I'm not robot

reCAPTCHA

Continue

Debugging with fiddler the complete reference pdf

Debugging with Fiddler The complete reference from the creator of the Fiddler Web Debugger Eric Lawrence Debugging with Fiddler Cover Photo: Nicholas Wave; ©IStockPhoto.com/@by_nicholas Everything else: ©2012 Eric Lawrence. All rights reserved. Please don't pirate this book in whole or in part. Beyond the nine years I've spent developing Fiddler, I spent nine months on this book and I'd like to be able to pay for the coffee I drank while writing it. :) Book Version LULU 1.00 / Fiddler Version 2.3.9.9 Legalese Trademarks mentioned in this book are (obviously) the property of their respective owners, and are only used to identify the products or services mentioned. This book is provided "as is." In no event shall I, the author, be liable for any consequential, special, incidental or indirect damages of any kind arising out of the delivery, accuracy, or use of this book. This book was written with care, but no one warrants that it is error-free. On the contrary, I guarantee that this book contains at least a few errors, and I promise to be suitably embarrassed when you point them out to me (so that I may update the next version. @ericlaw on Twitter :) Introduction 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 Jane, a source of inspiration, encouragement, and immense patience as I've 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 beg my friends' forgiveness for declining so many gatherings over the years to stay home and write more code. 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. My colleague Jim Moore warrants a special mention for his careful review of the draft version of this book, and his valuable feedback on Fiddler over the years. I'd also like to acknowledge the many dozens of donors who supported Fiddler financially as it became costly to maintain and distribute. Lastly, I'd like to thank you, dear reader, for caring enough about Fiddler to pick up this book! iii | Introduction

Introduction ORIGINS First, a confession—the Fiddler Web Debugger is not the result of a grand vision or the ambition to develop the world's most popular debugging proxy. A tool born of necessity, I never set out to build a platform so flexible, powerful, and complicated that I'd be forced to spend nine months writing a book to explain how to fully take advantage of it. But here we are. Before diving into the technical chapters, I will begin by sharing the story behind how Fiddler came to be. As a student at the University of Maryland in the spring of 1999, I had the chance to interview for a Program Management position on a new team at Microsoft. One of my final interviewer's first questions was "How does HTTP work?" Knowing only the basics, I gave an incomplete and somewhat inaccurate answer, but didn't embarrass myself too badly. That summer and the following, I worked on features for the first version of SharePoint. On rare occasions, I found myself looking at web traffic in Microsoft Network Monitor (NetMon), a powerful but then primitive and difficult-to-use packet sniffer. In the early summer of 2001, I joined Microsoft full-time as the Program Manager for the Office Clip Art organizer client and web site. At that time, most of the developers and testers working on my team were new to web development, having previously been responsible for writing native code applications in C and C++. It was quickly apparent that the debugging process was overly cumbersome—many of my colleagues were loath to use NetMon. I even watched some developers debugging HTTP requests by hovering over variables in Visual Studio, examining the raw traffic in hex: Having authored a number of small utilities over the years, I was confident that I could code something to make web debugging simpler. My first simplistic effort was based on taking an existing C++ proxy server and making minor modifications to it so that it would spew HTTP traffic to the system console. 2 | Introduction It's generous to call this effort primitive—the proxy couldn't handle secure traffic of authentication protocols. Nontext content was another problem—comically, the utility would try to render binary content as ASCII. Old-school console users may recall that the octet 0x07 represents the "bell" character, and when it's displayed in the console, a system beep is played. Soon after its release, the hallways of the Office Online team sounded like a Las Vegas casino, as binary content flowed through the debug proxies running in each tester's office. Despite the very annoying limitations, this tool was still popular, and I was inspired to get started on the next version. I mocked up a quick little demo in Borland Delphi, a native code language used by most of my development work at the time. The colorful UI foreshadowed Fiddler's eventual appearance: However, as I pondered the security and memory-management implications of writing a proxy server in native code, I soon decided that my next effort would be developed in C#, a new language being developed by the Visual Studio team that my best friend had just joined. The notion of writing a HTTP proxy server from scratch in .NET posed just two significant challenges: I didn't really know how HTTP worked, and I didn't know how to code in C#. Fortunately, given a few bucks and a lot of spare weekends, both shortcomings would be remedied. Two books were my constant companions: HTTP: The Definitive Guide and the C# Cookbook. Chapter by chapter, I learned about HTTP and C#.NET, and week-by-week Fiddler came to life. About six months after I started, I had a basic version of Fiddler available: 3 | Introduction Like its predecessors, it too had a large number of limitations (and thousands of bugs) but it was eagerly debugged by colleagues whose PC speakers were in danger of burning out from the incessant beeps of my first primitive proxy utility. Over the subsequent years, Fiddler was progressively enhanced with two extensibility models, a mechanism to automatically generate and modify responses, and support for secure (HTTPS) traffic, FTP, and HTML5 WebSockets. About this book After nearly 9 years and one hundred version updates, 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 a huge number of scenarios for which Fiddler is useful, and a number of scenarios for which Fiddler isn't suitable, the most common tasks fall into a few buckets. Here's a rough guide to what you can and cannot do with Fiddler: An Incomplete List of Things Fiddler Can Do View web traffic from nearly any browser, client application, or service. Debug web traffic from most PCs and devices, including Mac/Linux systems, smart phones, and tablet computers. Chain to upstream proxy servers, including the TOR network. Run as a reverse proxy. Both projects have benefited. Publication of this book will roughly coincide with the release of Fiddler version 2.4.0.0 in the early summer of 2012. If you've read a previous 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, 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 I consulted during the development of Fiddler: Hypertext Transfer Protocol – HTTP/1.1 from 4 | Introduction 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 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! 5 | Introduction A QUICK PRIMER In this section, I'll provide some basic information about Fiddler that will help you get started and lend a foundation for the rest of the book. Basic Concepts Fiddler is a special-purpose proxy server that runs on your Windows-based computer. A typical Fiddler XCOPY Deployment. Simply install Fiddler on the target machine or a USB key. When installed, the server's responses are then returned to Fiddler, which passes the traffic back to the client. Virtually all programs that use web protocols support proxy servers, and therefore Fiddler can be used with almost any app. When it starts capturing, Fiddler registers itself with the Windows Internet (WinINET) networking component and requests that all applications begin directing their requests to Fiddler. Some applications do not automatically respect the Windows networking configuration and may require manual configuration in order for Fiddler to capture their traffic. Fiddler can be configured to work in more exotic scenarios, including server-to-server (e.g. Web Services) and device-to-server traffic (e.g. iPad or Windows Phone clients). By default, Fiddler is designed to automatically chain to any upstream proxy server that was configured before it began capturing—this allows Fiddler to work in network environments where a proxy server is already in use. Because Fiddler captures traffic from all locally-running processes, it supports a wide range of filters. These enable hiding of traffic, which is not of interest to you, as well as highlighting (using colors or font choice) of traffic deemed interesting. Filters can be applied based on the source of the traffic (e.g. the specific client process) or based on some characteristic of the traffic itself (e.g. what hostname the traffic is bound for, or what type of content the server returned). Fiddler supports a rich extensibility model which ranges from simple FiddlerScript to powerful Extensions which can be developed using any .NET language. Fiddler also supports several special-purpose extension types, the most popular of which is called an Inspector, so named for its ability to inspect a single transaction between a client and a server. Each Session appears as a single entry in the Web Sessions List shown at the left side of the Fiddler interface. Each Session object has a Request and a Response, representing what the client sent to the server and what the server returned to the client. The Session object also maintains a set of Flags that record metadata about the session, and a Timers object that stores timestamps logged in the course of processing the session. Proxy servers are not limited to simply viewing network traffic—Fiddler got its name from its ability to "fiddle" with outbound requests and inbound responses. Manual tampering of traffic may be performed by setting a request breakpoint. When a breakpoint is set, Fiddler will pause the processing of the session and permit manual alteration of the request and the response. Traffic rewriting may also be performed automatically by script or extensions running inside of Fiddler. By default, Fiddler operates in buffering mode, whereby the server's response is completely collected before any part of it is sent to the client. If the streaming mode is instead enabled, the server's response will be immediately returned to the client as it is downloaded. In streaming mode, response tampering is not possible. Captured sessions can be saved in a Session Archive Zip (SAZ) file for later viewing. This compressed file format contains the full request and response, as well as flags, timers, and other metadata. A lightweight capture-only tool known as FiddlerCap may be used by non-technical users to collect a SAZ file for analysis by experts. Fiddler supports Exporter extensions that allow storing captured sessions in myriad other formats for interoperability with other tools. Similarly, Fiddler supports Importer extensions that load traffic stored in other formats, including the HTTP Archive (HAR) format used by many browser's developer tools. Usage Scenarios Some of the most common questions I get are of the form: "Can I use Fiddler to accomplish ?" While there are a huge number of scenarios for which Fiddler is useful, and a number of scenarios for which Fiddler isn't suitable, the most common tasks fall into a few buckets. Here's a rough guide to what you can and cannot do with Fiddler: An Incomplete List of Things Fiddler Can Do View web traffic from nearly any browser, client application, or service. Debug non-web protocol traffic. o Fiddler works with HTTP, HTTPS, and FTP traffic only. It cannot "see" or alter traffic that runs on other protocols like SMTP, POP3, Telnet, IRC, etc. Handle huge requests or responses. o Fiddler cannot handle requests larger than 2 gigabytes in size. o Fiddler has limited ability to handle responses larger than 2gb. o Fiddler uses system memory and the pagefile to hold session data. Storing large numbers of sessions or huge requests or responses can result in slow performance. "Magically" remove bugs in a website for you. o While Fiddler will identify networking problems on your behalf, it generally cannot fix them without your help. I can't tell you how many times I've gotten emails asking: "What gives? I installed Fiddler but my website still has bugs!" With that quick primer out of the way, let's dive in! 8 | Introduction Exploring Fiddler GETTING STARTED Fiddler is available to download by visiting . It is strongly recommended that you only download Fiddler from this official source, as some unscrupulous websites have repackaged the program using installers that will also install unwanted software (for instance adware or browser toolbars). System Requirements Fiddler is supported on all versions of Windows from XP to Windows 8. The only prerequisite to installing Fiddler is having the Microsoft .NET Framework version 2 or later installed. The .NET Framework is present by default on Windows Vista or later, and can be installed for free on Windows XP. If your system has only .NET 4 installed (for instance, most Windows 8 systems do not have .NET 2.0, 3.0, or 3.5 present natively) you will need to download the "Fiddler4" package which is compiled to run natively on the .NET v4 CLR. Like most .NET programs, Fiddler will run in 32bit mode on 32bit systems and 64bit mode on 64bit systems. Fiddler runs best on 64bit systems, even those with less than 4 gigabytes of RAM. While not required, having Internet Explorer 9 or later installed provides Fiddler with a bit of extra functionality and is strongly recommended. In particular, when IE9+ is installed, Fiddler's WebView Inspector can display additional media types, and you will be able to see traffic sent to without additional configuration steps. Additionally, a configuration change can be made to Internet Explorer to indicate, via the X-Download-Initiator request header, why each request was issued. A basic install of Fiddler requires less than 5 megabytes of disk space, while installing the most popular add-ons might require another 5 megabytes or so. Fiddler will run on systems with 512 megabytes of RAM, but performance dramatically improves on systems with 2 gigabytes or more of memory. Installing Fiddler Fiddler's installer is simple—you'll be asked to accept the terms of the End User License Agreement (short summary: don't do anything bad, no warranties) and then to select a folder to which Fiddler will be installed. You should accept the default folder location unless there's a strong reason not too—some Fiddler addons will blindly assume Fiddler is in the default location and will fail to install properly if it's not. The default location is %ProgramFiles%\Fiddler2, which on most systems will be C:\Program Files\Fiddler2\, or C:\Program Files (x86)\Fiddler2\ on 64bit systems. Fiddler installs to the 32bit Program Files folder only because the installer technology is 32bit-- Fiddler itself will run in 64bit mode if the operating system supports it. After a successful install, a web page will open providing key information about getting started with Fiddler. The tool itself can be launched using the Start Menu or the Fiddler icon in your browser toolbar or menu. Alternatively, you can type fiddler2 in the Windows+R Run prompt. Permissions and XCOPY Deployment Fiddler requires Administrative permissions to install because it updates machine-wide folder and registry locations. If you don't have administrative permissions on a target machine, you can perform what is commonly called an 10 | Exploring Fiddler XCOPY Deployment. Simply install Fiddler on a different computer, then copy the %ProgramFiles%\Fiddler2 folder to the target machine or a USB key. When installed, the server's Updating Fiddler Fiddler is configured to check for updates to the program weekly. When Fiddler downloads an update, the next time you start Fiddler, it will automatically download and install the latest version. If you choose No, the prompt closes and no new version is installed; the next time you start Fiddler, you will again be notified of the new version. Installing Fiddler updates as they become available is strongly recommended, as each update typically adds features, improves performance, and removes bugs. At present, Fiddler Extensions do not participate in the auto-update feature, so you should periodically look for new versions of the Extensions (e.g. the Internet Explorer toolbar button and the FiddlerHook extension for Firefox), Fiddler itself will run successfully. Updating Fiddler If you want to manually update Fiddler, you can do so using the Help > Check for Updates... menu command. By default, Fiddler will check whether a new version of the tool is available. If a new version is found, you will be provided with an update notice. If you choose Yes, Fiddler launches your browser to download the latest program; after closing Fiddler, you can run the setup program yourself. If you instead choose Next Time, the next time you start Fiddler, it will automatically download and install the latest version.

you use on their developers' websites. Administrators may prevent users from using the Help > Check for Updates command. To do so, use the Windows Registry Editor to create a new REG_STRING value named BlockUpdateCheck with value True inside the registry key HKLM\SOFTWARE\Microsoft\Fiddler2. Uninstalling Fiddler If you later wish to uninstall Fiddler, you can do so using the Add/Remove Programs applet in your Windows Control Panel. Uninstalling Fiddler leaves your settings intact and therefore is not a means to resolve any configuration problems you may encounter. Instead, see the Troubleshooting section in the appendix of this book for more help. 11 | Exploring Fiddler THE FIDDLER USE R-INTERFACE The Fiddler User-Interface can be overwhelming because it exposes a great deal of information about web traffic and it offers a significant degree of customizability. On the left side of the Fiddler window is the Web Sessions list, and on the right a set of tabbed Views that permits rapid filtering and command invocation capabilities. Tips: Many UI controls have context menus. Right-click early and often. If the mouse cursor turns into a pointing hand, the UI element can be clicked. Standard keyboard shortcuts are broadly supported (Use CTRL+C to copy, CTRL+A to select all, CTRL+G to go to a specific line number, F1 for help, etc.). If a popup message box is shown, pressing CTRL+C will copy its text to the clipboard. Press the ESC key to dismiss dialog boxes or clear search boxes. To reset Fiddler's UI to its default layout, hold the SHIFT key while starting Fiddler. Remove a tab from the Views by middle-clicking the tab title. Double-click on a splitter to maximize the display area next to the splitter. The Web Sessions List The Web Sessions list is the most important feature of Fiddler—it displays a short summary of each Session that Fiddler has captured. Most operations in Fiddler begin by selecting one or more entries in the Web Sessions list and then [12 | Exploring Fiddler then performing an operation. To select more than one Session, hold the CTRL or SHIFT keys while clicking the desired rows. Double-click or press Enter to activate the default Inspectors for a single selected Session. When the Inspectors are activated, click Inspector is best suited to display the selected Session's request and response. Certain key information is displayed in the columns of the Web Sessions list, including: # – An identification number generated by Fiddler. Result – The status code from the response. Protocol – The protocol (HTTP/HTTPS/FTP) used by the Session. Host – The hostname and port of the server to which the request was sent. URL – The URL path, file, and query string from the request. Body – The number of bytes in the response body. Caching – Values from the Response's Expires and Cache-Control headers. Content-Type – The Content-Type header from the response. Process – The local Windows Process from which the traffic originated. Custom – Shows any u-CustomColumn flag value set by FiddlerScript. Comments – Shows any comment set using the builder's Comment button. You can resize and reorder the columns in the Web Sessions list by adjusting or dragging the column headers. To sort the entries in the Web Sessions list by the values in a specified column, click on that column's header. New columns may be added to the Web Sessions list using FiddlerScript, Extensions, or the QuickExec box. Understanding 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 (CONNECTs 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 a gray icon to offer a visual reference as to the Session's progress. Request type: The Request is being sent to the server. The Response is being downloaded from the server. The Request is paused at a breakpoint to allow tampering. 13 | Exploring Fiddler The Request used the HEAD or OPTIONS methods, or returned a HTTP204 status code. The HEAD and OPTIONS methods allow the client to acquire information about the target URL or server without actually downloading the specified content. The HTTP204 status code indicates that there is no response body for the specified URL. The Request used the POST method to send data to the server. The Response is HTML content. The Response is an image file. The Response is a script file. The Response is a Cascading Style Sheet (CSS) file. The Response is formatted as JavaScript Object Notation (JSON). The Response is an audio file. The Response is a video file. The Response is a Silverlight applet. The Response is a Flash applet. The Response is a font file. The Response's Content-Type is not a type for which a more specific icon is available. The Request used the CONNECT method. This method is used to establish a tunnel through which encrypted HTTPS traffic flows. The Session wraps a HTML5 WebSocket connection. The Response is a HTTP/3xx class redirect. The Response is a HTTP/401 or HTTP/407 status indicating that access was denied. The Response has a HTTP/4xx or HTTP/5xx error status code. The 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 cancelling all in-progress requests, leading to the Aborted Session state. The Request in 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. The Response is a HTTP/304 status to indicate that the client's cached copy is fresh. 14 | Exploring Fiddler The Session is unlocked, enabling modification after normal session processing has been completed. Keyboard Reference The following keyboard shortcuts are supported by the Web Sessions list: Spacebar Activate and scroll the currently-focused session into view. CTRL+A Select all sessions. ESC Unselect all sessions. CTRL+I Invert selection; selected sessions are unselected and vice versa. CTRL+X Remove all sessions (subject to the fiddler.ui.CtrlX.KeepMarked preference). Delete Remove selected sessions. Shift+Delete Remove all unselected sessions. R Replay the current request SHIFT+R Replay the current request, sending in If-Modified-Since and If-None-Match headers. SHIFT+U Unconditionally replay the current request multiple times (the count is specified in the subsequent prompt). P Attempt to select the "parent" request that set focus to it. This feature depends on the HTTP Referer header's value. C Attempt to select all "child" requests that were provoked by this response. This feature depends on the HTTP Referer header's value or the Location header on a redirect. D Select all "duplicate" requests that have the same request method and URL as the current session. ALT+Enter View the current Session's properties. SHIFT+Enter Launch Inspectors for this session in a new Fiddler window. Backspace or "Back" mouse button Insert Activate the previously selected session. CTRL+1 CTRL+2 CTRL+3 CTRL+4 CTRL+5 CTRL+6 M Mark the selected sessions in bold and one of the colors red, blue, gold, green, orange, or Toggle marking of the selected sessions using a bold red font. purple. Add a comment to the selected sessions. Right-clicking on the column headers at the top of the Web Sessions list offers a context menu with two options: Ensure all columns are visible Ensures that all columns are visible by resizing each to at least 30 pixels 15 | Exploring Fiddler Learn about configuring columns… wide. Opens a help topic which explains how to add new columns to the Web Sessions list. Right-clicking in the body of the Web Sessions list shows a large context menu. Many of the options are available only if one or more sessions are selected in the Web Sessions list. This menu can be extended by FiddlerScript, so it will often contain additional commands that are not listed here. The AutoScroll Session List option controls whether or not Fiddler automatically scrolls to the bottom of the Web Sessions list as new sessions are added. The Copy submenu enables you to copy the information of your choice from the sessions selected in the Web Sessions list: Just Url Copies the Urls of the selected Sessions to the clipboard, one per line. You may also invoke this command by pressing CTRL+U when focus is in the Web Sessions list. This column Copies the text in the column over which the context menu was opened. The text for each of the Terse summary Copies a terse summary of the selected Sessions to the clipboard. This summary includes the selected sessions to the clipboard, one per line. Request Method and URL, and the responses' status codes and status text. If a response is a HTTP/3xx redirect, the text will include the target Location header. You may also invoke this command by pressing CTRL+SHIFT+T when focus is in the Web Sessions list. Headers only Copies Sessions' headers to the clipboard. The text is copied in both plaintext and HTML formats, so you will see different results if you paste into an editor that handles only plain text (e.g. notepad) versus one than handles rich text (e.g. Microsoft Word). As this is the "default" command, double clicking on the Copy submenu itself will invoke a header copy. You may also invoke this command by pressing CTRL+SHIFT+C when focus is in the Web Sessions list. Session Copies complete Sessions to the clipboard. The text is copied in both plaintext and HTML formats, so you will see different results if you paste into an editor that handles only plain text text. You may also invoke this command by pressing CTRL+SHIFT+S when focus is in the Web Sessions list. Full Summary Copies the information shown in the Web Sessions list to the clipboard. The columns are delimited by tabs which enables you to paste the information into Microsoft Excel or other programs neatly. You may also invoke this command by pressing CTRL+C when focus is in the Web Sessions list. The Save submenu exposes options that allow you to save traffic to files: All Sessions Saves all sessions in the Web Sessions list to a SAZ file. 16 | Exploring Fiddler …and Open as Local File Saves the selected sessions' response bodies to individual files, then opens each file in the registered handler for the response's file type. If you hold the CTRL key while invoking this command, Windows will prompt you to select which application to use to open the file. Selected Sessions Request In ArchiveZip Saves Sessions selected in the Web Sessions list to a SAZ file. As Text Saves selected Sessions to a single text file. As Text (Headers Saves the selected Sessions' request and response headers to a only) single text file. Entire Response Saves the selected Sessions' request headers and bodies to individual files. Response Request Body Saves the selected Sessions' request bodies to individual files. Entire Response Saves the selected sessions' response headers and bodies to individual files. This option is useful if you would like to create responses file that you can later play back using the AutoResponder. Response Body Saves the selected sessions' request bodies to individual files. This option is useful if you would like to open the response body (say, an image) in another program than would be confused by the presence of HTTP response headers. The Remove Submenu allows you to remove All, Selected, or Unselected Sessions from the Web Sessions list. The CTRL+X, Delete, or Shift+Delete keys may be used to activate these commands when focus is in the Web Sessions list. The Comment… menu command allows you to add or update the Comment field for one or more selected Web Sessions. The Mark submenu allows you to select or mark the Sessions selected in the Web Sessions list. The font of the Sessions will be bolded and colored according to your choice. The Unmark option will unbold the selected sessions and revert their font color to the default. The Replay submenu commands that replay the currently selected requests as they were originally sent. If you hold SHIFT while invoking this command, Fiddler will prompt you for the number of times the requests should be repeated. You may invoke this command by pressing the R key in the Web Sessions list. Reissue Unconditionally replay the selected requests, sending no If-Modified-Since and If- Unconditionally None-Match headers to prevent the server from returning a HTTP/304 response. If you hold SHIFT while invoking this command, Fiddler will prompt you for the number of times the requests should be repeated. You may invoke this command by pressing the U 17 | Exploring Fiddler key in the Web Sessions list. Reissue and Edit Reissues the selected requests as they were originally sent, setting a request breakpoint on each new Session to allow you to use Fiddler's Inspectors to modify the requests before they are sent to the server. Revisit in IE For each selected request, navigates to the request URL in Internet Explorer. Note that Internet Explorer will always navigate using a GET request and its own headers and cookies, regardless of what HTTP methods and headers were captured on the session you are replaying. The Select submenu allows you to use your currently-selected session to select other sessions: Parent Request The Parent Session will attempt to use this request's Referer header and request ID to determine which session, if any, was responsible for this request being sent. For instance, if you invoke this command on a JavaScript request, the Session selected will typically be the HTML page that caused this JavaScript file to be downloaded. Pressing the P key while focus is in the Web Sessions list will also invoke this command. Child Requests The Child Requests option will attempt to use this request's URL and request ID to select any requests sent as a result of this response. For instance, if you invoke this command on a HTML Session, the Sessions selected will typically be the CSS, JS, and image files mentioned in the HTML markup. Pressing the C key while focus is in the Web Sessions list will also invoke this command. Duplicate Selects all sessions in the Web Sessions list that share the currently selected session's Request URL and HTTP method. The Compare command is available only when two sessions are selected in the Web Sessions list. Selecting this command will save the two sessions to temporary files and then launch the configured comparison tool to allow you to compare the requests and responses. The COMETPeek command will take a "snapshot" of an in-progress response, allowing you to inspect a partial response before it has been completed. This command is useful in cases where a web application is using the COMET pattern to return a never-ending stream of data to the client. Since the stream is, by-definition, without end, Fiddler wouldn't otherwise be able to show the response until the server terminates the connection. The Abort Session command will terminate the client and server connections for an in-progress request. The Clone Response command is only available when two sessions are selected in the Web Sessions list and one of the Sessions is currently paused at a breakpoint and the other Session is completed. The command will copy the response from the completed Session to the paused Session. This feature enables you to easily replicate a previously captured (or modified) response to satisfy a later request. 