Debugging with Fiddler

The complete reference from the creator of the Fiddler Web Debugger

This is a SAMPLE containing the **Table of Contents** and a bit of content so you can decide whether the book meets your needs and renders nicely on your device.

Buy the book at `http://www.fiddlerbook.com`

Second Edition

Eric Lawrence

Debugging with Fiddler (2nd Edition)
Cover Photo: Nicholas Wave (@by_nicholas) ©iStockphoto.com

Book Version LULU 2.00 / Fiddler Version 2.5.0.0
Published March 5, 2015 – Austin, TX

e_lawrence@hotmail.com
@ericlaw on Twitter

Acknowledgements

This book, and Fiddler itself, would not have been possible without myriad contributions from hundreds of people around the world.

First, I'd like to thank my wife and son for their inspiration and encouragement as I spent innumerable nights and weekends working on Fiddler and authoring this book. Next, thanks to my parents and grandmother, who instilled in me a voracious appetite for books and the idea that one day I should try my hand at writing one.

I'm grateful for the many contributions of colleagues too numerous to mention (they know who they are!), and to the broader Fiddler community for providing a steady stream of encouragement, suggestions and bug reports. I'd like to thank my employer, Telerik, who acquired Fiddler in 2012 and generously continues to fund my work on the platform.

Finally, I thank you, dear reader, for caring enough about Fiddler to pick up this book!

TABLE OF CONTENTS

About this book

Over twelve years and well over a hundred version releases, Fiddler has evolved into a powerful utility and platform that can perform a wide variety of tasks. It has a rich extensibility model and a community of add-on developers who have broadened its usefulness as a performance, security, and load-testing tool. Questions in email, online discussion groups, and numerous conferences over the years made it overwhelmingly apparent that most users only exploit a tiny fraction of Fiddler's power. I came to realize that thousands of users would get a lot more out of Fiddler if there were a complete reference to the tool available. The first version of this book, released in 2012, was the product of that realization. This Second Edition, released in early 2015, builds upon the first and adds new content covering major enhancements to Fiddler since the book was originally published.

As Fiddler's developer, I've found it both easy and challenging to write this book. It's easy, because I understand Fiddler deeply, down to its very foundation, and can consult the source code to research obscure details. On the other hand, it's been very challenging, as every time I choose an interesting scenario or feature to write about, I'm forced to think deeply about that scenario or feature. Commonly, I've found myself developing improvements to revise Fiddler and minimize or eliminate the need to write about the topic in the first place. As a result, I've rewritten large portions of both this book and Fiddler itself. It's been a slow process, but both projects have benefitted.

Publication of this edition roughly coincided with the release of Fiddler version 2.5 in the winter of 2015. If you're using a later version of Fiddler, you will find some minor differences, but the core concepts will remain the same.

This book is deliberately limited in scope—it covers nearly every aspect of Fiddler and FiddlerCore, but it is not a tutorial on HTTP, SSL/TLS, HTML, Web Services or the myriad other topics you may want to understand to fully exploit Fiddler's feature set. If you want a deeper understanding of web protocols, I can recommend the references that I consulted during the development of Fiddler:

- Hypertext Transfer Protocol -- HTTP/1.1 from http://www.ietf.org/rfc/rfc2616.txt, later obsoleted by the HTTPbis Working Group's release of RFC7230 through RFC7235.
- HTTP: The Definitive Guide by David Gourley
- Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement by Balachander Krishnamurthy and Jennifer Rexford
- SSL & TLS Essentials: Securing the Web by Stephen A. Thomas
- Bulletproof SSL and TLS by Ivan Ristić

This book can be read either "straight through" or you can use the Table of Contents and Index to find the topics most interesting to you. Please consider skimming all of the chapters, even those that don't seem relevant to your needs, because each chapter often contains tips and tricks you might not find elsewhere.

I encourage you to begin by reading the primer in the next chapter, which lays out some terminology and the basic concepts that you'll need to understand to get the most out of Fiddler and this book. Enjoy!

Icons and Colors

The default text coloring of each row in the Web Sessions list derives from the HTTP Status (red for errors, yellow for authentication demands), traffic type (`CONNECT`s appear in grey), or response type (CSS in purple, HTML in blue; script in green, images in grey). You can override the font color by setting the Session's `ui-color` flag from FiddlerScript.

