** to add the reference.
7. In the **Solution Explorer**, right click the project. Choose **Properties**.
8. On the **Build Events** tab, add the following to the **Post-build event command line**:

```
    copy "$(TargetPath)" "%userprofile%\My
Documents\Fiddler2\Scripts\$(TargetFilename)"
```

9. Modify the default class1.cs (or create a new class) in your project as follows:

```
using System;
using System.Windows.Forms;
using Fiddler;

[assembly: Fiddler.RequiredVersion("2.3.5.0")]

public class Violin : IAutoTamper    // Ensure class is public, or Fiddler won't see
it!
{
  string sUserAgent = "";

  public Violin(){
  /* NOTE: It's possible that Fiddler UI isn't fully loaded yet, so don't add any UI
in the constructor.

     But it's also possible that AutoTamper* methods are called before OnLoad (below),
so be
     sure any needed data structures are initialized to safe values here in this
constructor */

     sUserAgent = "Violin";
  }

  public void OnLoad(){ /* Load your UI here */ }
  public void OnBeforeUnload() { }

  public void AutoTamperRequestBefore(Session oSession){
    oSession.oRequest["User-Agent"] = sUserAgent;
  }
  public void AutoTamperRequestAfter(Session oSession){}
  public void AutoTamperResponseBefore(Session oSession){}
  public void AutoTamperResponseAfter(Session oSession){}
  public void OnBeforeReturningError(Session oSession){}
}
```

Adding an Icon to your Extension's tab

The icons are chosen from the ImageList named **imglSessionIcons** attached to **FiddlerApplication.UI**.

In order to use an existing icon, you can simply set the .ImageIndex property as follows:

```
public void OnLoad()
{
oPage = new TabPage("Timeline");
oPage.ImageIndex = (int)Fiddler.SessionIcons.Timeline;
oView = new TimelineView();
oPage.Controls.Add(oView);
oView.Dock = DockStyle.Fill;
FiddlerApplication.UI.tabsViews.TabPages.Add(oPage);
}
```

If you want to add your own custom image, you'll first have to add it to **imglSessionIcons.**

This member was non-public in older versions of Fiddler, so you will get a visibility exception in older versions if you attempt to manipulate imglSessionIcons. To deal with this you could either:

1> Just reuse an existing icon via the ImageIndex property, or

2> Use the [assembly: Fiddler.RequiredVersion("2.3.5.0")] attribute to require that users upgrade to a current version of Fiddler.

Other Examples and Information

- The AnyWhere sample shows how to use IAutoTamper to change HTTP responses.
- The RulesTab IFiddlerExtension sample is a simple example of using the IFiddlerExtension interface to add a new tab to Fiddler's UI.
- The Content Blocker sample is a simple example of using IAutoTamper to block traffic based on URI.
- The Fiddler Architecture document provides useful information for extension builders.

Almost done...

- Compile your project.
- Drop your assembly .DLL in your **\My Documents\Fiddler2\Scripts** folder (or use **\Program Files\Fiddler2\Scripts** to make available to all users on the machine)
- Restart Fiddler2.

If you need help or have questions, please email me using the **Contact** link above.

Change Log
6/28/07 - Complete rewrite for new interfaces
8/20/08 - Minor clarifications
11/19/09 - Added info on x64
2/23/10 - Fix typos
8/8/11 - Add info on IAutoTamper2 and IAutoTamper3