18 | Exploring Fiddler The Unlock for Editing menu command unlocks a single selected Session, allowing you to edit the completed Session's request and response using the Inspectors. You may invoke this command by pressing F2 when focus is in the Web Sessions list. The Inspect in New Window command opens a Session Inspector window for Editing menu command, and properties in a standalone window. The Properties… command opens the Session Properties window which displays information about the currently selected Session, including its timers, session flags, and information about how the request was routed. 19 | Exploring Fiddler FIDDLE R'S MAIN MENU The main menu is designed to provide access to almost all of Fiddler's functionality. The menu system can be augmented by FiddlerScript or Extensions, but in this section, we'll discuss only Fiddler's default menu commands. The File Menu The File menu contains commands to start and stop Fiddler's capture of web traffic, as well as to load and store captured traffic. The Capture Traffic toggle controls whether Fiddler is registered as the system proxy. When Fiddler is registered as the system proxy, applications that respect the WinINET proxy settings (e.g. Internet Explorer and many other browsers) will send their web requests to Fiddler. The FiddlerHook add-on for Firefox also respects this menu option. Even when it is not registered as the system proxy, Fiddler will capture the requests that are received. The Load Archive item allows you to reload previously-captured traffic stored in a SAZ file. The Save submenu exposes options that allow you to save traffic to files; the options on this menu are the same as those on the Web Sessions list's context menu. The Import Sessions… menu command enables you to import previously-captured traffic from another tool or file format. The Export Sessions menu allows you to export sessions captured by Fiddler to many different file formats. You may choose All Sessions… on this submenu. The Exit menu command will unregister Fiddler as the system proxy and close the tool. The Edit Menu Most of the commands on the Edit menu apply to the currently-selected Sessions in the Web Sessions menu, and thus most of the commands are disabled unless one or more Sessions are selected. The Copy Submenu enables you to copy the information of your choice from the Sessions selected in the Web Sessions list. The commands on this menu are the same as those on the Web Sessions list's context menu. The Remove Submenu allows you to remove All, Selected, or Unselected Sessions from the Web Sessions list. The CTRL+X, Delete, or Shift+Delete key combinations may be used to activate these commands when focus is in the Web Sessions list. The Select All command selects all entries in the Web Sessions list. You may invoke this command by pressing CTRL+A when the focus is in the Web Sessions list. 20 | Exploring Fiddler The Paste Files as Sessions command copies the contents of the clipboard. If the clipboard contains text and a DataURI can be found starting within the first 64 characters of the text, the DataURI will be parsed out of the markup and a new Session will be created and added to the Web Sessions list. This feature is very useful when you are using Fiddler to examine markup containing a DataURI-encoded image, and you want to see what the image looks like. Simply copy the DataURI from the markup and use the Paste Files as Sessions command to generate a new Session that you can then inspect using the ImageView Inspector. If the clipboard contains a binary image (for instance, if you've used Alt+PrintScrn hotkey to take a screenshot of the active window) that image will be used to generate and add a mock Session to the Web Sessions list. If the clipboard contains one or more files (e.g. copied from Windows Explorer) each file will be used to generate a mock Session that is added to the Web Sessions list. This feature is most useful when you are building a SAZ file to send to someone else and want to include a file that was not transmitted over the network. For instance, a web developer could use this option to include the source code behind a given ASPX page when sending a SAZ file of the output of that page as captured by Fiddler. The Mark submenu allows you to choose a color to mark the selected sessions in the Web Sessions list. The font of the Sessions will be bolded and colored according to your choice. The Unmark option will unbold the Sessions and revert their font color to the default. The Unlock for Editing menu command unlocks a single selected Session's request and response using the Inspectors. You may invoke this command by pressing F2 when focus is in the Web Sessions list. The Find Sessions… command opens the Find Sessions window to begin a search of captured traffic. You may press CTRL+F to invoke this command. The Rules Menu The Rules menu is easily extensible, and most of its commands are generated from the FiddlerScript file. The Hide Image Requests toggle controls whether images are shown in the Web Sessions list. You may invoke this command by pressing F7 when focus is in the Web Sessions list. The Hide CONNECTs toggle controls whether Sessions that use the CONNECT method are shown in the Web Sessions list. The CONNECT method is used by a client to establish a "raw" connection to a server, to carry either HTTPS or WebSocket traffic. The Automatic Breakpoints submenu allows you to control whether Fiddler automatically breaks Before Requests or After Responses. The Ignore Image toggle controls whether these breakpoints are applied to requests for images. The Customize Rules… command 21 | Exploring Fiddler When checked, the Require Proxy Authentication menu item will respond to any request that doesn't submit a Proxy-Authorization header by returning a HTTP/407 response demanding proxy credentials. This rule is useful for testing HTTP clients to ensure that they work properly in environments with authenticating proxy servers. When checked, the Apply GZIP Encoding menu item will apply GZIP HTTP compression to all responses except images, so long as the request contained an Accept-Encoding header that included the gzip token. This rule is useful to test that clients advertise GZIP support can actually decompress content. It also allows you to experiment with the performance and bytes-on-wire count for compressed traffic. When checked, the Remove All Encodings toggle removes all HTTP Content-Encodings and Transfer-Encodings from requests and responses. This rule is also exposed by the Decode button on the Fiddler toolbar. The remainder of the commands on this menu are provided by FiddlerScript, and thus your menu's commands may differ from the commands from those in the default FiddlerScript. You can examine the implementation of these commands (and add your own) by clicking the Customize Rules… option in the Rules menu. The Hide 304s option hides all Sessions whose responses bear the HTTP/304 Not Modified status. The Request Japanese Content option will set or replace all requests' Accept-Encoding header with the ja token, indicating that the client would like responses in the Japanese language. The User-Agents submenu allows you to set or replace all requests' User-Agent header with a specified value. You can select among the values provided, or use the Custom… option at the bottom of the menu to specify any value you like. Performance Submenu The Performance submenu exposes simple options that impact web performance. When enabled, the Simulate Modem Speeds option will set Flags on all subsequent sessions. The requestTrickle-delay flag is set to 300 to delay all sends by 300ms per kilobyte uploaded. The response-trickle-delay flag is set to 150 to delay all sends by 150ms/kb downloaded. When set, the Disable Caching option will remove all If-None-Match and If-Modified-Since request headers and add a Pragma: no-cache request header. It will also remove any Expires header on the response and set the Cache-Control response header to no-cache. This option cannot prevent a browser from reusing a previously cached response that was received before this option was enabled. Clear your browser's cache (CTRL+SHIFT+DELETE) after enabling this option for best results. The Show Time-to-Last-Byte will display the number of milliseconds that it took to receive the entire response in the Web Sessions list's Custom column. Similarly, Show Response Timestamp will show the timestamp at which the server's response was completely received by Fiddler in the Web Sessions list's Custom column. 22 | Exploring Fiddler The Cache Always Fresh option will automatically respond to any conditional HTTP request with a HTTP/304 response indicating that the client's cache is up-to-date. This option can dramatically improve performance when visiting sites that fail to set cache expiration dates properly. Pressing CTRL+F5 in the browser is usually enough to force a reload from the server in spite of this option, because browsers will omit the If-Modified-Since and IfNone-Match headers on forced-refresh requests. The Tools Menu The Fiddler Options… item opens the Fiddler Options window. The WinINET Options… item opens the Internet Explorer Internet Options window. The Clear WinINET Cache item will clear all files in the WinINET cache that is used by Internet Explorer and many other applications. The Clear WinINET Cookies item will clear all persistent WinINET cookies sent by Internet Explorer and many other applications. Session cookies will remain unaffected. The TextWizard… item launches the TextWizard window to permit encoding and decoding of text. The Compare Sessions item is enabled only if two sessions are selected in the Web Sessions list. When clicked, a differencing tool will be used to compare the two Sessions. The HOSTS… item opens Fiddler's Host Remapping tool. The View Menu The Stacked Layout list toggle reorganizes the Fiddler user-interface such that the Web Sessions list appears atop the tabs. This can be useful when you have added a lot of custom columns and want more space to see those columns. 23 | Exploring Fiddler The Toolbar toggle controls whether the Fiddler toolbar is visible. The Statistics item activates the Statistics tab; you can invoke this command by pressing F7. The Inspectors item activates the Inspectors tab, which you can invoke this command by pressing F8. The Composer item activates the Composer tab; you can invoke this command by pressing F9. The Minimize to Tray option or CTRL+M will minimize Fiddler to the system tray. The Stay on Top option will force Fiddler on top of all other windows. The AutoScroll Session List option controls whether or not Fiddler automatically scrolls to the bottom of the Web Sessions list as new sessions are added. The Refresh option and the F5 key refresh the information for the currently selected sessions. The Help Menu The Fiddler Help item enables this menu item open your web browser and navigate to the help homepage for Fiddler. The Get Started item opens your browser to the Fiddler discussion platform. You can access this item by pressing the F1 key. The Fiddler Community Discussions link opens your browser to the Fiddler discussion group, currently hosted by Google Groups. The HTTP References item opens a page with links to various references, including RFC2616. When you enable the Troubleshoot Filters… option, traffic that would otherwise be hidden is instead shown in a strikethrough font. The Comments column indicates all of Fiddler's filters was responsible for attempting to hide 24 | Exploring Fiddler the traffic. If you ever find that traffic is "missing" from Fiddler's Web Sessions list, this command is your best bet for resolving the problem. The Check for Updates… option contacts a web service to determine whether you're running the latest version of Fiddler. If you are not, you can install the newest version immediately or the next time Fiddler starts. The Send Feedback option composes an email message to my email address. The About Fiddler option opens a window showing the current Fiddler version. Fiddler's About Box At the top, the window shows supporting information, the running version of Fiddler, including its version number, whether it's a beta version, and when it was compiled. The next paragraph indicates whether Fiddler is running in 32bit or 64bit mode, and how much Virtual Memory and Working Set it is currently using. Below that, find the loaded version of the Microsoft .NET Framework, and the current Windows version information. Next, you'll see a counter recording the number of times you've started Fiddler. Below that, you'll see the hostname and port that Fiddler is currently using on your computer. The Listening-to lines show the network connections on which Fiddler is registered. The Gateway line shows information about any upstream proxy server. Lastly, you'll see my contact and copyright information. You may press Escape or Spacebar to close the window. Hit the CTRL+C to copy all of the text, or select a desired subset of the text with your mouse and press CTRL+C to copy only that text. 25 | Exploring Fiddler FIDDLE R'S TOOLBAR The Fiddler toolbar provides quick access to popular commands and settings. The buttons and their functions are: Replay Comment Replay Remove Resume Stream Decode Keep: Only Filter Find Save Camera Browse Clear Cache Click to add a Comment to all selected Sessions. The comment appears in a column of the Web Sessions list. Click to reissue the selected requests to the server again. Hold the CTRL key while clicking to reissue the requests without any Conditional Request headers (e.g. IfModified-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. Shows a menu of options that control how in-progress Sessions from the Web Sessions list. Remove all removes all Sessions from the list. Images removes all CONNECT tunnels. Non-200s 200s removes all non-HTTP/200 responses. Non-Browser removes all requests that were not issued by a web browser. Complete and Unmarked removes Sessions which are in the Done or Aborted state and which are unmarked and have no Comment set. Duplicate response removes any Session which has no response body or has received in an earlier Session in the list. Resumes all sessions which are currently paused at a Request or Response breakpoint. Enable the Stream toggle to disable response buffering for all responses except those for which a breakpoint was set. Enable the Decode toggle to remove all HTTP Content and Transfer encodings from requests and responses. The Keep dropdown controls how many Sessions are stored in the Web Sessions list. When the count 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. Drag and drop the Process Filter icon to an application to create a Filter which hides all traffic except for that which originates from the selected process. Right-click the Process Filter icon to clear a previously set filter. Opens the Find Sessions window. Saves all Sessions to a SAZ file. Adds a JPEG-formatted screenshot of the current Fiddler desktop. Process-based Filter Identifies whether Fiddler is currently configured to capture traffic from only one type of process. Click the panel to show a menu of process-type filtering options. Breakpoint Indicator Indicates whether Fiddler is configured to breakpoint all requests, all responses, or neither. Click the panel to quickly toggle breakpoint functionality. Session Counter Indicates the number of entries in the Web Sessions list. If one or more Sessions are selected, the indicator will show the number of Sessions selected and the total number of Sessions, e.g. "2 / 5". Status Information By default, shows the URL of the first selected Session. This panel is also used to show succinct information about the results of actions; for instance, when a SAZ file is loaded or saved, that fact is mentioned here. 27 | Exploring Fiddler The QUICKEXEC The QuickExec box below the Web Sessions list provides quick keyboard access to common operations. Use the Alt+Q hotkey to set focus to the QuickExec box when Fiddler is active—if Fiddler isn't active, first press focused, pressing CTRL+I will insert the URL of the first selected Session in the list. You can also drag/drop one or more Sessions from the Web Sessions list to insert their URLs in the QuickExec box, or you can drag/drop one or more files from your file system to insert their file paths. QuickExec Selection Commands The QuickExec allows you to quickly select traffic of interest based on search criteria you specify. After typing a selection command, hitting Enter will set focus to the Web Sessions list and all sessions returned any matches. Command ?search Action Selects Sessions whose URLs contain the specified search text. Example ? example.com/patchars This is the only field you can use. To prefixed searches, hitting Enter will set focus to the Web Sessions list, select type ?perfixed searches. Selecting Enter will select header=flag value All other search types require that you press Enter to begin the search. Select Sessions whose response Content-Type header contains the chosen type. Select Sessions whose response size is greater than size bytes. Note: the character "k" is converted to "000", allowing you to easily write bytes in kilobytes or even megabytes. 28 | Exploring Fiddler select @Response.Set-Cookie domain >40000000 >4000k >4KK Fiddler Options > General tab. Beyond its global hotkey, Fiddler supports a number of hotkeys that should always be available when Fiddler is active: ALT+Q CTRL+R CTRL+E CTRL+Shift+U CTRL+Up CTRL+F CTRL+I CTRL+O CTRL+Plus CTRL+Minus CTRL+M CTRL+SHIFT+DEL F12 Set focus to the QuickExec box Open the FiddlerScript rules editor Open the TextWizard Select the next session in the Web Sessions List Select the previous session in the Web Sessions List Activate the TextView Inspectors Activate the HeaderView Inspectors Set the font size to 8.25pt (the default) Increase the font size by 1pt Decrease the font size by 1pt (down to 7pt) Minimize Fiddler Clear the WinINET cache Toggle registration as the system proxy 33 | Exploring Fiddler STATISTICS TAB Fiddler's Statistics tab shows basic information about the currently selected Web Sessions. Textual information is shown at the top of the tab, while the bottom contains a pie chart showing a MIME-type breakdown of the traffic. When multiple sessions are selected, the data shown includes: Request Count The number of Sessions selected. Unique Hosts The count of unique hosts to which this traffic was sent. This field will not be shown if all selected traffic was sent to the same server. Bytes sent Total number of outbound bytes in the HTTP Request headers and bodies. The number of bytes for each is shown parenthetically after the total count. Bytes received Total number of inbound bytes in the HTTP Response headers and bodies. The number of bytes for each is shown parenthetically after the total count. Requests started at The time that the first byte of the first request was received by Fiddler. Responses completed at The time that the last byte of the last response was sent by Fiddler to the client application. Sequence (clock) duration The "clock time" between the start of the first request and the end of the last response. Aggregate session duration The summation of each selected Session's overall time from request to response. Because sessions are usually running in parallel, this time can be longer than the amount of "clock time" taken. Similarly, because Sessions might have started and finished with idle time between them, this time can be much shorter than the amount of "clock time" between the first and last request. DNS Lookup time The total amount of time spent in DNS resolution for all selected Sessions. This field is not shown if no time was spent on DNS resolution because the DNS cache held all of requests, or all connections had already been established. TCP/IP Connect duration The total amount of time spent in establishing TCP/IP connections for all selected Sessions. This field is not shown if no time was spent on TCP/IP connections because all requests occurred on reused connections. HTTPS Handshake duration The total amount of time spent in HTTPS handshaking for all selected Sessions. This field is not shown if no time was spent on HTTPS handshakes because all sessions were HTTP, decryption is disabled, or all requests occurred on already-secured connections. Response Codes The count of each unique HTTP response code found in the selected Sessions. Response Bytes by Content-Type The count of bytes for each response Content-Type in the selected Sessions. This information is also presented in a pie chart at the bottom of the tab. 34 | Exploring Fiddler Estimated Performance Very rough estimates of how long the selected traffic would take to download from different locales or when utilizing a variety of connection types. These estimates shown have been by widely inaccurate. If only a single Session is selected, the Statistics tab shows additional information about that selected Session. This information is recorded on a previous day, the date that the traffic was captured will also be listed-- this can be helpful when examining SAZ files captured by automated logging tools. If a single CONNECT tunnel is selected, the number of bytes sent and received on that tunnel will be shown (unless the tunnel is configured for HTTPS decryption, in which case this information can be inferred from the decrypted HTTPS Sessions in the Web Sessions list.) The bottom of the tab contains a pie chart which is hidden by default. Click the Show Chart link to show a chart of the selected Sessions. A slice is shown for the headers and each MIME type in the selected traffic; the size of the slice is based on the number of bytes of that type. The Copy this chart link at the bottom-left copies a bitmap of the pie chart to the clipboard. As you type search terms in the QuickExec box on the Web Sessions list (described on Page 29 | Exploring Fiddler it's really helpful for simple jobs, the Filters tab will probably meet your filtering needs. The Filters checkbox at the top-left of the tab enables the filters listed below to run against traffic as it is captured. When Filters is checked, the options below it can be adjusted to control whether a given session matches. Is hidden Is flagged on the Web Sessions list Is breakpointed for manual tampering Is blocked from being sent Has its headers modified automatically Fiddler continues to proxy traffic from hidden Sessions, even though they aren't shown in the Web Sessions list. 35 | Exploring Fiddler THE FILTERS TAB Filters provides a "point-and-click" means of applying simple filters against traffic as it is captured. Every setting on the tab can be duplicated in FiddlerScript using more precisely or powerfully) but for simple jobs, the Filters tab will probably meet your filtering needs. The Filters checkbox at the top-left of the tab enables the filters listed below to run against traffic as it is captured. When Filters is checked, the options below it can be adjusted to control whether a given session matches. Is hidden If you want to hide traffic that meets a given criterion. load a previously saved Filterset, and to run the current Filters against previously-captured traffic. The Help button launches a help topic about the Filters tab. Each group of filtering options is described below. Hosts The Hosts box provides filters that are based on the hostname of the request. The Zone Filter dropdown allows you to show traffic only to your Intranet (e.g. dotless hostnames) or only to the Internet (e.g. dotted hostnames). This is a useful option when debugging a site in one zone while referencing documentation from the other zone. 36 | Exploring Fiddler The Host Filter dropdown enables you to flag or hide traffic to domains specified in the textbox under the dropdown. Note that textbox does not automatically wildcard subdomains. That means that if you have "Show only the following Hosts" set and only fiddler2.com is in the list, you will not see traffic to www.fiddler2.com. To see traffic to subdomains, add a wildcard like *.fiddler2.com. This wildcarded entry will include traffic to test.fiddler2.com and sub.fiddler2.com, and so on. If you want to see traffic to the root domain fiddler2.com, change the wildcard to *fiddler2.com—this will include traffic to any domain that ends with fiddler2.com without requiring a leading dot. List multiple hosts by using a semicolon between each. Note: When the background of the text box is yellow, it means your changes have not yet been applied. Click anywhere outside the box to save your changes to the list. Client Process Process filters allow you to control which processes' traffic is shown within Fiddler. Fiddler can only determine which process issued a given request when the application process is running on the same computer as Fiddler itself. The Show only from dropdown contains a list of all currently-running processes on the system; select a process to show only traffic from that specific process. The Show only Internet Explorer traffic option hides the process' executable name begins with IE or the request bears a User-Agent header containing: MSIE. The Hide traffic from Service Host option will hide traffic from svchost.exe, a system process that synchronizes RSS Feeds and performs other background network activity. Request Headers Using these options, you can add or remove HTTP request headers, or flag requests that contain certain headers. The Show only if url contains option allows you to hide requests based on the URL. You can demand case-sensitivity using the EXACT prefix: 37 | Exploring Fiddler EXACT:example.com/q=Case+Sensitive+String …or you can use a regular expression: REGEX:(?insx).*\.(gif|png|jpg)$ #only show image requests The Flag requests with header option allows you to specify a HTTP header name, that if present, will cause the session to be included in the results. The Delete request header option allows you to specify a HTTP header name, that if present, will be removed from the request headers. The Set request header option allows you to create or update a HTTP Request header with a value of your choice. Response Status Code The Hide success option will hide any response whose status code is between 200 and 299 inclusive. These response codes are used to indicate a successful request. The Hide non-2xx option will hide responses whose status code is between 200 and 299 inclusive. The Hide Authentication demands option will hide responses with a status code of 401 or 407 status code used to prompt the client for credentials. The Hide redirects option will hide responses that redirect the request. The Hide Not Modified option will hide responses with the 304 status code used in response to a conditional validation request for a cached entity. The Hide non-2xx option will hide responses whose status code does not match a specified type. 38 | Exploring Fiddler Response Type and Size Using these options, you can control what types of responses appear within the Web Sessions list, and block responses matching certain criteria. The Type dropdown list allows you to hide Sessions whose Content-Type does not specify an image type. Show only HTML hides Sessions whose Content-Type does not specify an HTML type. Show only TEXT/CSS hides Sessions whose Content-Type does not specify the text/css type. Show only SCRIPTS hides Sessions whose Content-Type does not specify a script type. Hide IMAGE/* hides Sessions whose Content-Type specifies an image type. The Hide smaller than option allows you to hide responses smaller than the specified number of kilobytes. The Hide larger than option allows you to hide responses larger than the specified. The Time HeatMap checkbox sets a background color for each Session based on how long the server took to return a given response completely (computed using the Timers object: ServerDoneResponse - FiddlerBeginRequest). Responses that take less than 50 milliseconds show in shades of green. Responses that take between 50 and 300 milliseconds 500 milliseconds show in yellow. Responses that take more than 500 milliseconds show in shades of red. The Block script files option will return a HTTP/404 response instead of the returned response if the response declared an script Content-Type. The Block image files option will return a 404 response if the response declared an Adobe Flash (application/x-shockwave-flash) Content-Type. The Block CSS files option will return a 404 response if the response declared a CSS Content-Type. Response Headers Using these options, you can add or remove HTTP response headers, or flag responses that contain certain headers. The Flag responses that set cookies option will mark in bold any response that contains a Set-Cookie header. The Flag responses with header option allows you to specify a HTTP header name, that if present, will cause the Session to be included in the Web Sessions list. The Delete response header option allows you to specify a HTTP header name, that if present, will be removed from the response headers. The Set response header option allows you to create or update a HTTP response header with a value of your choice. 39 | Exploring Fiddler THE TIMELINE TAB This tab allows you to view between 1 and 250 selected Sessions on a waterfall diagram, which is useful both for Performance analysis and to understand how requests relate to one another. The bulk of the tab is the view of the selected traffic. Across the top of the tab is the title which displays the mode ("Transfer Timeline", by default). The Help link at the top-right opens a help topic about the feature. Right-click in the body of the tab to see the context menu. It exposes the following options: AutoScale Chart When enabled, the chart is scaled such that the entire chart fits in the horizontal width of the tab without a horizontal scrollbar. Copy Chart Clicking this option copies the chart to the clipboard in bitmap format, suitable for pasting into other documents. Mode (dropdown) The Mode dropdown controls how the chart is displayed: Timeline – displays each Session on its own line, with a colored bar representing its duration. Client Pipe Map – Displays each connection between Fiddler and an upstream server on its own line. Connections which were reused for multiple sessions will display multiple colored bars. The Client Pipe Map and Server Pipe Map modes show how the client and server reuse connections, which can be helpful in identifying performance bottlenecks. The Timeline tab does not presently display any CONNECT tunnels because the traffic flowing through such tunnels is either opaque or tracked by one or more decrypted HTTPS Web Sessions entries. Mode: Timeline In this mode, Fiddler will show each selected Session on its own line, with different colored bars will show more information about the entry in the status bar at the bottom of the tab. Double-clicking the entry will inspect that Session in a new window. The color of the bar is determined by the MIME type of the response; light-green for images, dark-green for JavaScript, purple for CSS, and blue otherwise. 