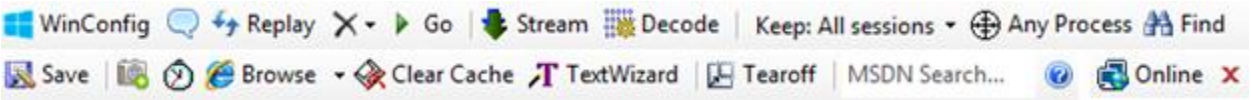
Each row is also marked with an icon for quick reference as to the Session's progress, Request type, or Response type:

	Request is being sent to the server.
	Response is being downloaded from the server.
	Request is paused at a breakpoint to allow tampering.
	Response is paused at a breakpoint to allow tampering.
	Request used the `HEAD` or `OPTIONS` methods, or returned a `HTTP/204` status code. The `HEAD` and `OPTIONS` methods allow the client to acquire information about the target URL or server without downloading the content. The `HTTP/204` status code indicates that there is no response body.
	Request used the `POST` method to send data to the server.
	Response is HTML.
	Response is an image.
	Response is a script.
	Response is a Cascading Style Sheet (CSS).
	Response is Extensible Markup Language (XML).
	Response is JavaScript Object Notation (JSON).
	Response is an audio file.
	Response is a video file.
	Response is a Silverlight applet.
	Response is a Flash applet.
	Response is a font.
	Response's `Content-Type` is not a type for which a more specific icon is available.
	Request used the `CONNECT` method. This method is used to establish a tunnel through which encrypted HTTPS traffic flows.
	Session is a `CONNECT` tunnel inside which Google's SPDY protocol is used.
	Session is a `CONNECT` tunnel inside which the HTTP2 protocol is used.
	Session is an RPC-over-HTTP tunnel; most commonly used by Microsoft Outlook.
	Session wraps a HTML5 WebSocket connection.
	Response is a `HTTP/3xx` class redirect.
	Response is a `HTTP/401` or `HTTP/407` demand for client credentials, or a `HTTP/403` error indicating that access was denied.

	Response has a `HTTP/4xx` or `HTTP/5xx` error status code.
	Session was aborted by the client application, Fiddler, or the Server. This commonly occurs when the client browser began downloading of a page, but the user then navigated to a different page. The client browser responds by canceling all in-progress requests, leading to the Aborted Session state.
	Response is a `HTTP/206` partial response. Such responses are returned as a result of the client performing a `Range` request for only a portion of the file at the target URL.
	Response has a `HTTP/304` status, indicating that the client's cached copy is fresh.
	Session is unlocked, enabling modification after normal Session processing has completed.

FIDDLER'S TOOLBAR

The toolbar provides quick access to popular commands and settings.



The buttons and their functions are:

WinConfig	*Shown only on Windows 8 and later.* Launches a tool that configures "Immersive" apps to permit sending traffic to Fiddler. Hold the `CTRL` key while clicking to automatically configure all applications.
Comment icon	Add a Comment to all selected Sessions. The comment appears in a column of the Web Sessions list. Hold the `SHIFT` key while clicking to add a new mock Session to the list with the comment of your choice.
Replay	Reissue the selected requests to the server. Hold the `CTRL` key while clicking to reissue the requests without any Conditional Request headers (e.g. `If-Modified-Since` and `If-None-Match`). Hold the `SHIFT` key while clicking to be prompted to specify the number of times each request should be reissued.
Remove icon	Show a menu of options for removing Sessions from the Web Sessions list: • **Remove all** removes all Sessions from the list. • **Images** removes all Sessions that returned an image. • **CONNECTs** removes all `CONNECT` tunnels. • **Non-200s** removes all non-`HTTP/200` responses. • **Non-Browser** removes all requests that were not issued by a web browser. • **Complete & Unmarked** removes Sessions in the `Done` or `Aborted` states that are unmarked and have no Comment set. • **Duplicate response bodies** removes any Session which has no response body or has a response body identical to one received in an earlier Session.
Go	Resume all Sessions which are currently paused at a Request or Response breakpoint. Hold the `SHIFT` key while clicking to resume only selected Sessions.
Stream	Enable the Stream toggle to deactivate response buffering for all responses except those for which a breakpoint is set.
Decode	Enable the Decode toggle to remove all HTTP Content and Transfer encodings from requests and responses.
Keep: *value*	Control how many Sessions are stored in the Web Sessions list. When the limit is reached, Fiddler will begin removing older Sessions to attempt to limit the list to the desired value. Incomplete Sessions and those with comments, markers, or open Inspector windows are not removed.
Process Filter	Drag and drop the Process Filter icon to an application to create a filter that hides all traffic except that originating from the selected process. Right-click the Process Filter icon to clear a previously set filter.