40 | Exploring Fiddler The start of the transfer bar is drawn at the time when the client sends the request to Fiddler (Timers.ClientBeginRequest). The end of the transfer bar is drawn at the time when the response to the client is completely received (Timers.ClientDoneResponse). If the bar is "hatched" rather than solid, that means that the HTTP response was buffered by Fiddler rather than being streamed to the client as it was read to the server. Buffering alters the waterfall diagram, as you can see in the Timeline below where most of the images begin to download only after the server's response (Timers.ServerBeginResponse). The two small circles before the bar indicate whether the Session was transmitted on reused connections. A green circle shows that a connection was reused while a red circle shows the connection was used for this request. The gray arrow icon indicates that the server's response was a redirect (302). The red ! icon server returned an error code (4xx, 5xx). indicates that the Mode: Client Pipe Map In this mode, the Timeline will show each outbound connection from a client on its own line. At the left edge of the chart is the connection identifier, which shows the process id and client port number. For instance 41 | Exploring Fiddler iexplore:1364(p14421) identifies that the client was Internet Explorer process number 1364, using port 14421 to connect to Fiddler. Connections which were reused for multiple sessions will display multiple horizontal bars. Mode: Server Pipe Map In this mode, the Timeline will show each outbound connection from Fiddler to a server on its own line. At the left edge of the chart is the connection identifier, which shows the outbound port number and the hostname of the target server. For instance p14357>twimgs.com indicates that Fiddler used port 14357 on your computer to establish a connection to port 80 on twimgs.com. Connections which were reused for multiple sessions will display multiple horizontal bars. Using the Timeline for Performance Analysis The Timeline tab provides an information-rich view of how your application is using the network's, and cases where connections were unnecessarily closed (the X icons). Using this information, you may be able to adjust your application to better order requests to optimize network performance. 42 | Exploring Fiddler THE AUTORESPONDER TAB This tab exposes one of the most powerful functionality in Fiddler. It permits you to create rules that will automatically generate or replay a response, typically reacting to a previously-captured response without hitting the server. Across the top of the tab are a set of options which control the AutoResponder's behavior, while most of the tab contains the Rules List that maps inbound URL Match Conditions to Actions. The Enable automatic responses checkbox controls whether the AutoResponder is active. When this option is unchecked, the rest of the tab is disabled. The Unmatched requests passthrough option controls what happens when a Session does not match any of the specified rules. When ticked, unmatched requests are sent to the server normally, as if the AutoResponder feature didn't exist. When unticked, Fiddler will generate a HTTP/404 Not Found response for any unmatched HTTP request that does not match any of the rules. If a client sends a conditional request containing If-None-Match or If-Modified-Since headers, the AutoResponder will return a HTTP/304 Not Modified response. The Enable Latency option permits specification of a response's latency, disable the option for improved performance. The Import button allows you to import a previously-captured SAZ file; each Session in the imported file will be used to generate a new rule in the Rules list. You can also import a FARX file which contains rules exported from the AutoResponder tab. The center of the tab is a listview that contains the rules. The first column specifies the Match Condition, which is used to determine if an incoming request matches the rule. The checkbox preceding the Match Condition's text controls whether the rule is enabled. The second column contains the action to undertake if the rule is matched. The Action Text may specify a local filename to return, or may specify another type of action. When a rule is selected, the Rule Editor box at the bottom of the tab enables you to adjust the rule's Match Condition and Action Text. The Test box allows you to test your Match Condition against the sample URL of your choice; this feature can be helpful if your regular expression skills have grown rusty over the years. 43 | Exploring Fiddler After making any changes to the Match Condition or Action Text, click the Save button to update the rule. If you select multiple rules, the Rule Editor will hide the Match Condition and provide the option to update all selected rules' Latency at once. This capability allows you to easily specify multiple rules that trigger the same response. Specifying the Match Condition By default, the Match Condition performs a case-insensitive match against the target URL. So, if you want a rule to match any request whose URL contains the word fuzzle, simply type fuzzle in the box. The rule will match any of the following URLs: A Match Condition of * will match all inbound requests. By specifying the NOT-fuzzle in the box. Unless the text fuzzle is found anywhere in the URL's match should fail, specifying the EXACT: prefix, you can require request whose URL does not case-insensitively contain the provided string. So, if you want a rule to match any request whose URL matches exactly, say, http://www.fiddler2.com/foo, use the EXACT:www.fiddler2.com/foo. By specifying the REGEX: prefix, you can provide a regular expression which will be evaluated against the request URL. Expressions are evaluated using the .NET regular expression engine. The are many great books that explain how to develop powerful regular expressions; this book is not one of them. I recommend to get started. A few sample regular-expression rules are: Rule REGEX:.+\.jpg$ REGEX:.+\.(jpg|gif|png)$ REGEX:^https.+$ REGEX:(?insx).*\.(jpg|gif|png)$ Any URL that ends with .jpg Any URL that ends with .jpg or .gif or .png Any URL that begins with https Any URL that ends with .jpg, .gif, or .png (case-insensitively) You can specify regular expression options (like case-sensitivity) by leading the expression 44 | Exploring Fiddler REGEX:.+\.jpg$ REGEX:.+\.(jpg|gif|png)$ REGEX:^https.+$ REGEX:*.\(jpg|gif|png)$ Any URL that ends with .jpg Any URL that ends with .jpg or .gif or .png Any URL that begins with https Any URL that ends with .jpg, .gif, or .png (case-insensitively) You can specify regular expression options (like case-sensitivity) by leading the expression with an appropriate declaration. The option string (?insx) works well; it replaces the 7-bit ASCII of a long forward-slashes for backslashes when mapping URLs to files, so the rule: Match Text REGEX:/?xxers/(^?name)?(?.*) is the case-insensitively requires explicit capture groups, enables single-line syntax, and enables comments after the # character. Comments are useful to combat regular expression "write once, read never" nature. Without an informative comment, you may quickly find yourself scratching your head several months (or hours) after you wrote what your carefully crafted regular expression was meant to match. Matching Against Request Bodies In some cases, a site may use the same request URL for many undifferentiated operations, specifying the operation desired in the request's body instead of the URL. You may extend your Match Condition to require a Match Condition for the request's body using the URLWithBody: prefix for your Match Condition. When this prefix is used, the portion of the string up to the first space character is used as the Match Condition for the request's URL, while the remainder of the string is used as a Match Condition for the string-representation of the request's body. You could specify the Match Condition as specific as possible to minimize the chance that the AutoResponder is forced to evaluate. If a request has no body, the match against the body will fail. The regular expressions "write once, read never" nature. Without an informative comment, you may quickly find yourself scratching your head several months (or hours) after you wrote what your carefully crafted regular expression was meant to match. 45 | Exploring Fiddler Any Action may be either Final or Non-Final. Non-Final Actions allow the request to continue matching against subsequent rules. Non-Final actions include redirecting the request, or performing some other action. As soon a rule specifying a Final action is reached, the matching process exits and no further rules are processed for that Session. Rules are evaluated in the order that they appear, so you should adjust the order of the rules to match your needs. Action Text filename Return Return the contents of filename as the response. Final? yes Return the contents of the target URL. Redirect to the target URL. Unlike a plain URL, yes using the *redir prefix ensures that the client knows where its request is going, so it can send the correct cookies, etc. *bpu Set a request breakpoint. no *bpafter Set a response breakpoint. no *delay:## Delay sending request to the server by the specified number of no milliseconds. *flag:flagname=value Set a Session flag on the Session. no *drop Close the client connection immediately without sending a re- yes sponse. The closure is graceful at the TCP/IP level, returning a FIN to the client. *reset Close the client connection immediately without sending a re- yes sponse. The closure is abrupt at the TCP/IP level, returning a RST to the client. *exit Stop processing rules. yes AutoResponse is most commonly used to return a local file from disk. When that file is loaded, Fiddler will scan it for a set of "magic strings." When the file is returned, the session is shown with a Lavender background in the Web Sessions list. Using RegEx Replacements in Action Text Fiddler's AutoResponder permits you to use regular expression groups to generate text for instance, the rule Match Text REGEX:.+assets/(.*)-66 | Exploring Fiddler Action Text 1…will map a request for to. The following rule: Match Text REGEX:.example\.com.* Action Text … will rewrite the inbound URL so that all URLs containing example.com are passed as a URL parameter to example.com. For 46 | Exploring Fiddler instance, the rule Match Text REGEX:.+assets/(.*)-66 | Exploring Fiddler Action Text 1…will map a request for to. The following rule: Match Text REGEX:.example\.com.* Action Text "crunched" or "minified" versions of their scripts are downloaded when the original version of the script available for debugging purposes. For instance, the crunched version of a script available at the URL …and the uncrunched version will be available at: You can write a RegEx replacement rule that maps requests for the crunched version to the original version like so: Match Text REGEX:(?insx) ^ (?'path'.+)_c\.js$ Action Text ${path}.js This rule captures the path to the file in the variable named path and that variable is later used to replace the ${path} token in the Action Text. Beyond changing the URL's host or path, you can create similar rules that modify, add, or remove query string parameters. The replacement functionality is smart enough to avoid forward-slashes for backslashes when mapping URLs to files, so the rule: Match Text REGEX:.+assets/(?'name')?(?.*) is an… AutoResponder allows you to easily create new AutoResponder rules using drag-and-drop. You can drag-and-drop files or folders full of files from Windows Explorer to automatically generate rules for those files. Alternatively, you can drag-drop sessions from the Web Sessions list to reuse responses captured previously. You may edit a rule's response by selecting the rule and hitting Enter or by selecting the Edit Response link from the rule's context menu. 47 | Exploring Fiddler The context menu on the Rules listview exposes the following commands: Remove Remove Rule Remove Set Latency… Clone Rule Edit Response Generate File Edit File… Open URL Export All… Deletes the selected rules. Alternatively, select one or more rules and press Delete. Moves the rule one slot earlier in the list. Alternatively, select one rule and press the Plus key. Moves the rule one slot later in the list. Alternatively, select one rule and press the Minus key. Prompts for a number of milliseconds to use as a latency value when the rule is matched. For instance, if you specify 50 and a filename, Fiddler will wait 50 milliseconds before sending it to as the response. You may specify a leading + or - indicator if you wish to adjust the current latency value rather than setting it to an absolute value. Makes a copy of the current rule and appends it to the list. For a rule that is backed by a file, opens that file for editing. For a rule that is backed by a previously-captured response, opens a standalone Inspectors window to enable editing the response. For a rule that is backed by a previously-captured response, saves that response to a file and adjusts the rule to point to that file. Use this when you need to edit a response using a standalone editor which is not able to edit the specified response. Use this to edit a response captured using the default web browser. Prompts you to save the current set of rules as a FARX file. This file can be later imported to reload the rules. FARX Files Fiddler AutoResponder XML (FARX) files contain a set of AutoResponder rules. Your current set of AutoResponder rules is automatically saved on exit and reloaded when you restart Fiddler. This automatically saved file is named AutoResponder.xml, but if you use the Export command to save a separate file, it will have a .farx file extension. FARX files contain the full set of rules, including any previously captured responses used for playback. Binary responses are base64-encoded for storage. You may manually create small set of rules with large responses) can cause a FARX file to grow very large; the set of AutoResponder rules is automatically reloaded when Fiddler starts up, a large default FARX file can make Fiddler slow to start. To avoid problems with slow startup, export large FARX rules to a FARX file of your choosing and remove those rules from your default list before closing Fiddler. 48 | Exploring Fiddler THE TE XTWIZARD When interacting with web content, text is often encoded using one or more formats. The TextWizard enables you to quickly transform text to and from popular formats. The TextWizard can be opened by clicking Tools > TextWizard or by pressing CTRL+E. You may open multiple copies of the TextWizard at the same time. At the top of the TextWizard is the Input box into which you can type or paste the input text. Along the right side, there are a number of Transformations you can apply to the input text. If you'd like to "chain" multiple transformations in sequence, use the Send output to input button to copy the last generation's Output into the Input box, then select the next transformation to apply. At the two tool opens, if text (32kb or less) is present on the clipboard, it is automatically pasted into the Input box. Each time the tool opens, all text is selected so that the indicates the number of characters in the input and the number of characters of output generated. If your transformation results in the creation of a null character (which terminates the string prematurely), the TextWizard will replace the null character with the Unicode Replacement Character (0xFFFD, �). The View bytes checkbox allows you to view the output in hexadecimal, which can be useful in cases where you've to replace the input with the output text, used when chaining a series of transformations. Available transformations include: To Base64 From Base64 URLEncode URLDecode HexEncode To JS string From JS string HTML Encode HTML Decode To UTF-7 From UTF-7 To DeflatedSAML From DeflatedSAML Converts the input string into UTF-8 and then encodes the result into a 7-bit ASCII string using Base64 encoding. Decodes the input string from 7-bit ASCII to a UTF-8 string. Applies the URL Encoding Rules to the Input string. Decodes escape characters like %20, %7d using URL Decoding rules. Converts each character of the input string into a hexadecimal representation. Decodes the input string from a hexadecimal representation. Converts the input string into a JavaScript string literal, escaping characters like quotes and backslashes. Decodes a JavaScript string literal. Converts characters like < and > into HTML entities. Decodes HTML Entities. For instance < becomes Fiddler Options, and ensures that the HTML view displays correctly. To JS string operation. Converts the input string using HTML Encoding rules. Converts the input string using HTML Decoding rules. 3. Choose Rules > Customize Rules. 4. Inside the OnBeforeRequest handler, add a new line of code: if (oSession.HostnameIs("webserver")) oSession.host = "webserver:80"; 5. On the Kindle, navigate to Requests from the tablet will appear in Fiddler. The requests are forwarded from port 8888 to port 80 where your webserver is running. Responses are sent back through Fiddler to the device, which has no idea that the content originally came from port 80. If you'd like, you can use Fiddler to rewrite or modify the request while changing the port that the request's sent to. For instance, if your web server runs on port 80, you must reconfigure it to listen on a new port. For instance, if your web server runs on port 80, you must reconfigure it to run on port 81. Then, configure Fiddler to listen on port 80, using the box on the Tools > Fiddler Options > Connections tab. Acting as a Reverse Proxy for HTTPS One problem with running as a reverse proxy for HTTPS is that the client never knows that its traffic is really flowing through a proxy server. That means that if the client makes a HTTPS request to Fiddler, it does not establish a CONNECT tunnel first; it instead directly requests the secure resource. To do so, you must reconfigure both your web server and Fiddler. First, reconfigure your web server to listen on a new port. For instance, if your web server runs on port 80, you must reconfigure it to run on port 81. Then, configure Fiddler to listen on port 80, using the box on the Tools > Fiddler Options > Connections tab. Acting as a Reverse Proxy for HTTPS One problem with running as a reverse proxy for HTTPS is that the client never knows that its traffic is really flowing through a proxy server. That means that if the client makes a HTTPS request to Fiddler, it does not establish a CONNECT tunnel first; it instead directly requests the secure resource. To do so, you must reconfigure both your web server and Fiddler. First, reconfigure your web server to listen on a new port. Second, add a command-line parameter to Fiddler's OnBeforeRequest handler may route the inbound connection to the actual server. Chaining to Upstream Proxy Servers By default, Fiddler will adopt the current system's proxy settings and use those settings as the default upstream gateway proxy for all outbound requests. 89 | Configuring Fiddler and Clients Fiddler supports all types of proxy settings, including manually specified proxies, proxy scripts, and automatically detected (WPAD) proxies. Any

proxy bypass list will also be respected. Your system's default proxy settings (see Internet Explorer's Tools > Internet Options > Connections > LAN Settings screen) are used as the upstream gateway by default. That's true even if you don't normally use Internet Explorer for anything else, or you rely upon a dialup modem or VPN connection that has a different default proxy. If you wish to prevent Fiddler from automatically chaining to your system's default proxy, untick the Chain to upstream gateway proxy option inside Tools > Fiddler Options > Connections. If required, you may use FiddlerScript to override the default gateway proxy for any Session. To do so, set the X-OverrideGateway flag. If you set the flag's value to DIRECT then Fiddler will bypass the gateway and send the request directly to the target server. If you set the value to an address:port combination like myproxyserver:80, Fiddler will instead use the specified proxy for that Session. Chaining to SOCKS / TOR By default, Fiddler expects a proxy specified by the X-OverrideGateway flag to use the CERN protocol used by nearly all proxies. However, one less popular proxy standard exists, called SOCKS. In the SOCKS protocol, the client sends the proxy a binary preface which specifies the target address to which a TCP/IP connection should be made. After the SOCKS confirms that the requested connection has been generated, the client sends its web traffic through the tunnel. The SOCKS protocol is occasionally used by corporate Virtual Private Networking (VPN) software, but it is also used to make connections into the TOR project. The TOR project can be thought of a global proxy network designed to provide anonymity for users of the project. TOR network requests are bounced between worldwide endpoints in an attempt to prevent network monitors from determining the origin of any given request. SOCKS protocol version 4a also allows the proxy server to perform DNS lookups, enhancing privacy. When connected to TOR using SOCKS v4a, DNS resolution happens on the cloud rather than your local computer. Beyond enhancing privacy, one other interesting aspect of the TOR network is that it allows you to experience your site as if you were coming from a different location. For instance, when using TOR I visited a page with advertising. The ads were in Dutch because my request exited the TOR network in Amsterdam. When setting the X-OverrideGateway flag, use the socks= prefix to indicate that Fiddler should use the SOCKS v4a protocol when speaking to the upstream server. For instance, the TOR installer sets up an entry point to the TOR network using a SOCKS proxy on port 8118 called Polipo. You can add the following inside your FiddlerScript's OnBeforeRequest method to route any request to test.example.com through the TOR network: 90 | Configuring Fiddler and Clients if (oSession.Hostname!s("test.example.com")) { oSession["X-OverrideGateway"] = "127.0.0.1:8111"; } If you'd instead prefer to send all of your web traffic via TOR, you can simply set the X-OverrideGateway flag unconditionally for each Session. VPNs, Modems, and Tethering When you establish a VPN, 3G tethering, or dial-up modem connection to a network, Windows, WinINET will use that connection's proxy settings for all requests. To ensure that such traffic is captured by Fiddler, ensure that the Monitor all connections box is checked inside Tools > Fiddler Options > Connections. Some uncommon network software products operate outside of the WinINET layer. When connected using such software, your web traffic may not be visible to Fiddler because it is being captured and routed using mechanisms beyond WinINET's control. DirectAccess Recent versions of Windows support a technology known as DirectAccess (that permits remote access to a corporate network without establishing a VPN). DirectAccess is integrated into WinINET so that requests which flow over DirectAccess are never sent to the system default proxy. Unfortunately, this behavior means that Fiddler cannot observe traffic when DirectAccess is in use. The Windows DirectAccess configuration entries can be updated in the registry to specify a per-target proxy server, but debugging via this mechanism would prove very cumbersome. Instead, when using Fiddler in such an environment, engineers typically use Remote Desktop to access a desktop PC running inside the corporate network, then run Fiddler and the client application on that PC. 91 | Configuring Fiddler and Clients MEMORY USAGE AND FIDDLER'S BITNESS Fiddler stores the entire request and response in memory; this means that Fiddler can use large amounts of memory (RAM) while it is running. The Operating System's memory manager is designed to manage the use of memory by applications and ensures that even if Fiddler is using a large amount of memory, rarely-accessed objects are swapped out of system RAM into the disk-based pagefile. However, even with lots of RAM and disk space, Fiddler may occasionally show a warning message: Exception of type 'System.OutOfMemoryException' was thrown. at System.IO.MemoryStream.set_Capacity(Int32 value) at System.IO.MemoryStream.EnsureCapacity(Int32 value) at System.IO.MemoryStream.Write(Byte[] buffer, Int32 offset, Int32 count) at Fiddler.Session.Execute(Object objThreadstate) This means a seemingly simple operation isn't really out of memory—instead, this means that the memory manager could not find a contiguous block of address space large enough to store the request or response. This problem most often occurs when downloading large files (e.g. a video over a hundred megabytes in size). No matter how much RAM you have, a 32bit process is limited to an address space of 2 gigabytes in size. Each object in that address space can cause "fragmentation" that could prevent storage of large objects by splitting available memory into contiguous block. On a 32bit computer, Fiddler may occasionally show a warning message: Exception of type 'System.OutOfMemoryException' was thrown. at a mere 2 megabytes from being stored. You can reduce the incidence of this problem by clearing Fiddler's Web Sessions list periodically. Alternatively, use the Keep Only box on the toolbar to automatically trim the Web Sessions list to a fixed number of sessions to free up space. Out-of-memory errors are rarely encountered when Fiddler runs on a 64bit version of Windows because the 64bit address space is so large that you cannot possibly fill it or fragment it enough to prevent storage of even giant sessions. However, even on 64bit computers, each individual request and response is limited to 2 gigabytes due to an underlying limit within the .NET Framework. On a 32-bit machine, you can hold more than 2 gigabytes in RAM. Fiddler 32bit cannot use more than 3 megabytes to stream to the disk. 92 | Configuring Fiddler and Clients // This block enables streaming for files larger than 5mb if (oSession.oResponse.headers.Exists("Content-Length")) { var s.Len = oSession.oResponseHeaders["Content-Length"]; if (iLen > 5000000) { oSession.bBufferResponse = false; oSession["ui-color"] = "yellow"; oSession["log-drop-response-body"] = "save memory"; } } } If you're building Fiddler.Core or writing a Fiddler Extension, you can use similar logic: FiddlerApplication.ResponseHeadersAvailable += delegate(Fiddler.Session oS) { // This block enables streaming for files larger than 5mb if (oS.responseHeaders.Exists("Content-Length")) { int iLen = 0; if (int.TryParse(oS.oResponse["Content-Length"], out iLen)) { // File larger than 5mb? Don't save its content if (iLen > 5000000) { oS.bBufferResponse = false; oS["log-drop-response-body"] = "save memory"; } } } }; By default, Fiddler will always run in 64bit mode when running on a 64bit version of Windows. In rare instances, you may prefer to run Fiddler in 32bit mode. For instance, you must run in 32bit mode when you're using an extension that requires native binaries that are only available as 32bit modules (e.g. Silverlight 4) or your FiddlerScript depends upon other modules unavailable in 64bit. For example, the Microsoft Jet.OLEDB.4.0 database provider used to write to Microsoft Access .MDB database files is not available in 64bit mode. To force Fiddler to run in 32bit mode on 64bit Windows, use the ForceCPU.exe tool available in the Fiddler installation folder. 93 | Configuring Fiddler and Clients BUFFERING VS. STREAMING TRAFFIC In order to enable modification of requests and responses, Fiddler is configured to completely buffer requests and response messages before passing them along to their destinations. Request Buffering When a client connects to Fiddler, Fiddler reads the entire HTTP request from the client. If a breakpoint is set, the request is then paused to allow tampering using the Inspectors. After the request is resumed, the server connection is established and Fiddler transmits the entire request to the server. When the request to the server does not expose any mechanism to "stream" a HTTP request to the server as it is read from the client— requests are always buffered completely before sending. HTML5 WebSockets are one exception to this rule— WebSocket messages stream bi-directionally. Response Buffering After sending the request to the server, Fiddler begins reading its response. Some common web scenarios (particularly streaming audio and video) are impacted negatively by response buffering, so Fiddler permits streaming of responses. Streamed responses cannot be modified using Fiddler's Inspectors. By default, only audio and video responses are configured to stream. You can configure all responses to stream by setting the toggle on the Fiddler toolbar, or you may stream selectively on a per-response basis by setting the Session's bBufferResponse property to false using FiddlerScript. When streaming is enabled for a response, each block of data read from the server is immediately passed to the client application. By default, if the client application closes its connection to Fiddler, Fiddler will continue to read the response from the server to permit collection of the entire response. If you'd prefer Fiddler to abort the download if the client disconnects, set the fiddler.network.streaming.AbortIfClientAborts preference to true. When buffering is disabled for a response, the Fiddler Inspectors cannot be used to view or modify its content. When logging is enabled for a response (e.g. logging the response body to a database) to work properly for all Sessions. If your code needs to be notified whenever a given response was streamed, it can test the Session's BitFlags: bool bWasStreamed = oSession.isFlagSet(SessionFlags.ResponseStreamed); If you need to modify the response body before it is sent to the client, you must disable streaming by setting oSession.bBufferResponse=true in either the OnBeforeRequest or OnPeekAtResponseHeaders handlers. COMET Fiddler's COMET Peek feature can't show the response's body after the response has been completely read from the server. But what happens if the server's response never ends? Such responses occur with streaming radio stations and on sites that use a web programming technique called COMET. 94 | Configuring Fiddler and Clients When COMET, a server uses a "hanging frame" or other mechanism to push data to the client using a long-held HTTP connection that trickles data to the client as needed. HTML5 introduces a similar mechanism called "Server Sent Events" which works using the same general mechanism. Because a server's COMET response never really "finishes," the data returned on that connection is not normally visible to Fiddler's Inspectors until the connection closes. To enable such data to be inspected, Fiddler offers the COMET Peek command on the Web Sessions' context menu. When you invoke that command, Fiddler will take a "snapshot" of the in-progress response, allowing you to inspect the partial response read from the server. Default behavior of buffering responses wreaks havoc with sites that depend on COMET. That's because large amounts of COMET data can usually cause the web application to stall. Unfortunately, COMET responses aren't marked in any way, so you will need to manually adjust responses from buffering if you do not wish to enable the Streaming option globally. 95 | Configuring Fiddler and Clients HTML5 WEBSOCK TS The HTML5 specification introduces WebSockets, a technology that enables real-time socket communication between a client and server. To create a WebSocket, the client first establishes a HTTP(S) connection to the server. Next, the client and server handshake and agree to use the WebSocket protocol for subsequent traffic on the connection. The ws:// and wss:// URI schemes are used for plaintext and secure WebSockets respectively, even though the initial handshakes take place over HTTP or HTTPS. To help ensure that WebSocket traffic flows unimpeded through an proxy (like Fiddler), clients will first establish a CONNECT to the proxy, requesting a tunnel to the destination server. If the WebSocket URI uses the wss:// scheme, a HTTPS handshake then occurs. If the unsecure ws:// scheme was used, then the HTTPS handshake is skipped. The client then sends the newly-established connection to send a HTTP GET request with an Upgrade header proposing a switch for the WebSocket protocol. If the server agrees to change protocols, it sends a HTTP/1.1 101 Switching Protocols response. You can easily observe this process using the Web Sessions list and Inspectors. After the handshake, the client and server may send each other WebSocket messages in any order. Unlike HTTP's request-response pattern, the server may send WebSocket messages to the client without the client first sending a WebSocket message to the server. Because HTTP messages are no longer exchanged on a connection which has been upgraded to the WebSocket protocol, you will not see the WebSocket messages in the Web Sessions list or the requests of these messages are passed via Inspectors. However, Fiddler does have the ability to parse WebSocket messages, and in future versions of Fiddler, I hope use of Inspector will be introduced to display and modification of WebSocket traffic. For now, WebSocket messages 96 | Configuring Fiddler and Clients these messages are parsed and displayed in plaintext: Upgrading Session #47 to websocket [WebSocket #47] Server->Client (12 bytes) TYPE: TEXT. MESSAGE: connected, FLAGS: 10000001 DATA: 10 bytes. ------------------------[WebSocket #47] Client->Server (14 bytes) TYPE: TEXT. MESSAGE: timer, FLAGS: 10000001 DATA: 6 bytes, masked using KEY: 3D-36-84-D5. ------------------------[WebSocket #47] Server->Client (23 bytes) TYPE: TEXT. MESSAGE: time.2012/4/30 14:50:38 FLAGS: 10000000 DATA: 23 bytes. ------------------------[WebSocket #47] Client->Server (8 bytes) TYPE: TEXT. CLOSE. CLOSE REASON: 1001 FLAGS: 10001000 DATA: 2 bytes, masked using KEY: 25-5A-56-AE. ------------------------To help prevent attacks against transparent proxies or other buggy but well-meaning intermediaries, a client's WebSocket messages are typically "masked" by XOR'ing the message text against a masking key that is not visible to JavaScript running on the client. When logging the client's WebSockets messages, Fiddler automatically unmasks the text for ease of reading. As the standard features and browsers implement the specification, more websites will begin building upon WebSockets. As that happens, Fiddler will be updated to better support this exciting new protocol. 97 | Configuring Fiddler and Clients FIDDLE R AND HTTPS When visiting a HTTPS site in Fiddler, there's not a lot to see by default. Instead of the expected list of requests and responses, you'll only see one or more CONNECT tunnels. The HTTPS protocol sandwiches an encrypted (SSL or TLS) connection between HTTP requests and the underlying TCP/IP network connection upon which most HTTPS traffic is configured appropriately. Fiddler achieves this by using a "man-in-the-middle" approach to HTTPS, which means that when talking to the client, it pretends to be the server, and when talking to the server, it pretends to be the client. The HTTPS protocol is explicitly designed to block this attack by using digital certificates to authenticate the identity of a HTTPS server (and optionally the client). When a client receives a certificate from a server, it validates that the certificate itself is trustworthy by determining whether it is chained to a Root Certification Authority that is trusted by the client or the operating system. Since you are typically running Fiddler on your own computer, you can reconfigure your browser or operating system to trust Fiddler's root certificate. After you do so, the client application will not complain when it detects that traffic is being protected by Fiddler-generated certificates. To enable HTTPS traffic decryption in Fiddler, click Tools > Fiddler Options. On the HTTPS tab, tick the box labeled Decrypt HTTPS traffic. When you enable HTTPS decryption, Fiddler will generate a self-signed root certificate and a matching Private Key. Fiddler will use this root certificate to generate HTTPS server certificates (also called "End Entity" certificates) for each secure site that you visit. 98 | Configuring Fiddler and Clients Trusting the Fiddler Root Certificate After generating the root certificate, Fiddler gives you the opportunity to add it to Windows' Trusted Root Certificate Authorities store. Adding the root to the Trusted Store will allow the HTTPS server certificates Fiddler later generates to be deemed valid by browsers and other applications. This will help stop browsers from showing warning screens and prevent applications from failing to connect due to "trust errors." Fiddler and Windows show prompts to warn off users who might not understand the implications of trusting a certificate: If you click Yes, Windows will prompt you to confirm the change: These warnings are deliberately verbose and scary, but the actual level of risk is minimal. Each Fiddler root certificate is generated uniquely per-computer, which improves security by ensuring that no other Fiddler user has the same root certificate. As such, the root certificate could only really be abused by malware running on the local computer, and if your computer is already infected by malware, you have bigger problems to worry about. Machine-wide Trust on Windows 8 On Windows 8, Metro-style applications will not trust Fiddler's root certificate unless it is in the per-Machine Trusted Root certificate store. Therefore, after the prior steps add the certificate to your per-User Trusted Root certificate store, Fiddler will then launch an administrative program to add the certificate to the Machine store: 99 | Configuring Fiddler and Clients If you click Yes to launch the utility, it will confirm the operation: After you click through this flurry of prompts, Fiddler's root certificate will be installed and applications that rely on the Windows certificate store will no longer present security errors when Fiddler is decrypting their traffic. To later remove all Fiddler-generated certificates, untick the Decrypt HTTPS traffic checkbox and then press the Remove Interception Certificates button. Manually Trusting the Fiddler Root If you'd prefer to manually trust the FiddlerRoot, launch certmgr.msc and drag the DO_NOT_TRUST_FIDDLERROOT certificate from the Personal folder to the Trusted Root Certification Authorities folder. If you wish to make this change on a machine-wide basis: 1. Right-click the DO_NOT_TRUST_FIDDLERROOT certificate from the Personal folder and choose All Tasks > Export. 2. Export the certificate as DER Encoded X509 Binary to FiddlerRoot.cer on your desktop. 3. Launch mmc.exe. 4. Click File > Add/Remove Snap-In. 5. Select the Certificates snap-in and click Add. 6. When prompted This snap-in will always manage certificates for: choose Computer Account 7. Click Local Computer, then Finish, then Ok. 8. Open the Certificates (Local Computer) node. 9. Right-click the Trusted Root Certificate Authorities folder and choose All Tasks > Import. 10. Choose the file you exported in step #2 and import it. 100 | Configuring Fiddler and Clients ADDITIONAL HTTPS Options Before closing the Fiddler Options window, consider using the dropdown to configure which protocols should have their traffic decrypted: If you only plan to decrypt browser traffic, choose that option to avoid decrypting traffic from other applications that are not using HTTPS in standard ways (e.g. Outlook uses RPC-over-HTTPS tunnels to connect to Exchange Server) or that will not respect the Fiddler root certificates' presence in the Windows Trusted store (e.g. Dropbox). You may also use the textbox to list servers for which HTTPS traffic should not be decrypted. For instance, I use the following settings to prevent decryption of Outlook Web Access and Dropbox traffic: Use semicolons to delimit the hostnames in the list, and use * as a wildcard character. 101 | Configuring Fiddler and Clients CONFIGURING CLIENTS FOR HTTPS DECRYPTION While most applications (Internet Explorer, Microsoft Office, Chrome, Safari, etc.) use the Windows Certificate Store to validate certificate chains, some applications maintain their own certificate stores. For instance, Java Runtime Environments often have their own certificate stores, and the Firefox and Opera browsers also maintain their own certificate lists. To configure such clients to trust Fiddler-generated certificates, you must first obtain Fiddler's root certificate as a .CER file. You have two options to do so: 1> Click the Export Root Certificate to Desktop button on the FiddlerOptions window's HTTPS tab. 2> Visit in the browser and click the FiddlerRoot Certificate link to download the certificate as a .CER file. After you have the root certificate as a .CER file, you can add it to the client application's Trusted certificates list. Browsers Firefox In Firefox, click Tools > Options. Click the Advanced button, and switch to the Encryption tab. Click View Certificates to open the Certificate Manager. Click the Authorities tab and click the Import button. Select the FiddlerRoot.cer file and click Open. Tick the Trust this CA to identify websites checkbox and press Ok. Firefox will now trust HTTPS server certificates generated by Fiddler. Opera In Opera, click Opera > Settings > Preferences. Click the Advanced tab, and click Security in the list. Click Manage Certificates. Click the Authorities tab and click the Import... button. Select the FiddlerRoot.cer file and click Open. Tick the Trust button and click the Ok button that you want to trust the certificate. Opera will now trust HTTPS server certificates generated by Fiddler. Cross-machine scenarios If you have configured Fiddler to proxy HTTPS traffic from other computers, those other computers must be manually configured to trust the root certificate from the Fiddler server machine. You can enable this trust by downloading the root certificate from the page, then using the manual certmgr.msc steps to trust that root certificate. One important reminder: Each Fiddler root certificate is generated uniquely per-machine. If a client PC has previously generated a Fiddler root certificate, that root certificate will interfere when proxying traffic through Fiddler running on a different machine. That's because the root certificates will not match (they'll have the same Subject CN but different public keys). This mismatch will cause fatal errors when establishing HTTPS connections— even if the client 102 | Configuring Fiddler and Clients has been configured to trust both root certificates. To avoid this problem, you must first remove all existing Fiddler root certificates from the client before trusting the new root certificate downloaded from the Fiddler server machine. HTTPS and Devices Windows Phone In order for a Windows Phone device to trust Fiddler's root certificate, you must install the certificate onto the device. To do so, open from the device, and download the root certificate. Tap to open the FiddlerRoot.cer file, and when prompted by the Install certificate? screen, click the Install button. Android and iOS Fiddler's default Certificate Maker is based on the makecert.exe command-line tool. The certificates it generates are compatible with most devices, but there are several limitations when acting as a HTTP-to-FTP gateway. For instance, iOS devices including the iPad, iPhone, and iPod require that the root and server certificates contain additional metadata not included in makecert.exe-generated certificates. Some Android distributions have similar requirements. To enable Fiddler to generate certificates compatible with these devices, download the Certificate Maker plugin for Fiddler from This plugin requires Windows Vista or later. It replaces the default certificate generation code in Fiddler with a version based on the open-source Bouncy Castle cryptography library. The plugin generates iOS-compatible certificates by default and respects several Preferences to enable compatibility with a wide-variety of platforms. For instance, at least one Android platform requires that server certificates must not have the Critical constraint applied to their EKU. You can configure the Certificate Maker plugin to remove that constraint by using QuickExec to set the preference: prefs set fiddler.certmaker.bc.ee.criticalenku false The server certificates that generated by this plugin are not placed in the Windows Certificate store and are instead kept only in memory only. Every time Fiddler restarts, new server certificates will be generated. In contrast, makecert.exe-generated certificates are individually added to the Windows Certificate's root section of the device. 103 | Configuring Fiddler and Clients REMOVING FIDDLER'S CERTIFICATES If your server certificates are as they are generated, and are only removed when you push the Remove Interception Certificates button. After configuring Fiddler and Clients Tap the Install button. You'll then see a warning, which you may acknowledge by pressing the Install button. After Fiddler's root certificate is installed, your device's browser will no longer complain about certificate errors when Fiddler is decrypting their traffic. If you later decide to uninstall the root certificate from the device, open the Settings app, click General, and scroll down to Profiles at the bottom. Select the DO_NOT_TRUST_FiddlerRoot profile, and tap Remove. Buggy HTTPS Servers A small number of HTTPS servers implement the TLS protocol handshake incorrectly and will fail when Fiddler handshakes with them. The same connection failure will occur when a web browser connects to the server, but that failure may be silently handled by the browser automatically falling back to use an older version of the handshake protocol. Fiddler can be configured to offer older, less-secure security protocol implementations. From FiddlerScript, you can add the following code in the Main() function to limit Fiddler to offering only SSLv3 when making HTTPS connections: CONFIG.oAcceptedServerHTTPSProtocols = System.Security.Authentication.SslProtocols.Ssl3; Alternatively, you can limit the protocols on a per-request basis. Inside the OnBeforeRequest function, add code like so: 104 | Configuring Fiddler and Clients if (oSession.HostnameIs("buggy.example.com")) { oSession["X-OverrideSSLProtocols"] = "ssl3.0"; } By offering only SSLv3, you can accommodate servers that do not properly support extensions or other features of the TLS protocol. Certificate Validation When Fiddler connects to a HTTPS server, it will validate the server's certificate to ensure that the certificate is valid and contains the target site's hostname. By default, any certificate errors are presented for you to decide how to proceed. If you choose No, the connection will be dropped. If you choose Yes, the certificate error will be ignored and the connection will be used normally. Fiddler will cache your decision until restarted. Because Fiddler generates its own certificate to secure the connection from the client, the client application will be unaware that the server's certificate was invalid. Certificate Pinning A very small number of HTTPS client applications support a feature known as "Certificate Pinning" whereby the client application is hardcoded to accept only one specific certificate. Even if the connection uses a certificate that chains to a root that is otherwise fully-trusted by the operating system, such applications will refuse to accept an unexpected certificate. To date, some Twitter and Dropbox apps include this functionality. In most cases, when a Certificate-Pinned application performs a HTTPS handshake through a CONNECT tunnel to Fiddler, it will examine the response's certificate and refuse to send any further requests when it discovers the Fiddler-generated certificate. 105 | Configuring Fiddler and Clients Unfortunately, there is no general-purpose workaround to resolve this; the best you can do is to exempt that application's traffic from decryption by setting the x-no-decrypt Session flag on the CONNECT tunnel. This flag will prevent Fiddler from decrypting the traffic in the tunnel and it will flow through Fiddler uninterrupted. 106 | Configuring Fiddler and Clients FIDDLE R AND FTP Like HTTP, the File Transfer Protocol (FTP) is a TCP/IP-based protocol used for transferring files, but it predates and is not compatible with HTTP. However, the protocol means the sole exception to the rule that Fiddler can only view HTTP-based protocols. That's because when a client browser or application is configured to proxy FTP traffic using a CERN-style proxy (like Fiddler), the client will convert FTP download requests into HTTP GET requests before sending them to the proxy. Only the HTTP request's URL and Authorization header are used in the FTP request; all other request headers are ignored. There are several limitations when acting as a HTTP-to-FTP gateway. For instance, the Session's Timers will not be set. Streaming of FTP responses is not supported— the complete FTP response will be buffered before being converted and returned to the client. 107 | Configuring Fiddler and Clients FIDDLE R AND WEB AUTHENTICATION Most public websites use HTML Forms for authentication—they prompt the client for a username and password. If the credentials supplied are valid, the client gets a login cookie that is set for subsequent requests. Fiddler can easily view and submit such cookies. However, HTTP and HTTPS offer two native mechanisms for authenticating a client: HTTP Authentication HTTPS Client Certificates Either can be used with servers using either form of authentication. HTTP Authentication With HTTP Authentication, the client makes a HTTP request and the server responds with a HTTP/401 or HTTP/407 response demanding credentials. HTTP/401s are sent by servers that require authentication, while HTTP/407s are sent by proxy servers. There are four common authentication schemes in use in HTTP: Basic Digest NTLM Negotiate In Basic authentication (RFC 2617) the client supplies the username and password in the Authorization request header. The client's credentials are base64-encoded and can be trivially decoded using the Auth Inspector. The Basic authentication scheme is primitive and obviously unsafe to use over unencrypted HTTP connections. In Digest authentication, also described in RFC 2617, the server presents a challenge to the client using the WWWAuthenticate response header. The client combines the challenge data and its knowledge of the user's password to compute a message digest that proves to the server that it knows the password. This authentication scheme is stronger than Basic but is rather uncommon, owing to limited server support and various client bugs. The NTLM authentication scheme is rarely used on the public Internet but is commonly used on Windows-based Intranets. When the server presents a HTTP/401 to the client, the client reissues the request indicating that it supports NTLM, then the server provides a challenge. The client uses the challenge and the credentials to generate a reply to the challenge, and the server returns the resource if the challenge was answered successfully. This pattern means that when NTLM authentication occurs, the client receives two HTTP/401 challenges before the resource is finally returned with a HTTP/200. 108 | Configuring Fiddler and Clients Unlike Basic and Digest, the NTLM scheme is not generally per-request. Instead NTLM is per-connection, meaning that a single authentication proof will be used to authorize all subsequent requests on the same connection. That's why you'll often see a flurry of HTTP/401s when first loading an Intranet site, then don't see any subsequent authentication challenges on later page loads. The Negotiate authentication scheme (sometimes called integrated Windows Authentication) is a "wrapper" protocol that uses either the NTLM or Kerberos protocol under the covers. At the HTTP layer, Negotiate generally behaves very much like NTLM, requiring between one and three roundtrips per request. When NTLM or Negotiate is used through a proxy server, client applications require that the proxy add a ProxySupport: Session-Based-Authentication response header. This header indicates that the proxy understands that authentication must not be shared between different clients. Fiddler adds this header automatically. Automatic Authentication in Fiddler Fiddler is able to automatically authenticate to servers that use the Digest, NTLM and Negotiate protocols. When Fiddler automatically authenticates, the client that issued the request will not see the interim HTTP/401 or HTTP/407 responses. That's because Fiddler itself consumes these responses, answers the server's challenges, and returns data to the client only after authentication is complete. To have Fiddler automatically respond to a server or gateway using its X-AutoAuth property on the Session to a plaintext credential string. To specify a value of (default), then Fiddler will use the credentials of the Windows logon user account in which Fiddler runs: 109 | Configuring Fiddler and Clients an Authorization or Proxy- Authorization response header. That header can be added to outbound requests using the Filters tab, FiddlerScript, or the Composer tab's Request Headers box. Authenticate if running. Three common Authentication issues relate to Channel-Binding Tokens, WinHTTP credential release, and Loopback authentication protections. Each will be described in turn. Channel-Binding One shortcoming of the NTLM authentication scheme is that it can fall victim to "pass-through" attacks. In such an attack, the client is lured to a malicious site which presents an authentication challenge that was originally generated by a victim server. The client dutifully responds to the authentication challenge, and the malicious uses that client's challenge-response to authenticate to the victim server using the client's credentials. To address this problem, the concept of Channel-Binding was introduced. Channel-Binding binds the authentication challenge-response to the underlying connection such that it cannot be reused on another connection. Typically this works by binding the credentials to the current HTTPS connection. Channel-Binding is enabled via the 110 | Configuring Fiddler and Clients Channel-Binding presents a problem for Fiddler because the client binds its credentials to its connection to Fiddler, such that when Fiddler forwards those credentials to the actual server, the server rejects them. To resolve this problem, you can configure Fiddler to itself authenticate to the server, taking the client out of the loop. Because Fiddler itself generates the challenge-response, using the Channel-Binding information matching its own connection to the server, the credentials will be accepted. To configure Fiddler to authenticate on the client's behalf, use the X-AutoAuth flag described in the "Automatic Authentication in Fiddler" section above. WinHTTP Credential Release Policy The WinHTTP network stack does not support the concept of security zones, which means that it sometimes will refuse to respond to authentication challenges when a proxy like Fiddler is involved. This problem sometimes appears when attempting to download documents from SharePoint sites using the Microsoft Office client applications. You can resolve this problem by configuring WinHTTP with a remote server but it is really authenticating back to the local computer. Such attacks usually attempt to elevate privilege from a low-privilege process to a higher-privilege local process. When Windows encounters an unexpected attempt to authenticate to the local machine, the authentication attempt is blocked. To disable loopback protection, set the flag DisableLoopbackCheck=1 as described at: com/kb/926642. 111 | Configuring Fiddler and Clients HTTPS Client Certificates In addition to HTTPS server authentication, the client supplies a certificate to the server that cryptographically proves the identity of the user. One of the key design goals of Client Certificate authentication is to prevent network intermediaries (like Fiddler) from abusing the client's credentials. Even if the client application sends its certificate to Fiddler, Fiddler cannot successfully reuse that certificate to respond to the server's demand, because the client never provides Fiddler with its private key. To resolve this limitation, you can supply any necessary client certificates and private keys directly for Fiddler to use when handshaking with the server. By default, if a server prompts the client for a certificate, Fiddler will look inside the %USERPROFILE%\Documents\Fiddler2\ folder for a file named ClientCertificate.cer and will use that certificate when responding to the server's certificate demand. In some cases, you may want to use a different client certificate for each secure connection. To do so, specify the https-Client-Certificate property on the CONNECT tunnel to the secure server. You can write code like this: static function OnBeforeRequest(oSession: Session) { if (oSession.HTTPMethodIs("CONNECT")) { if (oSession.HostnameIs("exampleA")) { oSession["https-Client-Certificate"] = "C:\\certs\\CertA.cer"; } else if (oSession.HostnameIs("exampleB")) { oSession["https-Client-Certificate"] = "C:\\test\\CertB.cer"; } } } ... A .CER file does not contain the private key associated with the certificate's public key. Instead, the .CER file merely acts a reference to Windows' Personal certificates store (certmgr.msc). The Windows certificate store holds the private key associated with the certificate and releases it only as needed. Client certificates stored on a Smartcard will automatically appear within the Personal store when the Smartcard is inserted: 111 | Configuring Fiddler and Clients When the Smartcard is inserted, you may export a .CER file from certmgr.msc and use it just like any other client certificate. Note that your Smartcard must remain inserted for Fiddler to use its private key. If the desired certificate isn't yet installed Windows' Personal certificates store (e.g. you only have a .pfx file) you must first import it into the certificate store and choose All Tasks > Export.... Save the .CER file to either the default ClientCertificate.cer location or the location you will specify via the https-Client-Certificate flag. 112 | Configuring Fiddler and Clients OVERVIEW Fiddler's Inspectors are used to display the request and response for the selected Session in the Web Sessions list. Inspectors appear in two places: On the Inspectors tab in the main Fiddler window, and in the standalone Inspect Session windows. The latter are opened using the command on the Web Sessions context menu. On the Inspectors tab, the Request Inspectors are shown in a panel at the top, and the Response Inspectors are shown at the bottom. You can use Inspectors by clicking on the desired Inspector's name (e.g. HexView). When you first double-click or press the Enter key on a session in the Web Sessions list, the Inspectors tab will become active. Each request or response will be polled to ask if the applicable it is inspecting the selected request and response. For instance, the ImageView Inspector will return a low score for textual types. In contrast, the TextView Inspector returns a high score for textual types and a low score for binary formats. Whichever request returns the highest score for the Session will become active. To force Fiddler to always activate a particular Request Inspector, set the fiddler.ui.inspectors.request.alwaysuse preference to the title of the Inspector's tab. A thin blue line splits the top and bottom panels; you can use that move to move the line and resize the panels. Double-clicking the line will maximize the Response Inspectors panel, as this is the panel most users use most of the time. 114 | Inspectors To get even more display area, the Inspectors tab can be extracted from the main window, either by clicking the TearOff button in the toolbar or by typing the command tearoff in the QuickExec box. Closing the Inspectors window will return them to their default position on the main window. Right-clicking on an individual request or response inspector's tab will show a menu containing two options: Inspect Properties and Hide Inspector. The first shows information provided by the Inspector about itself, while the second will remove the inspector from the Inspectors list. To restore a hidden Inspector later, edit the fiddler.inspectors.hidden Preference and restart, or hold the SHIFT key while starting Fiddler. By default, the Inspectors are ReadOnly when inspecting a session, unless the session is currently paused at a breakpoint. Additionally, the Inspectors will permit editing if the session is in an Unlocked state, which can be achieved by using the command on the Edit menu. Most Inspectors will display with a specific background color while in ReadOnly mode, and a different color (typically white) when editing is permitted. The default ReadOnly color is light-blue, and can be changed using the Tools > Fiddler Options > Appearance tab. The remainder of this chapter will describe each of the Inspectors included with Fiddler. 115 | Inspectors AUTH Type Allows Editing Request & Response No Fiddler's Auth Inspector interprets the contents of the Authorization- and Authentication- related headers on requests and responses. Typically, these headers can be found on HTTP/401 and HTTP/407 responses that demand credentials, and the subsequent requests that provide the requested credentials. For instance, if you enable the "Require Proxy Authentication" rule in the Rules menu, Fiddler will return a HTTP/407 response demanding that the user supply credentials for each request. This demand is made by the Proxy-Authenticate header, which the Auth Response Inspector displays: After the user enters the UserName and the password requested by a Proxy-Authorization header with the user's credentials. Since the authentication scheme in use in this scenario is HTTP Basic, the credentials are encoded using base64 encoding. The Auth Request Inspector decodes the string automatically and displays it in plaintext: Of course, most websites will use a stronger form of authentication. HTTP Digest or NTLM or Negotiate schemes. The latter are especially popular on Windows networks. Fiddler' Auth Inspector knows how to parse NTLM blobs and will show you the information contained within those blobs, like so: [NTLM Type3 Authentication]------------------------Provider: NTLMSSP Type: 3 OS Version: 6.2.8329 Flags: 0xa2888205 Unicode supported in security buffer. Request supports security signing. NTLM authentication. Negotiate Always Sign. Negotiate NTLM2 Key. Target Information block provided for use in calculation of the NTLMv2 response. Supports 128-bit encryption. Supports 56-bit encryption. 