Simply declaring the attributed `m_Hide304s` variable doesn't yet do anything useful—the variable simply tracks the state of the menu item. The *functionality* of the rule is provided by a block of code added to the `OnBeforeResponse` method found later in the script:

```
if (m_Hide304s && (304 == oSession.responseCode)) {

  oSession["ui-hide"] = "true";

  // Note: This block could be placed in the OnPeekAtResponseHeaders method,
  // since it does not depend upon the availability of the response body.
}
```

This block first checks to see if the rule is enabled and, if so, checks that the server returned a `HTTP/304`. If so, the block sets the `ui-hide` flag on the Session, which causes it to be hidden from the Web Sessions list.

You can also use different forms of the `RulesOption` attribute to create submenus of options. To do so, provide the name of the submenu as the second parameter of the attribute. For instance, the following three fields create a Performance submenu that exposes the three options:

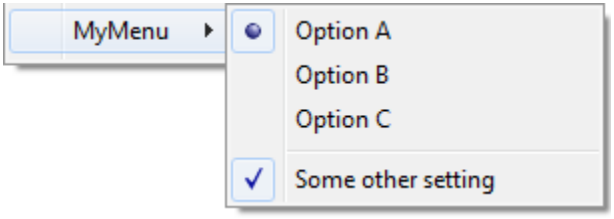
```
RulesOption("Simulate &Modem Speeds", "Per&formance")
public static var m_SimulateModem: boolean = false;

RulesOption("&Disable Caching", "Per&formance")
public static var m_DisableCaching: boolean = false;

RulesOption("&Show Time-to-Last-Byte", "Per&formance")
public static var m_ShowTTLB: boolean = false;
```

If you would like some of the items on your submenu to be mutually exclusive (showing as a radio group instead of a set of checkboxes), you can set a third boolean parameter to `true`, and you can add a splitter after a menu item by setting yet a fourth boolean parameter to `true`.

To create this menu:



Add the following script:

```
RulesOption("Option A", "MyMenu", true)
public static var m_OptionA: boolean = true;

RulesOption("Option B", "MyMenu", true)
```

Extend Fiddler's UI - Adding Tabs

While most major UI extensions are built in C#, you can add simple UI tabs very easily using FiddlerScript.

For example, say that you've decided that `http://httpstatusdogs.com` is the coolest site on the Internet, and you want to enhance Fiddler with this meme. Doing so is super-simple with FiddlerScript.

At the very top of the script file, add the line:

```
import System.Text;
```

Then, move the cursor to just inside the Handlers class. There, add the following code:

```
public BindUITab("HTTPStatusDogs", true)
static function ShowStatusDogs(arrSess: Session[]):String
{
  if (arrSess.Length < 1) return "<html>Please select one or more Sessions.</html>";

  var oSB: System.Text.StringBuilder = new System.Text.StringBuilder();
  oSB.Append("<html><head>");
  oSB.Append("<style>iframe { width: '100%'; height: 600px; frameBorder:0 }</style>");
  oSB.Append("</head><body>");
  for (var i:int = 0; i<arrSess.Length; i++)
  {
    oSB.AppendFormat(
      "<iframe frameBorder=0 scrolling='no' src='http://httpstatusdogs.com/{0}'></iframe>",
      arrSess[i].responseCode);
  }
  oSB.Append("</body></html>");
  return oSB.ToString();
}
```

When you save the script, it will automatically recompile and a new "HTTPStatusDogs" tab will appear; when you activate it, the image for each Selected Session's HTTP response code will be shown in the tab.

The "magic" that makes this work is invoked by the `BindUITab` attribute atop the function declaration:

```
public BindUITab("HTTPStatusDogs", true)
static function ShowStatusDogs(arrSess: Session[]):String
```

The presence of this attribute informs Fiddler that the following function will provide data to be rendered to a new tab, whose name is provided by the first parameter, `"HTTPStatusDogs"`. The second parameter, `true`, indicates that the string returned by the function should be rendered as HTML in a web browser view. To easily debug your HTML, change that true to `false`, and Fiddler will instead show the returned string as plain text in a textbox.