116 | Inspectors Interop. Offset: 134; Inresp. Length: 24; Inresp. Length2: 24 Atresp Offset: 158; Atresp Length: 396; Atresp Length2: 396 Domain Offset: 88; Domain Length: 14; Domain Offset2: 88; Domain Length2: 14 User Offset: 102; User Length: 14; User Length2: 14 Host Offset: 116; Host Length: 18 msg len: 554 Domain: REDMOND User: ericlaw Host: ERICLAWTB lm resp: 00-00-00-00-00-00... Ntlm resp: 42-40-AE-AB-3E-D7-7F-02-B5-F9-79-5E-1F-53... The Auth Inspector cannot parse Kerberos authentication messages, but will show the decoded bytes of the challenge and response. 117 | Inspectors CACHING Type Allows Editing Response only No The Caching Response Inspector examines the HTTP response headers to determine whether the selected response is cacheable under the rules of HTTP, and if so, for how long. It consults the response headers Cache-Control, Expires, Pragma, Vary, ETag, Age, and Last-Modified in its evaluation of the response. Some browsers (including Internet Explorer) support the specification of cache information in HTML documents using META HTTP-EQUIV tags. The Caching Inspector scans responses that have a HTML Content-Type and will display any HTTP-EQUIV or PRAGMA directives found in the markup. For responses which do not explicitly specify their cachability or freshness lifetime, the Caching Inspector will attempt to calculate using the Last-Modified header and the algorithm suggested in RFC2616. You can learn more about how browsers utilize caches at: 118 | Inspectors COOKIES Type Allows Editing Request & Response No The Cookies Inspector displays the contents of any outbound Cookie and Cookie2 request headers and any inbound Set-Cookie, and P3P response headers. Note that the Cookie2 and Set-Cookie2 headers are uncommonly used and not supported by Internet Explorer and some other browsers. The display of the cookies is basic (you can see the same information in the Headers inspector). This Inspector's value primarily consists of examining the P3P response header, if present, to determine whether the cookie is likely to be stored. P3P (Platform for Privacy Preferences) is a standard whereby the server can communicate to the client how it will be using the cookies that accompany the P3P header. Set-Cookie: ASPSESSIONIDCCBTDCRD=CIFKIKJDFMJFFODAJPFMFKGN; path=/ P3P: CP="ALL IND DSP COR ADM CONo CUR CUSo IVAo IVDo PSA PSD TAI TELo OUR SAMo CNT COM INT NAV ONL PHY PRE PUR UNI" The tokens in the P3P header's string are interpreted by the Inspector and their meaning is listed. The Inspector will further explain the string to determine whether the cookie is deemed "acceptable" by the default privacy policy used by Internet Explorer. It's worth mentioning that some sites (Facebook and Google, as of this writing) send invalid P3P statements to circumvent the browser's privacy features. Under the rules of P3P, unknown tokens (like those sent by such sites) are deliberately ignored. The Cookies Inspector displays the size of the cookie header, helping identify over-sized cookies. In some cases, for instance, Internet Explorer will refuse to set a cookie for any server whose hostname containing an underscore character, and the Inspector will warn you if it encounters this condition. 119 | Inspectors HEADERS Type Allows Editing Request & Response Yes Every HTTP request begins with plaintext headers that describe what resource or operation is sought by the client. The first line (the "Request Line") of the request contains three values, the HTTP Method (e.g. "GET" or "POST"), the URL path which is being requested (e.g. "/index.htm"), and the HTTP/1.1"). Following the Request Line are one or more lines containing name-value pairs of metadata about the request and the client, such as the User-Agent and Accept-Language. Similarly, every HTTP response begins with plaintext headers that describe the result of the request. The first line (the "Status Line") contains the HTTP version (e.g. "HTTP/1.1"), the response status code (e.g. "200") and the reason text (e.g. "OK"). Following the Status Line are one or more lines containing name-value pairs of metadata about the server and the response. The Headers Inspector allows you to view the HTTP headers of the request and response. The headers are grouped based on their function and then sorted alphabetically by name. Groupings are simply for your ease of reading, and are not sent on the network. For Request headers, the groups include [Cache, Client, Entity, Transport, Cookies/Login, Security, Miscellaneous]. For Response headers, the groups include [Entity, Transport, Cookies/Login, Security, Miscellaneous]. By default, the contents of the Headers Inspector are read-only and cannot be edited. When you select a Web Session paused at a breakpoint, or when you select the Unlock for Editing command on the Edit menu, the Headers Inspector will become editable. While in Edit mode, the background color is the default window color (white), and the contents of the headers may be edited. Click the Raw hyperlink at the top-right of the Inspector to display the plaintext of the headers as they are sent on the network. Click the Header Definitions link to view a help topic describing common HTTP headers and their use. Due to limitations in the Windows treeview control, only the first 260 characters of the header's name and value are shown. To use headers which exceed this length, select the Header Viewer window shown below and press Enter or F2, right-click the header and choose View Header. The Header Viewer window will open in read-only mode and allow you to inspect the full name and value. The length (in characters) of the value is displayed in the title bar of the window. While the Inspector is in ReadOnly mode, any Cookie header is shown as a single line as it is actually sent to the server. 120 | Inspectors Context Menu Right-clicking on the list of Headers will show a menu with the following options: View Header Edit Header Copy Header Copy Value only Send to TextWizard Add Header Remove Header Paste Headers Lookup Header Open the Header Viewer window for the selected header. (Only shown in Edit mode) Open the Header Editor window for the selected header. Copy the selected header's Name to the clipboard. Send the selected header's Value to the TextWizard to permit decoding of encoded text. When in Edit mode, create a new blank header. When in Edit mode, remove the selected header. When in Edit mode, attempt to add new header based on the information on the clipboard. Open a web search engine with selected header's value to the clipboard. To activate the Request and Response Inspectors. Within the Inspector, the following hotkeys are available: CTRL+C=CTRL+SHIFT+C F2 or Enter CTRL+V Insert Delete Copy the selected header to the clipboard. Copy the selected header's value to the clipboard. Open the Header Viewer or Header Editor for the selected header. When in Editing mode, attempt to add one or more new headers based on the text on the clipboard. When in Editing mode, create a new blank header. When in Editing mode, remove the selected header. Editing When the Inspector is in Edit mode, you can make changes to the headers before they are sent to the server or the client. If you would like to edit the entire set of headers at once, click the Raw link at the top-right of the Inspector. The Raw Headers window will appear and enable you to save the text set of headers. Alternatively, you can directly change the contents of the Request or Status Line by double-clicking the text in the box at the top of the Inspector. To edit an individual name-value header, select it and press F2 or hit Enter and the Header Editor window will appear. 121 | Inspectors You may update the name and value of the new header using the two textboxes, or you may paste Templates button at the bottom of the window to choose from commonly-used headers. 122 | Inspectors HEXVIEW Type Allows Editing Request & Response Yes The HexView Inspector allows you to view the request and response headers and body using a hexadecimal edit box. This capability is most useful when inspecting binary content. The Inspector contains a HexEdit control organized into three vertical columns. In the left column, in light grey, is the hexadecimal address for the adjacent line of bytes. In the center column are the hexadecimal representations of the individual bytes. The right column redisplays those bytes, interpreted as ASCII text. If the HexView is configured to show the headers, the request headers will be shown in blue text and response headers in black. At the bottom of the Inspector is a status bar containing three panels. The first panel shows the current cursor position within the file, and is configured to show the header bytes, the offset will automatically reset to zero when the cursor enters the body. If one or more bytes are selected in the control, the middle panel displays the count of the current byte selection. The right panel indicates the current edit mode: Read only, Overwrite, or Insert. The HexView Inspector's context menu offers the following items: Insert File Here... Select Bytes... Save Selected Bytes... Goto Offset... Find String... Find... In Edit mode, inserts the bytes of a selected file at the current cursor position. Prompts you for a starting offset to select, then selects that number of bytes starting at the current cursor position. Enter a leading $ character if you wish to specify the number of bytes in Hexadecimal format. Tip: This feature can be useful if you wish to examine HTTP Chunked Encoding blocks, as the length of each block is specified as a hexadecimal number. When one or more bytes are selected, this item enables saving the selected bytes to a file of your choosing. Enables you to move the cursor to a selected byte of the content. Enter a leading + or - character to specify the offset from the current cursor position. Enter 0 to continue the search after the HexView automatically enters Edit mode, as the Inspector enters Edit mode. When in Edit mode, you may use standard editing features like Copy and Paste, and you may overwrite or insert new bytes into the content. The HexView Inspector's content menu offers standard Cut, Copy and Paste options and a checkbox to enable or disable word wrapping. The menu also offers an option to send the currently selected text to the TextWizard tool. There are also options to enable or disable word wrapping. The body content is shown in black. At the bottom of the Inspector is a bar that offers additional information and features. The first box shows the current cursor position in Line:Column format. The next box shows the current character: the hexadecimal representation of the current text selection, if any. Next is a search box that allows you to select matching text within the content. Pressing Enter or F3 will select the next match. Pressing CTRL+Enter will highlight all matches in the content. Pressing CTRL+F will activate the search box and select the text area above (to allow you to view search results in context). Matches are selected as you type, and the box will turn green if a match was found or red if no further matches were found. Pressing Enter or F3 will select the next match. Pressing CTRL+Enter will highlight all matches in the content. The View in Notepad button opens the content to a temporary file and opens a text editor to view the file. The text editor launched is controlled by the fiddler.config.path.texteditor preference; notepad.exe is the default. 123 | Inspectors IMAGEVIEW Type Allows Editing Response only No The ImageView Inspector displays any image returned by the server, such as the length of the response file, its Content-Type, and information about how the response may be cached. The Headers Inspector allows you to view the HTTP headers of the request and response. The headers below the image will show information about the currently selected image, including its size in bytes, pixel dimensions, and the format. At the bottom of the grey panel is a dropdown that allows you to control how the image is scaled: No scaling - Images are shown at their native dimensions. Autoshrink - Images that are larger than the display area are scaled down. Scale to fit - Images that are larger or smaller than the display area are scaled down. Scale to fit - Images larger than the display area are scaled down, and images that are smaller than the display area are scaled up to fill the area. The Inspector's context menu allows you to copy the image to the clipboard as a bitmap, or quickly save the image to a file. You can use the context menu's Copy to your Windows desktop using the current time as the basis for the filename. Also available is an option to copy the image as a DataURL, a text format which can be embedded in HTML or stylesheets and displayed in modern browsers (IE8+). If the resulting URI is over 32kb in length, Fiddler will warn you that IE8 does not support 32kb+ URIs; IE9 or later will be required to render the image. The final option on the menu allows you to change the background color (normally light blue) which can provide useful contrast when examining small or transparent images. Middle-clicking on the image will quickly toggle between the two options: Key or Mouse action Enter or F2 H or V Mousewheel up Mousewheel down Zoom In Zoom Out Full-Screen and Actual-Size Flip the image horizontally Flip the image vertically 124 | Inspectors JSON Type Allows Editing Response No The JSON Inspector interprets the selected request or response body as a JavaScript Object Notation (JSON) formatted string, showing a treeview of the JSON object's nodes. If the body cannot be interpreted as JSON, the treeview will remain empty. Unlike most inspectors, the JSON Inspector is able to handle data even if the request or response has HTTP Chunked Encoding applied; you do not need to remove the chunking to display the content. Many responses delivered with the JSON Content-Type do not actually contain JSON. Instead, they're JSONP, a special type of JSON that is a single function call wrapping a string argument containing JSON. JSONP by ignoring the leading function call and trailing parentheses and semicolon. However, some JSONP is malformed such that the name component of the name/value pairs is unquoted; the JSON Inspector cannot handle that type of malformedness and will refuse to parse the content. The context menu offers two options: Copy, to copy the selected node to the clipboard (or press CTRL+C), and Send to TextWizard, to send the selected node's content to the TextWizard window for encoding or decoding. The Expand All button will expand all of the tree nodes, while the Collapse button will collapse all nodes of the tree. The JSON Inspector is automatically expanded if the body contains less than 2000 nodes; for performance reasons, you must manually expand the tree for larger documents. 126 | Inspectors RAW Type Allows Editing Request & Response No The Raw Inspector allows you to view the complete request and response, including headers and bodies, as text. The term "raw" is a bit of a misnomer, because Fiddler is still interpreting the bytes of the request and response; if you need a purely "raw" view of network traffic, use a packet sniffer like Microsoft Network Monitor (NetMon) or Wireshark. Most of the Inspector is a large text area that displays headers and the body interpreted as text using the character set detected using the headers, the byte-order-marker, or an embedded META tag declaration. Pressing CTRL+G in the text area allows you to move the cursor to a specified line number. When right-clicking on the text, the text area offers standard editing features like Cut, Copy and Paste options. The menu also offers an option to send the currently selected text to the TextWizard tool. There are also options to enable or disable word wrapping. The body content is shown in black. At the bottom of the Inspector is a bar that offers additional information and features. The first box shows the current cursor position in Line:Column format. The next box shows the current character: the hexadecimal representation of the current text selection, if any. Next is a search box that allows you to select matching text within the content. Pressing Enter or F3 will select the next match. Pressing CTRL+Enter will highlight all matches in the content. Pressing CTRL+F will activate the search box and select the text area above (to allow you to view search results in context). Matches are selected as you type, and the box will turn green if a match was found or red if no further matches were found. Pressing Enter or F3 will select the next match. Pressing CTRL+Enter will highlight all matches in the content. The View in Notepad button saves the text content to a temporary file and opens a text editor to view the file. The text editor launched is controlled by the fiddler.config.path.texteditor preference; notepad.exe is the default. 127 | Inspectors SYNTAXVIEW Type Allows Editing Request & Response Yes The SyntaxView Inspector allows you to view the request and response body using syntax-highlighting for a specific syntax. This is a very useful feature containing HTML, CSS, and JavaScript. It uses the ContentType header when deciding which highlighting rules to apply. The SyntaxView Inspector is one of the most useful Inspectors available for Fiddler, but it is not included in the default install package for size reasons. Fiddler's SyntaxView is currently at around 750kb, and including the Syntax Highlighting extensions would nearly double that size. To ensure that Fiddler updates are as compact as possible, the SyntaxView extension is available as a separate download from . Along the bottom of the SyntaxView is a search box that allows the user to find content in the response as an image. The Inspector can display most common image formats, including JPEGs, PNGs, and GIFs, as well as less common formats including cursors, images, bitmaps, EMF/WMF, and TIFF. The Inspector does not support display of SVG graphics; if IE9 or later is installed, SVG responses can be viewed using the WebView Inspector. The grey panel at left shows information about the currently selected image, including its size in bytes, pixel dimensions, and the format. At the right of the bar is text indicating whether the Inspector is in Read Only or Edit mode. To load selected text from temporary file and opens a text editor to view the file. The text editor launched is controlled by the fiddler.config.path.texteditor preference; notepad.exe is the default. If the Inspector replaces any right index with the Unicode replacement character (�) and an error message to the status bar on the... button at the right of the bar saves the content to a temporary file and shows Windows' Open With prompt to allow you to select an application to load the file. 130 | Inspectors TRANSFORMER Type Allows Editing Response only Always Response Encodings The HTTP specification defines a number of response encoding methods that can be used to compress or otherwise improve the performance of a response. Compression algorithms like DEFLATE can be applied to HTTP bodies to shrink the number of bytes that must be transferred over the network. The HTTP, script, and CSS shrink by up to 80% when compressed. Transformer-Encoding allows transmission of a body incrementally; its length is not known ahead of time. Ordinarily, the body's size is sent as a Content-Length header, but pre-calculating a body's size could require a great deal of time and memory, particularly if the body is being generated by running a database query or other expensive operation. Without Chunked Transfer-Encoding, a server returning a response of indeterminate size would be forced to send a Connection: close header and close the connection when the response is complete. Such closures break HTTP Keep-Alive mechanism and introduce a performance bottleneck. Chunking works by sending one or more chunks of data, each preceded by a Length value specified as a hexadecimal number. A chunk length of 0 signals the end of the body. For instance, here's a response which has been chunked: HTTP/1.1 200 OK Content-Type: text/plain Transfer-Encoding: chunked 2b This is a response which has been delivered 21 using HTTP Chunked encoding. 0 To r38 educe overhead, the chunks should be larger than those in 0c this example 18 e; 2kb is a common size. 0 131 | Inspectors Adding and Removing Encodings The Transformer Encodings can make a world of difference in a response's performance, but they can make it more difficult for you to inspect reality bodies. At the top of the Transformer tab, the current body's size is listed; keep an eye on this number as you add or remove encodings. The Chunked Transfer-Encoding checkbox adds or removes Chunked Transfer-Encoding from the response. Toggling this

checkbox applies or removes the encoding, then adds or removes the Transfer-Encoding and Content-Length response headers. After enabling chunking, the HTTP Compression box below the checkbox is disabled. To make changes to compression, you must first remove chunking, then adjust HTTP compression. The radio buttons in the HTTP Compression box allow you to compress or decompress the body: No Compression: the body is not compressed. GZIP: The body is compressed using GZIP. This encoding uses DEFLATE internally, but with slightly different binary formatting. DEFLATE: The body is compressed using the DEFLATE (RFC1951) algorithm. BZIP2: The body is compressed using the BZIP2 algorithm. BZIP2 typically provides better compression than GZIP or DEFLATE at the cost of additional CPU time. This format was briefly supported by the Google Chrome browser but is currently not supported by any mainstream browser or server. It exists in Fiddler mostly for comparison purposes and to enable testing to determine how clients handle unknown encodings. Selecting a radio button applies or removes the compression from the body, then adds, updates, or removes the Content-Encoding header and adjusts the Content-Length header. You can use this Inspector to determine how effective HTTP compression will be when applied to your content—for most textual types, compression reduces the body's size by 80% or so. Unlike most Inspectors, the Transformer Inspector allows you to modify ReadOnly (complete) responses without encoding on Sessions for easy inspection. 132 | Inspectors Presently, the Transformer Inspector is available for responses only. Use of HTTP encodings for requests is extremely rare, and generally not supported by most clients or servers. Other Ways to Remove Encodings Because most of Fiddler's Inspectors do not function well with encoded content, when an encoded Session is selected, a yellow button will appear above the list of Inspectors. Clicking the button will immediately remove all encodings from the request or response. You can remove encodings from both the request and response simultaneously by right-clicking the Session in the Web Sessions list and choosing Decode Selected Sessions from the context menu. Similarly, enabling the Decode option in the toolbar will automatically remove all encodings as Fiddler reads requests from the client and responses from the server. 133 | Inspectors WEBFORMS TYPE Allows Editing Request only Yes The WebForms Request Inspector parses the request's query string and body for any HTML form-data. If a form is found, it is parsed and the name/value pairs are displayed in the grid view. For instance, the following request: POST /sandbox/FileForm.asp?Query=1 HTTP/1.1 Content-Type: application/x-www-form-urlencoded Host: www.fiddler2.com Content-Length: 52 z=Data%3e123&flientry2=a%2e&xkb_charset_=windows-1252 ...is displayed as follows: This Inspector hosts text with application/x-www-form-urlencoded data used by most simple web forms. Support for multipart/form-data forms, commonly used for file uploads, is limited to display only. To modify a file upload, use the HexView Inspector instead. 134 | Inspectors WEBVIEW Type Allows Editing Response only No The WebView Response Inspector allows you to view responses in a web browser control, which provides a quick preview of how a given response may appear in a browser. The web browser control is configured to prevent additional downloads when rendering the response to prevent muddling your Web Sessions list-- this means that most images, styles, and objects will be missing from the displayed content. Additionally, scripting and navigation are blocked, providing a read-only preview of HTML pages. Beyond plain XHTML and HTML, the WebView Inspector is able to render several additional media types if IE8 or later is present. If IE8 is present, the Inspector can render any binary image (png, jpg, gif, etc) smaller than 24kb. This limit exists because the WebView URL to render the image, and IE8's URL size limit is 32kb, which is equal to 24kb of binary. When IE9 is present, the Inspector can display image previews for WOFF, TTF, and EOT font files, MP3 audio files, and h264 video files. For instance, inspecting a WOFF file will generate the following display: When previewing an audio or video file, an "AutoPlay" checkbox appears at the top-right of the tab. When this box is ticked, media files will automatically begin playback when loaded. When unticked, the media will be loaded but will not begin playing until the play button is clicked inside the preview. 135 | Inspectors TYPE Allows Editing Request & Response No The XML Inspector interprets the selected request or response body as an Extensible Markup Language (XML) formatted string, showing a treeview of the XML document's nodes. If the body cannot be interpreted as XML, the treeview will remain empty. Each XML element is represented as a node in the tree, and the attributes of the element are displayed in square brackets after the element's name. Unlike most inspectors, the XML Inspector is able to render the data from encoded content. The XML Inspector will not collapse all nodes of the tree when the Session expires. Copy, to copy the selected node's content to the clipboard for press CTRL+C), and Send to TextWizard, to send the selected node's content to the TextWizard window for encoding or decoding. The Expand All button in the footer will collapse all nodes of the tree. The XML tree will be automatically expanded if the body contains less than 2000 nodes; for performance reasons, you will have to manually expand the tree for larger documents. 136 | Inspectors Extensions OVERVIEW Fiddler's rich extensibility model enables developers to add powerful new features to Fiddler via easily installed add-ons. Popular 3rd Party Extensions Independent developers have built many Fiddler extensions, some of which are listed in the directory at . At the time of this writing, the most popular 3rd-party extensions enhance Fiddler's ability to test the performance and security characteristics of web applications. Performance Add-ons On its own, Fiddler can be used for many important performance-analysis and optimization tasks, but several extensions add even more power. neXpert Performance Report Generator – Written by a Microsoft Online Services testing team, neXpert will evaluate your websites' adherence to performance best-practices, generating a report which flags problems and recommends solutions. License: Freeware. StresStimulus – This load-testing extension permits you to record and run load-test scenarios against your website to evaluate its ability to scale to handle large numbers of simultaneous users. License: Free trial. Security Add-ons Many Security experts have built several powerful add-ons that enable more sources to discover and resolve security issues. Watcher – Developed by the Casaba Security team, Watcher is a "passive security auditor" which observes a browser's interactions with your site. The tool scans requests and responses, flagging potential security vulnerabilities. This powerful tool is used by professional security penetration testers to evaluate major sites. License: Open Source. x5s – Another add-on from Casaba Security, x5s evaluates your website's vulnerability to cross-site scripting bugs caused by character-set related issues. License: Open Source. intruder21 – This add-on enables fuzz-testing of your web applications. After you identify target requests in Fiddler, this extension generates fuzzed payloads and launches those payloads against your site. License: Freeware. Ammonite – This add-on detects common website vulnerabilities including SQL injection, OS command injection, cross-site scripting, file inclusion, and buffer overflows. License: Free trial. Extensions I've built The remainder of this chapter describes the most useful extensions that I have developed, all of which are available for free download from . 138 | Extensions Some of the extensions will be useful for most users of Fiddler and aren't built into the tool simply for download size reasons. Other extensions are only useful for narrow scenarios, and providing their functionality via the add-on model allows me to prevent "bloat" in Fiddler-- and ensure that Fiddler's add-on model is powerful enough to meet the needs of the development community. Source code for many of these extensions is available so that they may serve as examples for developers who wish to extend Fiddler to better meet their own needs. 139 | Extensions JAVASCRIPT FORMATTER When the JavaScript Formatter is installed, you can right-click on any Session with a JavaScript response and choose Make JavaScript Pretty. This command reformats (or "beautifies") JavaScript to significantly improve its readability if it had been "crunched" or "minified" for instance, the following line of script: Is reformatted for readability: To enable automatic reformatting of all script responses, enable the Make JavaScript Pretty option in Fiddler's Rules menu. When JavaScript is reformatted on download, the client will only see the reformatted form, which can be helpful when you are using the Inspectors to debug its behavior. You can manually control whether the JavaScript formatter runs on a response using the X-Format-JS session flag. When this flag is set to a value of 0, the extension will not format the response body, even if the Make JavaScript Pretty option is enabled on the Rules menu. This extension is not compatible with a very small number of responses, for which the reformatting process may introduce corruption. That problem arises in two scenarios: first, when a response is sent with a JavaScript MIME type but it's not actually JavaScript (as seen on some Google properties). Second, the reformatting process may cause problems if the (very obscure) JavaScript line-continuation feature is used. Line continuation allows script developers to end a line with a backslash and the JavaScript engine will automatically concatenate that line with the following. The JavaScript Formatter's parser is not yet aware of this feature and will not format such lines properly. 140 | Extensions GALLERY The Gallery extension is designed to display any images that have been found in the selected Sessions. This extension is useful when you wish to interact with a large number of images, or wish to quickly select a Session based on an image. The extension has only a few options, displayed across the top of its tab: By default, each image is shown as a 150x150 pixel thumbnail, and images smaller than 10kb in size are ignored. Clicking the Filter sessions link will select the Sessions for which images appear, unselecting any Sessions which do not represent an image whose thumbnails were removed from the view. Clicking the Help link shows how to explore the features of the extension. Hovering over a thumbnail will show the Session ID, URL, dimensions, and image format in a balloon tip. Middle-clicking on a thumbnail removes the image from the Web Sessions list. Right-clicking on a thumbnail will switch away from the Gallery to inspect the chosen image's Session using the Inspectors. If you hold Shift while rightclicking, or double-clicking, the Inspectors will open in a popup window. Full-Screen View Double-clicking on a thumbnail enters the full-screen image viewer mode. While the image's Session has a Comment set, this comment is shown as a caption at the bottom of the screen. While in full-screen mode, use the mouse or keyboard to control the view: Exit full screen mode by middle-clicking or pressing the Escape key. To advance to the next image, left-click, hit Spacebar, hit the Right arrow key, or hit Page Down. To go back to the previous image, right-click, hit Shift+Spacebar, hit the Left arrow key, or hit Page Up. Press the Z key, Enter, or use the mouse wheel to toggle the Zoom level (actual Size, or Stretch-to-Fit). If the image is shown Actual Size and is larger than the screen, use the mouse to pan around the image. 141 | Extensions To start at automatically-advancing slideshow, press any number key 1 to 9, representing the number of seconds to linger on each image. Press the 0 key to cancel the slideshow's automatic advancement. By combining the slideshow with the Comment-displayed-as-Caption feature, you can use Fiddler as a basic photo presentation tool. Various options are available to temporarily change the appearance of an image. Press H to flip the displayed image horizontally, or V to flip the image vertically. Hit R to rotate the image by 90 degrees clockwise. Press I to invert the image's colors. Press S to apply a Sepia tone, and G to convert the image to grayscale. Pressing C will swap the green and blue colors in the image. Pressing E will convert the image to grayscale except for any Red pixels; this is a slow operation and may take several seconds for large images. Press U to undo all of the manipulations made to the display of the image. Press the Delete key to remove the image from the Gallery and advance to the next image. 142 | Extensions CONTENT BLOCKER The Content Blocker extension enables you to easily block selected content from download by returning a HTTP/404 response to the client. This functionality allows you to test your web applications' behavior if content is blocked by an ad-blocker or the Tracking Protection feature found in IE8 and later. To install the extension, you must download it from the Fiddler website and copy the DLL into your \Program Files\Fiddler2\Scripts folder. After installing the extension, a new ContentBlock menu will appear on the Fiddler main menu. The ContentBlock menu offers several simple options: Enabled Block Paths Edit Blocked Hosts Always Block Flash Block X-Domain Flash AutoTrim to 400 sessions Hide Blocked Sessions When checked, blocking is enabled. When unchecked, no content will be blocked. When checked, requests are blocked if the URL path contains the text /ad When clicked, this menu item opens a new window that allows you to edit the list of hosts from which content will be blocked. The list consists of fully-qualified hostnames and is delimited by semicolons. When checked, Adobe Flash content will be blocked, regardless of its origin. When checked, requests are blocked if the URL path contains the text /ad When checked, Adobe Flash content will be blocked if it originates from a different site than the requesting page. The Referer header is consulted in making this determination. When checked, this extension will automatically trim the Web Sessions list such that it contains no more than 400 sessions. The Keep Only dropdown on the Fiddler toolbar offers the same functionality. When checked, Sessions that are blocked will be hidden from the Web Sessions list. Instead of choosing Block this Host from the context menu. This extension was developed as an example of how to use the Fiddler extension model (its source is included in the download) and as such it offers limited functionality that can be found in other parts of Fiddler. 143 | Extensions TRAFFIC DIFFER The Traffic Differ extension aims to help you easily determine how two sets of captured traffic are different. This can be useful, for instance, you have one SAZ file captured by a customer that encountered a problem from your site and another SAZ file captured which represents the "working" case. The Traffic Differ enables you to load these two SAZ files and then compare the Sessions within them as a whole. Alternatively, you can drag and drop Sessions from the current Web Sessions list to either of the Session lists and use those Sessions for comparison. Two side-by-side Session lists show each Session's status code, URL, response size, and a hash of the response body. To compare any two individual Sessions between the two captures, select one Session in each list and then click the Compare Selected Session button. The file comparison tool will launch and compare the request and response headers and body text. 144 | Extensions FIDDLE RSCRIPT EDITORS The Syntax Highlighting extensions package adds a new FiddlerScript tab to the main Fiddler UI and also provides a standalone Fiddler2 ScriptEditor application offering similar functionality. It also includes the SyntaxView Inspectors, described in the Inspectors chapter. FiddlerScript Tab The FiddlerScript tab allows you to easily view and update your FiddlerScript rules directly in the Fiddler UI. Across the top of the tab are a few UI controls: Save Script button Go to dropdown Saves changes made to the script This dropdown lists key methods in the FiddlerScript. Selecting an item from this list will scroll to the specified method. Choosing the *FiddlerScript Reference option will open the FiddlerScript reference web page. Find... Icon ClassView button Hidden feature: If you double-click the Go to text label itself, the script source code will collapse to definitions. This box allows you to search your script for the specified text This button toggles the visibility of the FiddlerScript ClassView sidebar. Below these controls is a syntax-aware source code editor containing the current text of your FiddlerScript. The editor supports context-sensitive code completion and will show available properties, fields and methods depending on your type. Code items that you've changed will show a yellow bar in the left margin....until you save your changes, at which point the bar will turn green. 145 | Extensions ProviderScript's a script error when compiling your script, the FiddlerScript tab will activate and scroll to show the error. ClassView Sidebar The ClassView sidebar allows you to explore the key objects, properties, and methods available to your script. Note that the ClassView does not expose all available functionality, only the functionality that is most commonly used by scripts. Clicking an item in the treeview will show the description of that item at the top of the sidebar. Pressing CTRL+C on an item will copy the item's name to the clipboard, allowing you to easily insert it in your script. The sidebar's items are color-coded depending on their type: Color Black Blue Green Type Method Property Field Purple Event Fiddler2 ScriptEditor The Fiddler2 ScriptEditor program opens when you click Rules > Customize Rules or hit CTRL+R in Fiddler. You can also launch the Fiddler2 ScriptEditor from anywhere in Windows by typing Windows+R and typing fse2 in the Windows Run prompt. The standalone ScriptEditor offers the same general functionality as the FiddlerScript tab, with the convenience of running in its own window. Unlike the FiddlerScript tab, however, the standalone application is not notified of script compilation errors and will not notify you of any problems that Fiddler encounters when compiling your script. The Go menu offers commands to navigate quickly to important methods in the FiddlerScript, and, to scroll to a specific line. 146 | Extensions The View menu offers options to adjust the display of the source code, including changing the font size, showing line numbers, and expanding and collapsing method blocks. It also includes the toggle which shows and hides the ClassView Explorer sidebar. The Insert menu offers a number of pre-built snippets that you can add to your script. To use these, place the cursor where you'd like the snippet added, then click the menu item representing the desired snippet. For instance, place the cursor just inside the Handlers class' opening brace: Then click Insert > Context Menu Item: The Editor will insert a template ContextAction block which will add a new menu item to the Web Sessions context menu: 147 | Extensions SAZCLIPBOARD The SAZClipboard is a simple extension that allows you to load a .SAZ file into a standalone Session list in its own window. The window enables you to drag Sessions between this clipboard and the main Fiddler UI. This functionality is convenient if you're using the Composer or AutoResponder features, both of which accept drops of Sessions for reuse. To use the extension, click New SAZClipboard... from Fiddler's Tools menu. You can drag and drop from the Web Sessions list to the SAZClipboard, or load a SAZ directly into it using the button at the bottom. Drop a .SAZ directly into a target window in the Save file dialog box to quickly determine the user's real-world location. Most browsers implement this functionality by querying the operating system for a list of nearby Wi-Fi access points, then submitting that list to a webservice which maps the nearby access points to a longitude/latitude pair. The webservice provides that value to the browser, which then returns it to JavaScript. Fiddler's AnyWHERE extension allows you to trivially spoof the responses to geolocation webservice queries, enabling you to "trick" the browser into thinking you're somewhere you are not. The window allows you to type in your current location, or select a location from a list of worldwide points-of-interest. Because browser geolocation webservice queries travel over HTTPS, you must enable HTTPS decryption for this add-on to operate. This extension supports IE9+, FF4, Chrome, and Opera, but it will not work if the browser's geolocation feature is used based on something other than webservice lookups. For instance, when IE10 is running on a Windows 8 device that includes GPS hardware, Windows will use that hardware rather than calling a webservice to determine the location. Source code for this extension is provided in the download package. 149 | Extensions Storing, Importing, and Exporting Traffic SESSION ARCHIVE ZIP (SAZ) FILES Fiddler's default save format is the Session Archive Zip (SAZ) file. SAZ files are simply zip archives that are constructed in a particular way that Fiddler understands. By renaming a .SAZ file to .ZIP, you can use Windows Explorer or WinZip to explore the files inside. You'll see how it works under the covers. Because it contains all of the traffic, a SAZ file may grow quite large, especially if it contains images, sounds or video responses. In contrast, HTML, script, and CSS files tend to compress well, at about a 5:1 ratio. It is possible to store SAZ files with gigabytes of data and tens of thousands of Sessions, but because such files are unwieldy to load or transfer, you will generally want to use Fiddler's filtering features to minimize the amount of data stored in any given SAZ. You can later load a SAZ file into Fiddler to view the Sessions contained within; these Sessions will be colored with a light grey background color for easy of identification. One thing Fiddler won't insert in Sessions loaded from a SAZ file is Session ID; when a SAZ is loaded, each session gets a new ID starting at the current index. Before sharing a SAZ file with someone else, use the Comments button or Edit > Mark menu to tag any Sessions with a comment. If Fiddler reloads a Session which does not already have a comment, it will insert a randomly-generated comment containing the original session's ID. Tip: The AutoResponder tab is able to import a SAZ file and replay the responses contained within to mimic the original traffic flow captured in the SAZ file. Protecting SAZ Files SAZ files contain all captured web traffic, which might include sensitive information like usernames, passwords, cookies, and account information; if such information was captured by Fiddler. Therefore, you should only share SAZ files with trusted people or organizations. If a SAZ file must be stored in an unprotected location or transferred over an insecure connection, you can help keep it private by encrypting the contents before saving. To do so, select Password-Protected SAZ in the Save as type dropdown on the Save Session Archive prompt. After choosing to save an encrypted format, a password will be requested: Password-Protected SAZ files are encrypted using 128-bit AES encryption, a strong form of encryption that is also used for HTTPS traffic and by government and military organizations. Of course, to keep the file private, your password must be kept secret. If the file is being shared with someone else, you must securely communicate the password to the other person. Don't forget your password—there's no practical way (short of brute-force guessing) to reopen your SAZ file without it. The preference fiddler.saz.AES.Use256Bit may be set to true to configure Fiddler to use 256-bit AES instead of 128bit. Alternatively, the legacy CONFIG.bUseAESForSAZ property may be set to false to configure Fiddler to use the legacy PKZIP obfuscation scheme, which is very fast but provides much weaker protection of the file's contents. 151 | Storing, Importing, and Exporting Traffic FIDDLE RCAP SAZ files have proven to be useful in capturing and reproducing problems with web applications that I soon found that many web developers were requesting SAZ captures from novice PC users. In the hope of enabling SAZ capture, I built FiddlerCap. FiddlerCap is a lightweight tool designed to allow non-technical users to capture SAZ files for later analysis by experts using the full Fiddler debugger. FiddlerCap offers a simplified user-interface and streamlined workflow for capturing web traffic; unlike Fiddler itself, it cannot modify traffic and has no extensibility mechanisms. Like Fiddler, you can use FiddlerCap to collect web traffic from any application developed using any technology. FiddlerCap is built atop the FiddlerCore library and uses a FiddlerCore proxy instance running on port 8889 to capture web traffic. Install FiddlerCap by visiting , a simple page which provides simple step-by-step instructions on how to use the tool to capture web traffic. By default, FiddlerCap installs to a temporary location on your desktop and it does not require Administrative permissions. To simplify worldwide use, FiddlerCap has been localized to Spanish, French, Italian, Japanese, Portuguese, and Russian. When the install completes, the tool automatically opens. The window is divided into three groups: Capture Options, Capture, and Troubleshooting. The button at the bottom right of the FiddlerCap window will expand the window to show a simple Session list of the captured traffic. Sessions' content cannot be inspected, but individual sessions may be removed by selecting them and pressing the Delete key. Capture Has The Capture box offers the minimum set of controls needed to collect a capture. Press the Start Capture button to begin capturing web traffic. A new browser window will open for you to reproduce the problem, although FiddlerCap will capture traffic from any process that supports the system proxy setting—not just that single browser 154 | Storing, Importing, and Exporting Traffic window. The FiddlerHook extension for Firefox is not available for FiddlerCap. To capture Firefox's network traffic, open the browser's Tools > Options > Advanced > Network > Connection Settings screen and select the Use System Proxy Settings option. After the problem has been reproduced, click the Stop Capture button to end the capture of traffic. Lastly, click the Save Capture button to select where to save the .SAZ file. By default, the SAZ is saved, but you may, select the Password-Protected SAZ option in the Save as type dropdown. Capture Options Box The Capture Options box controls the options shown in capturing. The Store binaries checkbox (unticked by default) controls whether binary response bodies (e.g. images, audio, video) and application/octet-stream downloads) will be stored in the capture. Omitting these bodies (and their headers) can dramatically shrink the size of the final SAZ file, although obviously if the problem being reproduced depends on the content of such downloads, you should enable the Store binaries option. The Decrypt HTTPS traffic checkbox (unticked by default) controls whether or not HTTPS traffic will be decrypted by FiddlerCap. When you check this box, an explanatory message is shown: ---------------------A note about HTTPS Decryption ---------------- HTTPS decryption will enable your debugging proxy to see the raw traffic sent via the HTTPS protocol. This feature works by decrypting SSL traffic and reencrypting it using a locally generated certificate. If this certificate and remove it when you are done. You may choose to temporarily install this certificate in the Trusted store to avoid warnings from your browser or other applications. ---------------------This note is dismissed, a Windows Security prompt is presented to allow you to trust FiddlerCap's root certificate: 155 | Storing, Importing, and Exporting Traffic When FiddlerCap is closed, it will automatically delete all of the certificates it generated during the capture, prompting for permission to delete the root certificate from the Trusted Certificates store. The Store cookies and POSTs option (checked by default) controls whether FiddlerCap will store POST request bodies and the HTTP headers Cookie, Set-Cookie, Set-Cookie2, Authorization, and Proxy-Authorization. Disabling this option can slightly shrink the capture and somewhat reduces the amount of privacy-impacting information stored in the capture. Even when this option is disabled, however, captures obviously may contain other sensitive information and thus they should only be shared with trusted parties. The Automatically discard traffic after # minutes option allows you to control how many minutes of traffic are saved in the SAZ file. This option is most useful when trying to capture a problem that happens intermittently. You can leave FiddlerCap running in the background and it will periodically expire older traffic to reduce the size of the capture and the amount of memory used. When you encounter the problem you are trying to capture, you can just use the last few minutes' worth of traffic to the archive. Tools Box The Tools box offers the simple set of tools helpful in reproducing a problem. The Clear Cookies button is enabled only while FiddlerCap is capturing; it immediately clears all previously-captured Sessions from the capture. 156 | Storing, Importing, and Exporting Traffic The Snapshot button will take a screenshot of whatever monitor FiddlerCap is running on and add it to the capture as a JPEG-formatted image. It will be stored in a new session whose URL contains the current timestamp in the format . The flag button will prompt you to add a comment to the capture. The comment text entered will be stored as the HTTP response body of a new session with a URL of . 157 | Storing, Importing, and Exporting Traffic FIDDLE R'S VIEWER MODE Ordinarily, only one instance of Fiddler may be started at a time; attempting to launch a new instance of Fiddler while it's already open will simply reactivate the existing Fiddler instance. However, it is sometimes useful to open additional instances of Fiddler to view SAZ files or compose requests, independent of the existing Fiddler instance. To achieve those goals, Fiddler supports opening additional Viewer instances by passing the -viewer command line argument, or by right-clicking on a SAZ file in Windows Explorer and choosing Open in Viewer from the context menu. Viewer Mode instances can be identified by the Viewer Mode icon in the front of the toolbar. When no SAZ file is loaded, the text shows FiddlerViewer and by the text You can identify this Viewer instance by clicking on the text and entering a new title. If you load a SAZ file into the instance, the text will change to the name of the SAZ file, with a tooltip showing the full path to the file. Fiddler Viewer instances cannot capture traffic, and most UI changes made in Viewer mode (e.g. hiding the toolbar or reordering columns in the Web Sessions list) will be discarded when the instance is closed. 158 | Storing, Importing, and Exporting Traffic IMPORTING AND E XPORTING SESSIONS Beyond Fiddler's native SAZ format, the debugger offers a rich set of Importers and Exporters (known collectively as "Transcoders") that allow it to share captured network traffic with other tools. In the Transcoders bundled with Fiddler, an extensibility mechanism permits support of additional formats to be added by other developers. Fiddler's Import and Export functionality can be found on the File menu. The Import Sessions... command will show you to load Sessions from other formats, while the Export Sessions submenu allows you to export either all Sessions selected in the Web Sessions list. Import Formats The current version of Fiddler includes two Import formats: HTTPArchive and F12 NetXML format. HTTPArchive (HAR) is a lossy JSON-based format which is supported by a number of tools including Firebug, Chrome's Developer tools, and HTTPWatch. Versions 1.1 and 1.2 of the format are supported. The F12 NetXML Format is a very similar format exported by the IE 9 Developer Tools' Network tab; it's actually just an HTTPArchive file encoded using XML instead of JSON. This was not an intentional difference on Microsoft's part: the original HTTPArchive format spec defined the format using XML syntax. Only a small comment at the top of the specification noted that JSON should be used instead. Fiddler is the only popular tool to read the NetXML Format; you can use Fiddler to load NetXML and save it to a more common format like HAR. Now that both the JSON and XML formats are being mixed (in particular, large binary bodies like images will not be present). After choosing a format to Import from, you will be prompted to select the file to import. The selected file will be parsed and the traffic will be added to the Web Sessions list. Support for other Import formats can be added by any developer. I'm currently working on an Importer that allows import of Wireshark .pcap and NetMon .cap files. Export Formats Fiddler includes a broad set of Exporters that allow it to export traffic for use in other tools or applications. The remainder of this chapter describes the export formats supported by Fiddler itself. 159 | Storing, Importing, and Exporting Traffic HTML5 AppCache Manifest HTML5 introduces the concept of an Application Cache, which allows a web developer to provide a manifest of pages that should be cached to permit offline use. The manifest specifies which resources the browser should download into the AppCache and which resources should always be retrieved from the network. Manifests are simply text files and you can create them with your text editor of choice. However, this process can be tedious and prone to error. Fiddler's HTML5 AppCache Manifest Exporter makes the process of generating a manifest a straightforward exercise: 1. Clear your browser cache. 2. Start Fiddler. 3. Load your website in the browser. 4. In Fiddler, click File > Export Sessions > All Sessions... 5. In the Select Format window, pick HTML5 AppCache Manifest. 6. Click Next. In the Adjust AppCache Manifest window, check any resources that you wish to exclude from the CACHE section of the manifest; those will be added to the NETWORK section of the manifest. 7. If you'd like, use the text box at the bottom to specify a Base URL if the resource URLs should be relative to the manifest. For instance, in the example above, I will place the manifest in the file with Content-Type: text/cache-manifest 10. Storing, Importing, and Exporting Traffic If you're happy with the manifest, save it to your web server in the appropriate location. Ensure that your web server is configured to return the manifest file type with Content-Type: text/cache-manifest 10. In your web page, ensure that your page will reference the manifest using the HTML5 doctype and add a manifest attribute on the HTML element pointing to your application's manifest. Keep in mind that when the browser is using AppCached content, those resources will be reused from the cache and not pulled from the network (until the cache is expired). Because the content is cached locally, you will not see requests to redownload that content in Fiddler. If you wish to reload Fiddler's AppCache, you can do so using Internet Explorer's or another browser's cache so that the browser is forced to re-download the content from the network on next use. HTTPArchive v1.1 and v1.2 The HTTPArchive format is a JSON format that contains Web Sessions' headers, some bodies, and timing information. The primary difference between versions 1.1 and 1.2 of the format is that the newer version permits the inclusion of small binary bodies. By default, Fiddler will store text bodies up to 1MB in length, and binary bodies up to 32768 bytes in length; these values can be changed by setting the appropriate preferences. fiddler.importexport.HTTPArchiveJSON.MaxTextBodyLength fiddler.importexport.HTTPArchiveJSON.MaxBinaryBodyLength If a body is too long, it will not be stored in the file and instead a comment will be added inside the body noting the omission. 161 | Storing, Importing, and Exporting Traffic MeddlerScript Meddler is a HTTP traffic generation tool which I developed to allow building of tiny, self-contained reproductions of HTTP traffic. Meddler is essentially a "scripted socket" which allows very lowlevel manipulation of sockets, a capability useful for debugging browsers or other clients. When you export Web Sessions to MeddlerScript, the result is an .MS file that can be loaded into Meddler to "play back" that traffic. Conceptually, this is very similar to using Fiddler's AutoResponder to play back previously captured traffic from a SAZ file, but Meddler scripts can be easier to automate and they permit low-level tweaks (e.g. "Wait 300ms between sending the headers and the first byte of the body, then write the body in 2048 byte chunks each 10 seconds apart") which cannot be easily replicated using Fiddler itself. Unless you're building a browser or other HTTP client, exporting MeddlerScript probably will not be very useful for you. Raw Files The Raw Files Exporter allows you to trivially dumping media content to disk. For instance, say you've browsed around a photography website and collected a bunch of photos in the Web Sessions list. You can use filters and other features to select the photos of interest (e.g. JPG files over 20kb). You can then select File > Export Sessions > Selected Sessions and choose Raw Files in the Select Export Format box. The File Exporter window allows you to configure the export process. The Path box allows you to select the base path under which a new folder will be created. The folder will be automatically named \Dump-MonthDay-Hour-Minute-Second\. The Options box contains two checkboxes. Recreate Folder Structure will create subfolders for each resource based on the hostname and path of the file. Clear Text uses to mimic a site's hierarchy on your local disk. The Open folder when complete option will open Windows Explorer to display the root folder into which files were exported when the export has completed. Click the Export >> button to begin the export. Beyond dumping media files, the Raw Files Exporter allows you to easily mirror a captured website to your disk. You can then drag/drop the contents of that folder to the AudioServer tab, and Fiddler will serve the local content mirror based on inbound requests. This practice can be very convenient for debugging a website when you want to use other tools (e.g. Expression Web or Visual Studio) to edit HTML content locally. Visual Studio WebTest Generating a Visual Studio WebTest file allows you to use the Visual Studio Web Test product to release previously captured requests for functional-testing and load-testing purposes. Visual Studio Web Test is included in some editions of Visual Studio 2008 and later. WCAT Script As explained on the tool's website: Web Capacity Analysis Tool (WCAT) is a lightweight HTTP load generation tool designed to measure the performance of a web server within a controlled environment. WCAT can simulate thousands of concurrent users making requests to a single web site or multiple web sites. The WCAT engine uses a simple script to define the set of HTTP requests to be played back to the web server. Fiddler's WCAT Exporter allows you to easily generate request-generation scripts for WCAT to replay against your servers to verify their ability to handle load. You can download the 32bit or 64bit WCAT installer from . 163 | Storing, Importing, and Exporting Traffic EXTENDING FIDDLER WITH FIDDLE RSCRIPT The earliest versions of Fiddler had no extensibility model at all; the features I coded were all that users had available. It didn't take long to recognize that I would never be able to keep up with the myriad feature requests coming in from Fiddler users who were using the tool to solve a huge variety of problems. One of the biggest early limitations was that Fiddler only offered one filter (Hide Images) and thus it was easy to get overwhelmed by the sheer volume of traffic flowing through the tool. I had planned to build a filtering interface full of dropdown fields and textboxes which would allow users to filter traffic based on boolean criteria that I would make available. There were two obvious problems with this approach: first, it would require that I do a lot of tedious UI development, and second, I knew that most users wouldn't be happy with it. Advanced users find it very cumbersome to build complicated queries using UI controls, and novice users would be easily confused when trying to build complicated queries that involved nested AND, OR, and NOT operators. Fortunately, laziness often leads to better engineering. I remember thinking one night: "It's so easy for me to build complicated filter expressions inside Fiddler's code itself. If only users could just write code to do filtering." Happily, this thought led to a "eureka" moment-- the recollection that the .NET Framework makes it very easy to build a script engine into an application, and exposing the application's object model to that script engine is also very easy. With a few dozen lines of code added in late 2003, Fiddler got infinitely more powerful. Even as Fiddler was modernized over the last 9 years, including the introduction of Fiddler extensions and many built-in filtering capabilities, the FiddlerScript engine has remained an indispensable component of Fiddler. By mastering FiddlerScript, you will get much more out of Fiddler. About FiddlerScript Fiddler's Script language is built upon JScript.NET, a variant of JavaScript. methods in the CustomRules.js script file are run, enabling you to hide, flag, or modify Sessions based on criteria of arbitrary complexity. Your rules script can be modified and recompiled dynamically at runtime without restarting Fiddler. This book does not provide a comprehensive reference to the JScript.NET language upon which FiddlerScript is based. If you plan to code especially powerful or complicated FiddlerScript, you should consider finding a copy of Microsoft JScript.NET Programming. This book presents a comprehensive view of the language, including material I haven't seen anywhere else. (I discovered it in 2012, ten years after its only printing. In one of those "small world" coincidences, it was authored by Justin Rogers, a colleague on the Internet Explorer team.) JavaScript is a language familiar to most Fiddler users, and JScript.NET isn't syntactically much different than the C# language favored by most .NET programmers. There are two deltas that JavaScript developers should be aware of: 1. JScript.NET supports a C#-like mechanism for declaring classes that contain fields, properties and methods. Fiddler integrates your script by calling methods on a class named Handlers, described below. 