BitFlags

Each Session's `BitFlags` property holds zero or more `SessionFlags` that supply commonly-queried state information. The `SessionFlags` enumeration includes:

SessionFlags	Description
`None`	No flags are set.
`IsHTTPS`	The request's URI has a HTTPS target.
`IsFTP`	The request's URI has a FTP target.
`Ignored`	The Session should be ignored: hide from the Web Sessions list, do not store the request or response, and do not invoke event handlers.
`ClientPipeReused`	The request was read from a previously used connection from the client.
`ServerPipeReused`	The request reused an existing connection to the server.
`RequestStreamed`	The request was transmitted to the server as soon as the headers were complete; any request body was streamed to the server as it was read from the client.
`ResponseStreamed`	The response was not buffered and was instead streamed to the client as it was read from the server.
`RequestGeneratedByFiddler`	The request was generated by Fiddler itself (e.g. from the Composer tab).
`ResponseGeneratedByFiddler`	The response was generated by Fiddler itself (e.g. AutoResponder or the `utilCreateResponseAndBypassServer` method).
`LoadedFromSAZ`	This previously-captured Session was reloaded from a SAZ file.
`ImportedFromOtherTool`	The Session was imported by a Transcoder.
`SentToGateway`	The request was sent to an upstream (CERN) gateway proxy.
`IsBlindTunnel`	This `CONNECT` tunnel "blindly" shuttles bytes across without decryption.
`IsDecryptingTunnel`	This `CONNECT` tunnel decrypts HTTPS traffic as it flows through.
`ServedFromCache`	The response was served from a client cache, bypassing Fiddler. Fiddler only "sees" this Session if other software reports it to Fiddler, because Fiddler only observes network traffic.
`ProtocolViolationInRequest`	There was a HTTP Protocol violation in the client's request.
`ProtocolViolationInResponse`	There was a HTTP Protocol violation in the server's response.
`ResponseBodyDropped`	The response body was not stored, e.g. because the Preference `fiddler.network.streaming.ForgetStreamedData` is set.
`IsWebSocketTunnel`	This `CONNECT` tunnel is used for WebSocket traffic.
`SentToSOCKSGateway`	The request was proxied using the SOCKS protocol.
`RequestBodyDropped`	The request body was not stored, e.g. because the Session's `Log-Drop-Request-Body` flag was set.
`IsRPCTunnel`	The request created an RPC-over-HTTPS tunnel.

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

```
using System;
using System.IO;
using System.Text;
using System.Collections.Generic;
using System.Windows.Forms;
using Fiddler;

[assembly: AssemblyVersion("1.0.0.0")]
[assembly: Fiddler.RequiredVersion("2.4.9.5")]

// Note that this Transcoder only works when loaded by Fiddler itself; it will
// not work from a FiddlerCore-based application. The reason is that the output
// uses the columns shown in Fiddler's Web Sessions list, and FiddlerCore has
// no such list.

// Ensure your class is public, or Fiddler won't see it!
[ProfferFormat("Tab-Separated Values", "Session List in Tab-Delimited Format")]
[ProfferFormat("Comma-Separated Values",
    "Session List in Comma-Delimited Format; import into Excel or other tools")]
public class CSVTranscoder : ISessionExporter
{
  public bool ExportSessions(string sFormat, Session[] oSessions,
      Dictionary<string, object> dictOptions,
      EventHandler<ProgressCallbackEventArgs> evtProgressNotifications)
  {
      bool bResult = false;
      string chSplit;

      // Determine if we already have a filename
      // from the dictOptions collection
      string sFilename = null;
      if (null != dictOptions && dictOptions.ContainsKey("Filename"))
      {
          sFilename = dictOptions["Filename"] as string;
      }

      // If we don't yet have a filename, prompt the user
      // with a File Save dialog, using the correct file extension
      // for the export format they selected
```

APPENDIX B: COMMAND LINE SYNTAX

`Fiddler.exe` accepts zero or more command-line arguments, consisting of zero or more option flags and a single filename of a file to load on startup.

```
fiddler.exe [options] [FileToLoad]
```

The `FileToLoad` argument may either be a SAZ file or a file of an importable type (e.g. `.pcap` or `.har`).

Fiddler registers itself in Windows' `AppPaths` key so that you can launch it by typing `fiddler` in the shell's Start > Run prompt (hit `Windows+R`) instead of specifying a fully-qualified path to `fiddler.exe`.

Option Flags

Option flags may be preceded by either a `/` or `-` character.

Flag	Description
`-?`	Show the list of available command line arguments.
`-viewer`	Open a **Error! Reference source not found.** instance.
`-quiet`	Launch in "quiet" mode, where prompts and alerts are suppressed, and the main window is minimized to the system tray. This mode is most often used when Fiddler is running as a part of an automated script.
`-noattach`	Do not register as the system proxy on startup, even if otherwise configured to do so. You can manually register Fiddler as the proxy for an individual application or set it as the system proxy using the option on the File menu.
`-noversioncheck`	Do not send a web service request to check for updates on startup.
`-extoff`	Do not load Fiddler Inspectors or Extensions. This flag is used to troubleshoot problems related to buggy extensions.
`-noscript`	Do not load FiddlerScript. This flag is used to determine whether your Fiddler-Script is causing some problem.
`-port:####`	Specify the port that Fiddler should listen on, overriding the default setting configured in the Fiddler Options window.