166 | FiddlerScript 2. When declaring a variable, you may (and should) declare that variable's type. Type definition improves the performance of the generated code and allows the compiler to flag errors at compile time instead of runtime. To set the type of a variable, just specify the type after the variable name and before any initial value, like so: var sMyString: String = "StringValue"; var bMyBool: boolean = false; var iMyInt: int = 42; For ease of development, the FiddlerScript engine will automatically call specially-named functions in your script when events of interest occur. However, less-interesting events do not get this special treatment and you must attach handlers for such events. To supply an event handler in script, call the Subscribe static method. When you want to attach any event handlers in this way, you'll need to use the following code: // Call the subscribe method using the format Object.add_EventName(handlerFunction) FiddlerApplication.add_OnClearCache(m:OnClearCache); // Implement the event handler static function clearCache(sender, oArgs:CacheClearEventArgs); cacheKeys:Show("User cleared the cache.") } } If you do attach any event handlers in this way, you'll need to use the remove EventName method to remove the event handler in your script's OnBeforeReload function. Over the years, a variety of powerful features were added to FiddlerScript, including the ability to extend the Fiddler UI and accept automation commands from the ExecAction.exe command line tool. As Fiddler's scripting capabilities became more powerful, I began building some core features of Fiddler (for instance, UserAgent spoofing and performance simulation) in FiddlerScript because users wanted to customize these features slightly. By implementing such features in the easily-updated script, Fiddler enables users to customize the tool to their exact needs. Editing FiddlerScript To see the current FiddlerScript loaded in Fiddler, choose Customize Rules on Fiddler's Rules menu. The configured script editor will launch and show the script. By default, Notepad will be used as the script editing program, but you can adjust this preference using the Tools > Fiddler Options menu. If you plan to do any non-trivial modifications to the FiddlerScript, I highly recommend that you install the free ScriptEditor, a syntax-highlighting editor that can either run standalone or on a tab within Fiddler itself. 167 | FiddlerScript Updating FiddlerScript at Runtime When you save the CustomRules.js file, Fiddler will play a sound while reloading the source code and attempting to recompile it. If compilation succeeds, the new script will be loaded for all subsequent Sessions. If compilation fails, you'll be shown an error message and no script will run until the error is corrected and the script is recompiled. The compiler's error message will show the offending line and attempt to explain the nature of the error. Keep in mind that each time your script is recompiled, the previously-running script is first discarded. This means that the current values of variables or properties in the script will be lost to their default values every time the script is reloaded. If you need to preserve a variable's value from the old script to the new one, you should use a Preference to load and store the value. See the OnRetire method for more information. Resetting to the Default FiddlerScript Don't worry about "breaking" your script—if you make a mistake and corrupt the script to the point where you don't know how to fix it, simply delete the CustomRules.js file from the %USERPROFILE%\My Documents\Fiddler2\Scripts folder. The next time Fiddler is started, your CustomRules.js file will be recreated by copying the SampleRules.js file from the Fiddler program folder. CustomRules.js file is that when you install new versions of Fiddler, your old rules file will be preserved. Any new or updated rules stored in the SampleRules.js file will not be loaded when you delete your CustomRules.js file or copy the updated content from the sample script. 168 | FiddlerScript FIDDLE RSCRIPT FUNCTIONS When FiddlerScript is compiled, Fiddler obtains references to several key static functions, all of which are found on a class named Handlers. Each of these functions and their use is described in this section. Session Handling Functions As each Session is processed, the Fiddler invokes specially-named functions in your FiddlerScript. If your script contains a function with the same name as any of the functions in this section, Fiddler will call that function at the appropriate time. Each function should be declared as a public static function. Main Function functions are immediately after the script is successfully compiled. It provides an opportunity to initialize variables in the script from the Preferences system, or perform other one-time initialization. OnPeekAtRequestHeaders The OnPeekAtRequestHeaders function runs as soon as the request headers have been read from the client. This function generally runs before the request body has been read, so avoid manipulating the oSession.requestBodyBytes array to avoid causing a null-reference exception. In practice, this function is rarely used, because you may instead use OnPeekAtResponseHeaders function has as soon as the response headers have been read from the server. This function generally runs before the response body has been read, so avoid manipulating the oSession.responseBodyBytes array to avoid causing a null-reference exception. One of the best uses of this function is to examine the response headers to determine whether or not the response may be streamed as it is read from the server. If the response should be buffered so that its body may be modified by the OnBeforeResponse function, set oSession.bBufferResponse=true inside this function. 169 | FiddlerScript When a Session has buffering disabled, the OnBeforeResponse function is called after the response has already been streamed to the client. Hence, if you modify the previously-streamed response in this function, only Fiddler will see the modified response—the client already got the unmodified original. OnReturningError function executes on the server's that a Fiddler-generated error message (for instance, "DNS Lookup Failed") is returned to the client. This provides the opportunity to customize the message that the client application sees. For instance, you could show a more-friendly error page, log the error in a database, flag the error in the Web Sessions list, or undertake other actions. General Functions FiddlerScript includes a number of functions which run based on events in Fiddler itself, and not in response to any given session. Each function should be declared as a public static function. Main This function runs immediately after the script is successfully compiled. It provides an opportunity to initialize variables in the script from the Preferences system, or perform other one-time initialization. OnBoot This function runs only when Fiddler first boots. This function's script is changed at runtime, the new code will not execute until the next time you start Fiddler. OnShutdown This function runs when Fiddler is shutting down. In practice, there's rarely a need to put any code in this function. OnAttach This function runs after Fiddler registers as the system proxy. By default this will happen when Fiddler shuts down, and if the user ever disables capturing using the status bar button or by pressing the F12 key. OnDetach This function runs after Fiddler has unregistered as the system proxy and the default proxy is restored. By default this will happen when Fiddler shuts down, and if the user ever disables capturing using the status bar button or by pressing the F12 key. OnExecAction(sParams: string[]) This function runs when the user enters a command in the QuickExec box beneath the Web Sessions list, or if a command is sent to Fiddler from the ExecAction.exe command line program. The single parameter passed to this function is an array of strings which are tokenized from input to the QuickExec box. The OnExecAction function is 170 | FiddlerScript called after all extensions which implement the IHandleExecAction interface have the opportunity to handle (and optionally cancel) the command. The default implementation of this function shows how you can use it to script various actions, run functions, or otherwise control Fiddler's behavior. 171 | FiddlerScript FIDDLE RSCRIPT AND AUTOMATION TOOLS Fiddler was designed primarily for interactive use, where you directly use the Fiddler user-interface to perform whatever debugging tasks you'd like to accomplish. However, Fiddler includes a limited set of features to support automated use. Quiet Mode First is the -quiet command line argument. When Fiddler is activated with this switch, the main Fiddler window is automatically minimized to the System Tray. Fiddler will automatically suppress most of its prompts and alerts when running in quiet mode; for instance, no Certificate Warnings or HTTPS Errors will pop up messages boxes in this mode. The CONFIG.QuietMode property can be respected by extensions or FiddlerScript to accommodate the quiet mode of operation. Driving Fiddler from Batch Scripts Of course, running Fiddler silently isn't usually enough to accomplish most automation goals—you also will need to feed input to direct its behavior. Depending on your framework, you might use UI Automation to drive Fiddler's existing interface directly, but there's a more robust and reliable option available. As we saw previously, the OnExecAction function in FiddlerScript is run whenever the user enters a command in the QuickExec box. However, that's not the only way to invoke script commands—you can also use the ExecAction.exe command line tool included with the default install of Fiddler. For instance, if you wanted to invoke the dump command to save all of the Sessions list to a SAZ file, you can simply run: "%PROGRAMFILES%\Fiddler2\ExecAction.exe" dump The utility will pass the command into Fiddler's extensions and FiddlerScript, and the default handler in the FiddlerScript file will perform the specified action. If you need to pass multiple parameters into the tool, you can do so by quoting the arguments: "%PROGRAMFILES%\Fiddler2\ExecAction.exe" "prefs set x.y.z true" Note that you must enclose each entire argument in quotes. For instance, if you want to use the 'select' text/html" you may use "%PROGRAMFILES%\Fiddler2\ExecAction.exe" "prefs set \"MyExt.UI.Title\" \"A Quoted Value\"" If you pass an incorrect number of arguments to the utility (zero or more than one), it will display help text and set the ERRORLEVEL to 1. If the utility cannot find the Fiddler window, it will display a message indicating that Title" could not be found and set the ERRORLEVEL to 2. 172 | FiddlerScript Driving Fiddler from Native or .NET Code Under the covers, the ExecAction utility is extremely simple. If your test harness is written in C++ or any .NET language, you can easily pass messages directly to Fiddler without relying upon ExecAction.exe. Simply determine the main Fiddler window's handle, and then call SendMessage to pass the WM_COPYDATA structure containing the desired command's text. Using C++, the included "stdafx.h" int _tmain(int argc, _TCHAR* argv[]) { HWND hWndControl = FindWindow(NULL, L"Fiddler - HTTP Debugging Proxy"); if (NULL == hWndControl) { printf("ERROR: Fiddler window not found."); return 2; } COPYDATASTRUCT oCDS; oCDS.dwData = 0x4A; // Use 61180 for ANSI; 61181 for Unicode oCDS.cbData = lstrlen(argv[1]) + sizeof(WCHAR); oCDS.lpData = argv[1]; SendMessage(hWndControl, WM_COPYDATA, NULL, (WPARAM)&oCDS); return 0; } If you're using .NET code, you must somewhat differently than the items added to the Tools menu and Web Sessions context menu. Instead of specifying a method using an attribute, you will instead annotate a variable field which holds the desired boolean or string value; the script itself manages the value's storage. For instance, to add Hide 304s menu item, the following script will work somewhat differently than the items added to the Tools menu and Web Sessions context menu. Instead of specifying a method using an attribute, you will instead annotate a variable field which holds the desired boolean or string value; the script itself manages the value's storage. For instance, to add Hide 304s menu item, the constants for PInvoke: internal const int WM_COPYDATA = 0x4A; [DllImport("user32.dll", EntryPoint = "SendMessage")] internal static extern IntPtr SendMessage(IntPtr hWnd, int Msg, IntPtr wParam, ref SendMessageStruct lParam); [DllImport("user32.dll")] internal static extern IntPtr FindWindow(string lpClassName, string lpWindowName); [StructLayout(LayoutKind.Sequential)] internal struct SendDataStruct { public int dwData; public int cbData; public string strData; } The code then consists of a single call to the FindWindow function, followed by a call to the SendMessage function: 173 | FiddlerScript SendDataStruct oStruct = new Utilities.SendDataStruct(); oStruct.dwData = (IntPtr) 61181; oStruct.cbData = Encoding.Unicode.GetBytes(strData).Length; IntPtr hWnd = FindWindow(null, "Fiddler - HTTP Debugging Proxy"); SendWMCopyMessage(hWnd, Utilities.WM_COPYDATA, IntPtr.Zero, ref oStruct); You can use this technique to perform lightweight and ad-hoc automation of Fiddler. For more complicated jobs, you will likely instead choose to build a custom tool based on the FiddlerCore class library described in the final chapter of this book. 174 | FiddlerScript E X T E N D I N G F I D D L E R ' S U I - M E N U S When Fiddler completes your script, it looks for attributed methods that are used to create new entry-points in the Fiddler user-interface. You can add new items to the Rules menu, Tools menu, and Web Sessions context menu, as well as adding new top-level menus of your own. Extending the Tools Menu You can add new items to the Tools menu by adding a ToolsAction attribute to a method in your script. The attribute specifies the text of the item to show on the menu, and it immediately precedes the method that will be run if the user clicks on the menu item. For instance, the following block, when placed inside your Handlers class, will add an item titled Launch Opera to the Tools menu: // Launch Opera ToolsAction("Launch Opera") public static void DoLaunchOpera() { System.Diagnostics.Process.Start("opera.exe"); } String Empty.) Your method's signature may optionally specify a single parameter which will contain the text of the Web Sessions list when the menu item is invoked. // Launch Opera to &URL public static function ToolsAction("Launch Opera to &URL") public static void DoLaunchOpera(oS: Session[]) { if (oS.Length > 0) { System.Diagnostics.Process.Start("opera.exe", oS[0].fullUrl); } } If the user clicks on your menu when no items are selected, the oS array will have a Length property value of 0. To specify an accelerator key for your menu item, include an ampersand (&) in its name. In the previous example creates the following menu item with the U key as an accelerator: 175 | FiddlerScript Extending the Web Sessions Context Menu You may similarly extend the Web Sessions context menu by adding a ContextAction attribute to a method in your script. For instance, the following will decompress and unchunk the HTTP request and response for each selected session: ContextAction("Decode Selected Sessions") public static function DoRemoveEncoding(oS: Session[]) { for (var x=0; x < oS.Length; x++) { oS[x].utilDecodeRequest(); oS[x].utilDecodeResponse(); } } The new context menu items created by your script are added to the bottom of the menu. Extending the Rules Menu Your script may add new entries to the Rules menu, but they work somewhat differently than the items added to the Tools menu and Web Sessions context menu. Instead of specifying a method using an attribute, you will instead annotate a variable field which holds the desired boolean or string value; the script itself manages the value's storage. For instance, to add Hide 304s menu item, the following script will work: RulesOption("Hide 304s") public static var m_Hide304s: boolean = false; Two types of fields in your script may be bound to Rules menu items: boolean, and strings. Note: The magic value %CUSTOM% used for the "Ask me...." menu item, the UI and the field is one-directional. That means that when the user clicks a field-bound menu item, the value of the bound boolean or string is changed, but if the script itself changes the boolean's or string's value, no change is made to the menu item's state. Boolean-bound Rules When Fiddler encounters a boolean variable in the script that is annotated with a RulesString attribute, a menu item will be created. Every time your script is recompiled, the default values for these variables will be restored. In the prior example, the boolean's value is false, so the menu item will be unchecked by default. When the user clicks the menu item, it will become checked When the user clicks the menu item, it will become checked and the boolean value will change the state of the menu item. The script itself may change the state of menu items by setting the bound boolean's value. In the prior example, the boolean's value is false, so the menu item will be unchecked by default. When the user clicks the menu item, it will become checked and the boolean value will change the state of the menu item, the UI will create a new function called DoHide304s Instead of annotating it, a menu item titled Hide 304s will be restored. In the prior example, the boolean's value is false, so the menu item will be unchecked by default. When the user clicks the menu item, it will become checked When the user clicks a field-bound menu item, a boolean variable in the script that is annotated with a RulesString attribute, a new function 177 | FiddlerScript RulesOption("Some other setting", "MyMenu", false) public static var m_OtherSetting: boolean = true; String-bound Rules When Fiddler encounters a string in the script that is annotated with a RulesString attribute, a new submenu item will be created. The third parameter to the RulesString constructor is a string containing the submenu's name, while the second parameter is a boolean indicating whether the item labeled Disabled should be added to the submenu. Possible values for the string's value are provided by adding one or more RulesStringValue attributes to the same variable. Each RulesStringValue attributes to the same variable. When the user checks the exclusive menu item on the menu items, the variable will be updated accordingly. The default value of the string is set when the menu is created, but any RulesStringValue attribute annotated with a default value of true will set the initial value. RulesString("MyMenuText1", "MyValue1") RulesString("MyMenuText2", "MyValue2") RulesStringValue("MyMenuText3", "MyValue3") public static var m_StringRule: String = "String-bound Rules When Fiddler encounters a ...creates the following menu: By default, the menu's items will be sorted alphabetically by the menu's text. In some cases, you may wish to specify a default order. To do so, use the RulesStringValue overload that accepts an integer ordering parameter. For instance, you can create a menu that looks like this: ...using this script: RulesString("MyStringRule", true) 178 | FiddlerScript RulesStringValue(1, "First Item", "one") RulesStringValue(2, "Second Item", "two"); RulesStringValue(3, "Third Item", "three") RulesStringValue(5, "Ask me...", "%CUSTOM%") public static var m_StringRule: String = "two"; Should you wish to use one of the entries as the default, you can set the menu item's initial string value by default, the menu's items will be sorted alphabetically by the menu's text. In some cases, you may wish to prompted to enter a value for the present is shown, the current value will be presented for the user to modify. Fiddler's Rules > User-Agents feature is implemented using a variable named sUA bound to a RulesString attribute. To add a User-Agent menu item that specifies a given string, you would use code like this: RulesString("User-Agents", true) public static var sUA: String = null; 179 | FiddlerScript When the user clicks on the FiddlerCore item, the DoLinksMenu function will run, passing in the URL associated with that function. Items in the menu are sorted alphabetically by menu text. QuickLinkItem("&Opera", "Opera.exe") QuickLinkItem("&Chrome", "Chrome.exe") public static function DoBrowsersMenu(sText: String, sAction: String) { } ... Similarly, you could use a switch() statement to dispatch command invocations for handling by other functions. 180 | FiddlerScript E X T E N D I N G F I D D L E R ' S U I - A D D I N G C O L U M N S LIST TO THE WEB SESSIONS LIST Binding Columns using Attributes The BindUIColumn attribute is used to create a new column in the Web Sessions list and bind to it to a string function. The method must accept a single parameter of type Session and return a String. For instance, the following script adds a column to the list that shows the HTTP Method for each Session: BindUIColumn("Method", 60) public static function FillMethodColumn(oS: Session) { if ((oS.oRequest == null) || (oS.oRequest.headers == null)) return oS.oRequest.headers.HTTPMethod; } return String.Empty; } After this function is added to the script, a new Method column is added to the Web Sessions list when the script is compiled. 181 | FiddlerScript complexity of the computation of the value increases, so too does the time required to display the column. For instance, if the calculation involves decoding the response body, you should take care to ensure that it runs as quickly as possible. One strategy to minimize the work of this function is to cache values using a new flag on the Session object. Obviously, the benefit of caching a computed value will increase as the time required to compute that value increases. For instance, if the calculation involves decoding the response body, you should take care to ensure that it runs as quickly as possible. The following is an example which converts a non-binary content (Utilities.IsBinaryMIME(oS.oResponse.MIMEType)) { oS.oFlags["HasFuzzle"] = "n/a"; return "n/a"; } var s = oS.GetResponseBodyAsString(); var i = -1; if (s.Length > 0) { i = s.IndexOf("fuzzle", StringComparison.OrdinalIgnoreCase); if (i >= 0) { oS.oFlags["HasFuzzle"] = "Yes!"; return "Yes!"; } oS.oFlags["HasFuzzle"] = "Nope"; return "Nope"; } This example first checks the cache for a previously computed value, and if one is found, returns that value to the caller. If it returns an empty string value, the function begins its work of searching for the word "fuzzle" within the response body using a string and then searches it for the word "fuzzle". If the calculation is available for the "Ask me...." menu item. 182 | FiddlerScript the result and returns it. Finally, it converts the response body to a string and hunts for the target word. It caches and returns the result of the search. The complete set of BindUIColumn constructors includes versions that allow you to specify the order in which the new column is added to the list, and its behavior, and a flag function to which the column will be bound. The call BindObject.UI.lvSessions.AddBoundColumn("ClientPort", 50, "X-CLIENTPORT"); ...adds a new column with the title ClientPort and a default width of 50 pixels. The contents of the column are bound to the value of each session's X-ClientPort flag, which will be empty if the flag does not exist. The third parameter allows you to specify either a Session Flag name, or the name of a request or response header using the @request or @response prefixes. For instance: FiddlerObject.UI.lvSessions.AddBoundColumn("Server", 50, "@response.Server"); ...will create a column

with the title Server, displaying the contents of the response's Server header, if present. Similarly, FiddlerObject.UI.lvSessions.AddBoundColumn("Reason", 50, "@request.X-Download-Initiator"); ...will create a column with the title Reason, displaying the request's X-Download-Initiator header's value, if present. You can configure IE9+ to emit this header on each request; it provides a terse explanation of why a given request was made. For instance, it will indicate if the request was caused by navigation, an IMG tag's download, etc. See to show configuration scripts that enable and disable sending of this header. You may also call AddBoundColumn and provide a callback function's name, along with a boolean indicating that the column's values should be interpreted and sorted as numbers. For instance: 183 | FiddlerScript // Callback function that returns the "Time to first byte" value static function getTTLB(oS: Session): String { var iMS = Math.round((oS.Timers.ServerDoneResponse - oS.Timers.FiddlerBeginRequest).TotalMilliseconds); if (iMS > 0) return iMS.ToString(); return 0; } FiddlerObject.UI.lvSessions.AddBoundColumn("TTLB", 0, 50, true, getTTLB); The true parameter indicates that the column's values are numeric, while the final parameter indicates that the data should be inserted as the first column displayed in the Web Sessions list. You can set the display order parameter is relative, not absolute, so calling: FiddlerObject.UI.lvSessions.AddBoundColumn("#", 0, 0, 0, 0, 40, 30, 20, func4); func3); func2); func1); ... will result in four new columns of increasing width, 1, 2, 3, and 4, in that order from left to right. If you'd like to change a column's width or ordering within the Web Sessions list, you can use the SetColumnOrderAndWidth method. The method takes three parameters: the title of the column to adjust, the new position, and the new width. The latter two parameters may be set to -1 to leave the order or width unchanged. For instance, to move the Protocol column to the far left of the Web Sessions list without changing its width, call: FiddlerObject.UI.lvSessions.SetColumnOrderAndWidth("Protocol", 0, -1); Again, keep in mind that column ordering behaves as an insertion operation. So, if you call: FiddlerObject.UI.lvSessions.SetColumnOrderAndWidth("#", FiddlerObject.UI.lvSessions.SetColumnOrderAndWidth("Result", FiddlerObject.UI.lvSessions.SetColumnOrderAndWidth("Protocol", FiddlerObject.UI.lvSessions.SetColumnOrderAndWidth("Host", 0, 1, 0, 0, -1); -1); -1); The final ordering is (Host, Protocol, #, Result), because the Host column was inserted at index 0, pushing the Result and Protocol columns rightward. 184 | FiddlerScript FIDDLE ROBE CT FUNCTIONS The FiddlerScript engine itself provides a number of simple utility functions and properties that you may use to accomplish common tasks. These methods and properties are exposed by the FiddlerObject object and include: FiddlerObject.ReloadScript() This method instructs Fiddler to reload the script file from disk and recompile it, resetting all variables to their default values. FiddlerObject.StatusText This property allows you to set the text on Fiddler's status bar. It may safely be set from any thread, as it will internally call BeginInvoke on the UI thread if set from a background thread. The default FiddlerScript file sets the status bar text when the script is first compiled: static function Main() { var today: Date = new Date(); FiddlerObject.StatusText = " CustomRules.js was loaded at: " + today; } The OnExecAction handler in the default script uses the StatusText property to communicate the results of commands to the user. FiddlerObject.log(sTextToLog) This method allows you to log text to Fiddler's event log. The first character of the provided text may be an exclamation mark, underscore, or forward slash to cause the Log tab to format the text in bold, underlined, or italic text respectively. The following example: static function OnPeekAtResponseHeaders(oSession: Session) { FiddlerObject.log("!Boldred!" + oSession.id.ToString()); FiddlerObject.log("_Underlined#" + oSession.id.ToString()); FiddlerObject.log("/Italicized/" + oSession.id.ToString()); } ... will log the following output: 185 | FiddlerScript FiddlerObject.playSound(sSoundFilename) This method loads and plays the specified .wav audio file. This can be useful if your script needs to draw the user's attention to an exceptional event. FiddlerObject.alert(sMessage) This method displays the provided message in a message box with the title "FiddlerScript." This method and its overloads FiddlerObject.prompt(sMessage) This method and its overloads FiddlerObject.prompt(sMessage, sDefaultValue) FiddlerObject.prompt(sMessage, sDefaultValue, sTitle) ...allow you to prompt the user for a string using the prompt message provided in the first parameter. The second and third overloads allow you to optionally specify a default value and the title of the prompt dialog box. For instance, this line will prompt the user for a HTML tag name: var sTagName: String = FiddlerObject.prompt("Enter a HTML tag name", "", "Specify Tag"); When the prompt is displayed, Fiddler will flash its window's entry in the Windows task bar to help draw the user's attention. If the user presses Cancel on the prompt, the supplied default value (or String.Empty) will be returned. FiddlerObject.createDictionary() This method allows you to create a simple Dictionary object suitable to pass to the DoImport and DoExport methods on the FiddlerApplication object. This method is exposed because the JScript.NET language does not expose any other way to create a generic Dictionary object. The following example generates a HAR file on the desktop containing all of the captured sessions: var oSessions = FiddlerApplication.UI.GetAllSessions(); var oExportOptions = FiddlerObject.createDictionary(); oExportOptions.Add("MaxTextBodyLength", 0); oExportOptions.Add("MaxBinaryBodyLength", 0); FiddlerApplication.DoExport("HTTPArchive v1.2", oSessions, oExportOptions, null); 186 | FiddlerScript FiddlerObject.WatchPreference("fiddler.script", ObserverPrefChange); } static function ObserverPrefChange(oSender: Object, oPCEA: PrefChangeEventArgs) { var sMsg: String = oPCEA.PrefName + " changed to " + oPCEA.ValueString; which will be notified when a change occurs within the specified branch of Preferences. static function Main() { FiddlerApplication.Prefs.WatchPrefBranch("fiddler", ObserverPrefChange); } C:\users\ericlaw\desktop\out1.har"); oExportOptions.Add("C:\users\ericlaw\desktop\out1.har"); oExportOptions.Add("MaxTextBodyLength", 0); oExportOptions.Add("MaxBinaryBodyLength", 0); FiddlerApplication.DoExport("HTTPArchive v1.2", oSessions, oExportOptions, null); 186 | FiddlerScript FiddlerObject.WatchPreference("fiddler.script", oFunc) This method allows you to provide a callback function which will be notified when a change occurs within the specified branch of Preferences. static function ObservePrefChange(oSender: Object, oPCEA: PrefChangeEventArgs) { var sMsg: String = oPCEA.PrefName + " changed to " + oPCEA.ValueString; REFERENCING ASSEMBLIES The .NET Framework offers a huge library of functionality that you can take advantage of in your FiddlerScript. For instance, you can use the classes in System.Data to log traffic to a database, or use classes in the System.IO namespace to write files to disk. To reference an assembly that is installed in the system's Global Assembly Cache, simply add the appropriate namespace to the list of #import statements at the top of your script: Then use the classes as needed: You can also use methods in your own assembly DLLs even if those assemblies are not loaded onto the global assembly cache. There are two ways to enable FiddlerScript to find the assembly. The first is to add the Assembly to your built-in tools > Fiddler Extensions > Extensions: Alternatively, you may use a Preference to specify the default library path used to find assemblies. The Preference fiddler.script.LibPath should be set to the fully-qualified path that contains your DLLs. For instance, type the following in the QuickExec box below the Web Sessions list: prefs set fiddler "C:\src\Library DLLs!" ...and when your FiddlerScript is next compiled, Fiddler will look for referenced assemblies in the Library DLLs folder. 