Examples

Launch Fiddler without attaching:

```
"C:\program files (x86)\Fiddler2\fiddler.exe" -noattach
```

Launch Fiddler with no UI, running on port `1234`:

```
fiddler -port:1234 -quiet
```

Open a SAZ file in a new Fiddler Viewer instance:

```
fiddler -viewer "C:\users\joe\desktop\Sample.saz"
```

Networking Flags

The following flags control and log Fiddler's use of the network:

Flag Name	`x-overrideHost`
Explanation	Controls the hostname used for DNS resolution (and optionally the target port) when deciding what address this request should be sent to. This flag will not change any of the Headers in the request itself.
Supported Values	Specify either an alternative hostname or an IP Address (and optionally a Port) to which this request should be targeted. Setting this flag may have no effect if the request is sent to an upstream gateway, because the upstream gateway will perform its own DNS resolution.

Flag Name	`x-overrideHostName`
Explanation	Controls the hostname used for DNS resolution when deciding what address this request should be sent to. This flag will not change any of the Headers in the request itself.
Supported Values	Specify either an alternative hostname or an IP Address to which this request should be targeted. Setting this flag may have no effect if the request is sent to an upstream gateway, because the upstream gateway will perform its own DNS resolution.

Flag Name	`x-OverrideGateway`
Explanation	Controls which upstream gateway proxy, if any, this request is sent to.
Supported Values	Provide a string that specifies the target gateway proxy in the format `ProxyHost:Port`, for example, `myproxy:8080`. The provided address information will be used instead of any default gateway proxy. If you prefix the string with `socks=` the provided gateway will be used as a SOCKS proxy. For instance, the string `socks=127.0.0.1:9150` can be used to send traffic to a locally running instance of the Tor SOCKS proxy. A value of `DIRECT` means that the request will be sent directly to the server, bypassing any gateway. This value is equivalent to setting the Session's `bypassGateway` boolean to `true`.

Flag Name	`x-ReplyWithTunnel`
Explanation	When set on a `CONNECT` tunnel's Session, this flag causes Fiddler to automatically respond with a `200 OK` response without contacting the server or gateway. This flag is set by the AutoResponder to enable capture of HTTPS requests from a client even when the system is offline. Otherwise, the `CONNECT`'s failure would prevent subsequent HTTPS requests from being sent by the client.
Supported Values	Any value will result in returning a `200 OK` response to the `CONNECT`.

Flag Name	`FTP-UseASCII`

APPENDIX D: PREFERENCES

Fiddler's Preferences system allows you to control myriad aspects of Fiddler's behavior. This appendix contains a list of the Preferences that affect Fiddler and its default set of extensions.

Network Preferences

The following preferences control Fiddler's network behavior:

Name	`fiddler.network.timeouts.dnscache`
Default	`150000` (2.5 minutes)
Explanation	Number of milliseconds for which Fiddler should cache DNS lookup results.

Name	`fiddler.network.timeouts.serverpipe.reuse`
Default	`115000` (115 seconds)
Explanation	Number of milliseconds for which Fiddler is willing to leave a server connection idle. A connection which has been idle for this time without being reused will be closed. The Firefox team has found that values over 115 seconds can cause problems with buggy servers.

Name	`fiddler.network.timeouts.clientpipe.receive.initial`
Default	`60000` (1 minute)
Explanation	Number of milliseconds for which Fiddler is willing to wait for a client to begin sending a request on a newly established connection. After this timeout expires, Fiddler will send a `HTTP/408` timeout and close the connection from the client.

Name	`fiddler.network.timeouts.clientpipe.receive.reuse`
Default	`60000` (30 seconds)
Explanation	Number of milliseconds for which Fiddler is willing to wait for a client to begin sending a request on a previously used connection. After this timeout expires, Fiddler will close the connection from the client.

Name	`fiddler.network.timeouts.serverpipe.send.initial`
Default	`-1` ("infinite")
Explanation	Number of milliseconds for which Fiddler is willing to wait when sending to a newly established server connection. After this timeout expires, Fiddler will close the connection to the server.

Name	`fiddler.network.timeouts.serverpipe.send.reuse`
Default	`-1` ("infinite")
Explanation	Number of milliseconds for which Fiddler is willing to wait when sending to a server on a previously used connection. After this timeout expires, Fiddler will close the connection to the server.

INDEX

E

F

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z