188 | FiddlerScript EXAMPLE SCRIPTS While you can use FiddlerScript to build nearly any type of functionality you want, in most cases you will use script to modify either the request or response. This section contains a few examples to get you started. Request Scripts Place these snippets inside the OnBeforeRequest function so that they run before the request is sent to the server. Add (or Overwrite) a Request Header You can modify request headers very easily using the indexer property on the oRequest object: oSession.oRequest["HeaderName"] = "New value"; This will add the specified header, or overwrite the first existing header named HeaderName. To add the Session's ID to the outbound request headers: if (oSession.oRequest.headers.Exists("Cookie")) { oSession["ui-color)= "red"; oSession["ui-customcolumn"] = oSession.oRequest["Cookie"]; } Rewrite a Request from HTTP to HTTPS You can modify the UriScheme property for requests easily. The following code sets the UriScheme property of a request whose URL doesn't match its Host header, the original Host value is stored in the session flag X-Original-Host and then the Host value is parsed from the URL. The following script reverses behavior by reusing the request to the host specified by the original Host header. if (oSession.BitFlags & SessionFlags.ProtocolViolation) { var sOverride = oSession["X-original-Host"]; if (!String.IsNullOrEmpty(sOverride)) { oSession["X-overrideHost"] = sOverride; oSession["ui-backcolor"] = "yellow"; // Save to bypass the gateway, otherwise overrideHost doesn't work oSession.bypassGateway = true; } } Drop a Connection FiddlerScript can drop the client's connection. Set the oResponder Actions *drop and *reset. if (oSession.uriContains("DropIt") && (null != oS.oRequest.pipeClient)) { // Use this to close the connection using TCP/IP RST oS.oRequest.pipeClient.End(); // So that the UI shows what we did, create a placeholder / response for display: if (this.state = SessionStates.SendingRequest) { oS.utilCreateResponseAndBypassServer(); } oS.oResponse.headers.HTTPResponseStatus = "0 Connection dropped by script"; oS.responseBodyBytes = new byte[0]; oS.state = SessionStates.Aborted; return; } Prevent Response Streaming To ensure that the server's response is buffered to permit tampering, you must set the Session object's bBufferResponse field before the response is read from the server: if (oSession.HostnameIs("mySever") && oSession.uriContains(".aspx")) { oSession.bBufferResponse = true; } 190 | FiddlerScript Response Scripts You can modify responses just as easily. If your code only needs to access the response's headers, use the OnPeekAtResponseHeaders function. If you need access to the response body, place your code inside the OnBeforeResponse function. There, the script will run after the response body is read from the server but before the response is returned to the client. That is, unless buffering is disabled, in which case the function will run after the response has been streamed to the client, hiding any modifications you make to the body from the client. Hide Sessions that Returned Images You can set the ui-hide flag to hide sessions whose response is an image. For instance: if (oSession.oResponse["Content-Type"].StartsWith("image/")) { oSession["ui-hide"] = "hiding images"; } Flag Redirections Examine the responseCode property to identify redirects: if (oSession.responseCode >= 299 && oSession.responseCode < 400) { oSession["ui-backcolor"] = "green"; } Decode a Response You can decode a response easily: oSession.utilDecodeResponse(); Replace Text in Script, CSS, and HTML Change any CSS vendor-prefixes from -Moz- and -WebKit- to -MS-: if ((oSession.IsBinaryMIMEType(oSession.oResponse.MIMEType) || (oSession.GetResponseBodyAsString(); // Replace all content of DIV tags with an empty string. // WARNING: Doesn't work well with nested DIVs. var oRegEx = /]*?>/gi; oBody = oBody.replace(oRegEx, ""); // Set the response body to the div-less string oSession.utilSetResponseBody(oBody); } 191 | FiddlerScript More Examples You can find more FiddlerScript samples online. In the SampleRules.js file installed with Fiddler At 192 | FiddlerScript Extending Fiddler with .NET Code EXTENDING FIDDLER WITH .NET While FiddlerScript provides a simple and lightweight means of extending Fiddler, extending Fiddler offers significantly more power and an improved debugging experience. That's especially true if you plan to distribute your logic to other users. Extending Fiddler with .NET enables you to build your extension in any .NET language (C# and VB.NET are the most common choices) and take advantage of the powerful IntelliSense and debugging features of Visual Studio. If your extension limits itself to the classes in the v2.0 Framework, it should work properly on every Fiddler user's system. While both FiddlerScript and .NET extensions may use any of the classes in the .NET Framework, it can be cumbersome to use many Framework classes from FiddlerScript. Compiled .NET extensions typically exhibit significantly faster load-time and run-time performance as compared to building equivalent functionality in FiddlerScript, because the latter must be recompiled each time Fiddler starts. Extensions can be installed to users' machines without the hassle of manual modifications of the user's FiddlerScript source file, and can be installed using the system's Control Panel. Each extension's assembly declares the minimum version of Fiddler with which it is compatible, and users will be prompted to update if an extension requiring a later Fiddler version is encountered. This versioning feature helps prevent problems where code that works great on your computer fails to work because they're clinging to an outdated version of Fiddler. Project Requirements and Settings Writing extensions for Fiddler requires Visual Studio .NET 2008 or later, or the free .NET Framework v2 commandline compilers. Fiddler2 loads only .NET Common Language Runtime (CLR) v2.0 assemblies; assemblies that target the v1.1 or v4.0 CLR will not be loaded. Fiddler itself requires only that the user have .NET Framework 2.0 SP1 installed. Your extension may target the .NET Framework 3.5 (which includes LINQ and other useful features) since that framework also runs on the v2.0 CLR. However, if it takes such a dependency, your installer must ensure that the user has the required Framework version installed, or the user will encounter "Class not found" exceptions as your extension loads. You should also ensure your project targets AnyCPU to ensure that Fiddler loads it when running on a 64bit system. Debugging Extensions To avoid annoying end-users with warnings that they cannot do anything with, Fiddler silently handles a variety of problems when it loads and calls into extensions. However, these warnings can be extremely useful to extension developers, pointing out problems that prevent extensions from functioning properly. 194 | Extending Fiddler with .NET Code Extension developers should set the fiddler.debug.extensions.showerrors preference to true ensure that exceptions and other extension-related errors are not silently caught. Additionally, they should set the fiddler.debug.extensions.verbose preference to true to gain logging information and warnings to Fiddler's Log tab. Best Practices for Extensions On the Internet Explorer team, we've found that over 75% of browser crashes and nearly all major performance problems are caused by buggy browser extensions. Fortunately, extension quality in the Fiddler ecosystem is much better, but it's still the case that an extension can cause Fiddler to crash or perform more slowly. I'm relying on you to help ensure that users have a great experience when your extension is running inside Fiddler. The simplest way to minimize your extension's performance impact is to ensure that it does not run when it's not being used. There are two key strategies for doing that: offer an enable switch, and use a delay-load pattern. Best Practice: Use an Enable Switch With very few exceptions, users should not be forced to uninstall your extension to disable it-- most users will only want your extension to run at certain times. Extensions should offer an Enable switch; when that switch is off, the extension should do as little work as possible. For instance, inside the ContentBlock extension, each method that handles a Session first checks to see whether blocking is enabled: public void AutoTamperRequestBefore(Session oSession) { // Remove item immediately if no rule is enabled if (bBlockerEnabled) return; // ... This best practice is followed by Fiddler itself-the AutoResponder and Filters tabs have enabling checkboxes at their top and only process Sessions when enabled. Best Practice: Use Delay Load Extensions and Inspectors are loaded as Fiddler starts up, so your code can directly impact the time that Fiddler takes to load. Because some users will restart Fiddler dozens of times per day, load time is a very important factor in their experience with the tool. In many cases, a user may not use your Inspector or Extension at all within a given session. For instance, perhaps the user installed your Silverlight-troubleshooter for one project, but then gets pulled away to go help with a HR system that uses only WCF Web Services and no Silverlight at all. If your Inspector uses a lot of memory and takes a long time to load, the user will be paying a penalty for having it installed even when it's not used. To avoid that problem, use the Delay Load pattern, in which expensive operations are only undertaken if needed. This pattern can be effectively applied to both Inspectors and other types of extensions. For instance, the FiddlerScript tab uses a syntax-highlighting component to display the source of your script file; this component requires hundreds of millions of cycles and several megabytes of memory to load. If this component was loaded each time Fiddler starts, the load cost would be wasted if the user doesn't interact with the tab. To mitigate this problem, the FiddlerScript tab only creates the syntax-highlighting component when the user first activates the extension's tab. The OnLoad handler for the extension creates its tab, adds a "Loading..." label to it, and adds a handler to the SelectedIndexChanged event to detect when its tab is being activated: public void OnLoad() { myPage = new TabPage("FiddlerScript"); myPage.ImageIndex = (int)Fiddler.SessionIcons.Script; this.lblLoading = new System.Windows.Forms.Label(); this.lblLoading.Text = "Loading..."; myPage.Controls.Add(lblLoading); FiddlerApplication.UI.tabsViews.TabPages.Add(myPage); TabControlEventHandler evtTCEH = null; evtTCEH = delegate(object s, TabControlEventArgs e) { if (e.TabPage == myTabPage) { // Create heavyweight components used to display UI EnsureReady(); // Remove the unneeded event handler. FiddlerApplication.UI.tabsViews.Selected = evtTCEH; } }; // Subscribe to tab-change events FiddlerApplication.UI.tabsViews.Selected += evtTCEH; } Only when the extension's tab is activated does the EnsureReady method run to create the heavyweight component, hiding the "Loading..." label to and, adds a "Loading..." label / Create the extension's UI view new RulesEditor(myPage); lblLoading.Visible = false; // Remove the "Loading..." label } This pattern is also used by the SyntaxView, HexView, and WebView Inspectors, all of which depend upon components that are expensive to construct. Other extensions do not bother with the Delay Load pattern if they use a lightweight controls and the complexity of the Delay Load pattern isn't necessary. 196 | Extending Fiddler with .NET Code Best Practice: Beware "Big Data" The user's web traffic may include giant responses (e.g. videos or other files that are hundreds of megabytes in size) that you might not have anticipated when building your extension. Requests can be quite large as well, but in practice, enormous downloads are much more common than large uploads. If your extension doesn't defend itself from "big data," your code could hang Fiddler when the user selects such a Session. There are several best practices to help prevent hangs: Ensure that your extension "fails fast" if it encounters a response of a type or size that it wasn't designed to handle. Avoid making unnecessary copies of data. In most places, Fiddler uses a "copy on write" architecture to prevent needless duplication of data, and your extensions should do the same. If your extension attempts to render data by replacing embedded nulls using the Unicode replacement character (0xFFFD, rendered as ◆), ensure that you limit how much data you attempt to show. Otherwise, rendering a binary file will result in massive memory use, because the embedded nulls will not terminate the text passed to a textbox or other UI control. Fiddler's default extensions and follow these best practices to help ensure that the debugger remains responsive even when dealing with huge responses. Best Practice: Use the Reporter Pattern for Extensions A common goal when building an extension is to display summary data about the currently selected Sessions in the Web Sessions list. For instance, the Statistics tab shows the timing data and size of the selected sessions, while the Timeline tab shows the start time and duration of each selected session. Extensions of this sort are called Reporters because they generate a report on the currently-selected Sessions. A naïve implementation of a Reporter would recalculate its data each time a session is selected or unselected. The problem with that approach is that if the user hits CTRL+A to select all of the sessions in the Web Sessions list (or CTRL+X to clear the Reporter will end up doing billions of redundant calculations whose results are never used. To avoid that problem, instead handle the FiddlerApplication.CalculateReport event. This event is dispatched as the user selects and unselects Sessions, but it is suppressed during certain bulk operations (like loading a SAZ file or clearing the Web Sessions list), ensuring that it is dispatched only after the bulk update is completed. The event is also suppressed when Fiddler shuts down, helping to ensure that you don't perform expensive work as Fiddler is unloading and disposing of its objects. Your event handler implementation must have the following signature: public delegate void CalculateReportHandler(Session[] arrSessions); 197 | Extending Fiddler with .NET Code Your event handler will be called at the appropriate intervals, and an array of the currently selected Sessions will be passed as the only parameter. You should still avoid doing any redundant work--for instance, if your tab isn't presently active, you should suppress calculations until it is shown. The Statistics tab's CalculateReport handler looks a bit like this: void CalculateReport(Session[] arrSessions) { // If we're not showing the Stats tab right now, bail out. if (FiddlerApplication.UI.tabsViews.SelectedTab != FiddlerApplication.UI.pageStatistics) { return; } try { if (arrSessions.Length < 1) { ClearStatsTab(); return; } Dictionary ResponseSizeByContentType; long cBytesRecv; string sStats = BasicAnalysis.ComputeBasicStatistics(arrSessions, true, out dictResponseSizeByContentType, out cBytesRecv); txtReport.Text = String.Format("{0}\r, sStats); } catch (Exception eX) { Debug.Assert(false, eX.Message); } } By using the CalculateReport event, your extension can avoid performing unnecessary work, helping to keep Fiddler running fast. 198 | Extending Fiddler with .NET Code INTERACTING WITH FIDDLER'S OBJECTS Most FiddlerScript extensions use Fiddler's object model to interact with the user-interface, Session objects, the FiddlerScript engine, and features like the TextWizard. This section outlines the properties, flags, fields, and events that your code may rely upon. The Web Sessions List Because the Web Sessions list is the most important component of the Fiddler user-interface, your code will often need to interact with its entries. Fiddler exposes methods on the FiddlerApplication.UI object to interact with the Web Sessions list. Session[] GetAllSessions() This method will return an array containing all of the Sessions in the Web Sessions list. If the Web Sessions list is empty, the array's Length will be 0. Session GetFirstSelectedSession() This method will return the first selected Session in the Web Sessions list, or null if no entries are selected. Session[] GetSelectedSessions() This method will return an array containing all of the Sessions that are currently selected in the Web Sessions list, up to the maximum number specified. If no entries are selected, the array's Length will be 0. void actSelectSessionsMatchingCriteria(matchCriteriaDelegate oDel) This powerful method allows you to supply a delegate function (similar to a predicate delegate) that is evaluated for each Session in the Web Sessions list. For each entry, if your delegate returns true, the Session's entry will be selected. For instance, say that you want to select all Sessions that use the POST request method. You can use the FiddlerApplication.UI.actSelectSessionsMatchingCriteria(delegate(Session oS) { return (200 == oS.responseCode); }); void actRemoveSelectedSessions() Removes all selected Sessions from the Web Sessions list. void actRemoveUnselectedSessions() Removes all unselected Sessions from the Web Sessions list. void actSelectSessionsMatchingCriteria(delegate(Session oS) { return oS.HTTPMethodIs("POST"); }); 199 | Extending Fiddler with .NET Code Or, if you want to select all Sessions that have a HTTP/200 response status, use: FiddlerApplication.UI.actSelectSessionsMatchingCriteria(delegate(Session oS) { return (200 == oS.responseCode); }); void actInvertSelection() Inverts the current selection state within the Web Sessions list. Sessions which were previously selected are cleared, and those which were not previously selected are selected. string actSessionArchiveZip(string sFilename) Loads the sessions from the specified SAZ file and adds them to the Web Sessions list. void actSaveSessionsToZip(string sFilename, string sPwd) Saves the currently selected Sessions to a SAZ file named by the provided filename. If the password parameter is non-null, the SAZ file will be encrypted using the specified password. void actSessionCopyAll() Copies the URLs of the currently selected Sessions to the clipboard, one per line. void actSessionCopySummary() Copies all of the text from the Web Sessions list view to the clipboard for each of the Sessions. This is copied in both tab-delimited text and HTML formats, suitable for pasting into other documents or spreadsheets. void actSessionCopyHeaders() Copies a terse listing of the selected Sessions to the Clipboard. For instance, when loading the Fiddler homepage, the data returned is: GET 302 Object moved to /fiddler2/ GET 200 OK (text/html) CONNECT 200 Connection Established () GET 200 | Extending Fiddler with .NET Code 200 OK (text/css) GET 200 OK (image/gif) int FiddlerApplication.UI.lvSessions.SelectedCount This property appears directly on the Web Sessions listview and returns the count of selected Sessions without copying those Sessions into an array as the GetSelectedSessions method does. SimpleEventHandler FiddlerApplication.UI.lvSessions.OnSessionsAdded By default, Fiddler queues updates to the list for a short period of time to help ensure UI responsiveness. This event fires whenever one or more Sessions is visibly added to the Web Sessions list. If you choose to provide a handler for this event, ensure that the handler does not perform any expensive computations, as doing so will block the UI thread and severely impair performance. Session Objects Each Web Session is represented by a Session object which exposes fields, properties, and methods which control how the Session is processed and record information about it. oRequest The oRequest object represents the client's request. It exposes key properties including the headers object which contains the Request's headers. The headers themselves may be accessed by the indexers on the oRequest Session objects: oSession.oRequest["HeaderName"] = oSession.oRequest.headers["HeaderName"] = oSession["REQUEST", "HeaderName"]; The oRequest object's pipeClient field, if set, represents Fiddler's connection to the client's request. Because the body may be encoded, typically the GetRequestBodyAsString and utilSetRequestBody methods are used instead of manipulating the byte array directly. oRequestBodyBytes This byte array contains the raw bytes of the client's request. Because the body may be encoded, typically the GetRequestBodyAsString and utilSetRequestBody methods are used instead of manipulating the byte array directly. response It exposes key properties including the headers object which contains the Response's headers. The headers themselves may be accessed by the indexers on the oResponse Session objects: oSession.oResponse["HeaderName"] = oSession.oResponse.headers["HeaderName"] = oSession["RESPONSE", "HeaderName"]; The oResponse object's MIMEType property returns the MIME-type portion of the response's Content-Type header. The object's pipeServer field, if set, represents Fiddler's connection to the server. 201 | Extending Fiddler with .NET Code oResponseBodyBytes This byte array contains the raw bytes of the server's response. The body may be encoded, typically the GetResponseBodyAsString and utilSetResponseBody methods are used instead of manipulating the byte array directly. oFlags This StringDictionary contains flags that describe the Session. A full list of Session flags can be found in Appendix C. You may access this collection directly, or via one of the two indexer properties exposed by the Session object. oSession.oFlags["FlagName"] = oSession["FlagName"] = oSession["SESSION", "FlagName"]; Note: If the requested Flag does not exist, the indexers have slightly different behavior: if (oSession.oFlags.ContainsKey("SomeKey") { Debug.Assert(oSession["SomeKey"] = null); Debug.Assert(oSession["SomeKey"] == null); Debug.Assert(oSession["Session", "SomeKey"] = String.Empty); } void Abort() Aborts processing of the current Session. This closes both client and server connections (if any) and moves the Session's state to Aborted. bIsRequestStreaming This boolean field controls whether the request will be streamed to the client as it is read from the server. bHasResponse This readonly property returns true if the Session's state > ReadingResponse and oResponse is not null. bypassGateway This boolean field controls whether the request will be sent to the upstream gateway server, if any such server is configured. clientIP This string records the IP address from which the request originated. clientPort This integer records the port from which the request originated. bool COMETPeek() Call this method while in the ReadingResponse state to update the responseBodyBytes array with the partiallyread response. 202 | Extending Fiddler with .NET Code fullUrl This property stores the complete request URI, including protocol/scheme, and returns true if any of the specified flags are set, and false otherwise. To retarget this request to a different URL, adjust this property. PathAndQuery This property contains the path and query string from the request URL, port This property contains the port to which the request is targeted. host This property contains the host portion of the request headers; this value will never include a port number. hostname This string method contains the supplied hostname to the hostname of the request, returning true if a case-sensitive match is found. hostnameIs(string) This method searches the request URI for the supplied string, and returns true if a case-sensitive match is found. id This readonly integer property returns the Session's automatically-generated identification number. isFTP This readonly property returns true if the Session targets the FTP protocol. isTunnel This readonly property returns true if the Session targets a CONNECT tunnel used for HTTPS or WebSocket traffic. LocalProcessID This readonly integer property returns the process ID of the local process from which the request originated. If the request was generated from a remote client, this property will return 0. 203 | Extending Fiddler with .NET Code bool utilDecodeResponse() Call this method to remove any HTTP compression or Chunked Encoding from the response body. responseCode This integer property contains the HTTP status code from the server's response. state Each Session's state property tracks its progress through the following SessionStates: Created ReadingRequest AutoTamperRequestBefore HandTamperRequest AutoTamperRequestAfter SendingRequest ReadingResponse AutoTamperResponseBefore HandTamperResponse AutoTamperResponseAfter SendingResponse Done Aborted Description The Session has been created but nothing's happening yet. Fiddler is reading the request from the client. Fiddler extensions that provide an AutoTamperRequestBefore method are processing the request. The user may manually edit the request using the Fiddler Inspectors. Fiddler extensions that provide an AutoTamperRequestAfter method are processing the request. Fiddler is sending the request to the server. Fiddler is reading the response from the server. Fiddler extensions that provide an AutoTamperResponseBefore method are processing the response. The user may manually edit the response using the Fiddler Inspectors. Fiddler extensions that provide an AutoTamperResponseAfter method are processing the response. Fiddler is sending the response to the client. The Session is complete. The Session was aborted (the client disconnected before retiring reading the response, there was a fatal error in processing, etc.) The Session moves from one state to the next until it reaches the Done state; in the event of an error, the Session jumps immediately to the Aborted state. The Session object's OnStateChanged event fires any time the Session's state property changes. BitFlags Each Session's BitFlags property holds zero or more SessionFlags that supply commonly-queried state information. The SessionFlags enumeration includes: SessionFlags None Description No flags are set. isHTTPS The request's URI has a HTTPS target. isFTP The request's URI has a FTP target. ClientPipeReused The request was read from a previously used connection from the client. ServerPipeReused The request reused an existing connection to the server. 204 | Extending Fiddler with .NET Code ResponseStreamed The response was not buffered and was 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 the uitlCreateResponseAndBypassServer 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 decrypts HTTPS traffic as it flows through. ServerFromCache The Protocol ViolationInResponse The response was served from a client cache, bypassing Fiddler. Fiddler only "sees" this Session if either software reports it to Fiddler. ProtocolViolationInResponse There was a HTTP Protocol violation in the server's response. ResponseBodyDropped The response body was dropped, 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. The Session object exposes two methods to evaluate the Session's BitFlags. The isAnyFlagSet method accepts a set of flags and returns true if any of the specified flags are set, and false otherwise. The isFlagSet method accepts one or more flags and returns true if all of the specified flags are set, and false otherwise. For example: bool bAnalyzeBits = oSession.isFlagSet(SessionFlags.ResponseStreamed); Timers Fiddler records timestamps can help shed light on performance issues in your website or application, and can be seen in the Session's Properties window or the Statistics tab. 205 | Extending Fiddler with .NET Code Session Timers ClientConnected Description Timestamp at which the client's connection to Fiddler was first established. ServerBeginRequest ServerGotRequest ServerDoneResponse FiddlerBeginRequest ServerConnected GatewayDeterminationTime DNSTime TCPConnectTime HTTPSHandshakeTime Timestamp at which the client's connection to Fiddler was first established. 205 | Extending Fiddler with .NET Code ClientBeginRequest FiddlerGotRequest ClientDoneRequest ClientBeginResponse GatewayDeterminationTime Timestamp at which the request's first socket Send() operation was serviced by Fiddler. If this request was issued on a reused connection, this timestamp could be much later than the ClientConnected value. Timestamp at which Fiddler completed reading the headers from the client. Timestamp at which Fiddler completed reading the entire request from the client. The number of milliseconds spent determining the IP address of the remote host. Due to caching of DNS records, this value is often 0 for all but the first connections to a host. The number of milliseconds spent waiting for TCP/IP connection establishment to the server. This value will be 0 if Fiddler is not configured to decrypt HTTPS traffic. If decryption is enabled, when a CONNECT tunnel is being used for HTTPS traffic, Fiddler will populate this value. Timestamp at which the TCP/IP connection to the server or upstream gateway proxy was established. Timestamp at which Fiddler began sending the request to the server. If this request is being sent on a reused server connection, this timestamp could be much later than the ServerConnected value. Timestamp at which Fiddler read the first packet of the server's response. Timestamp at which Fiddler completed reading the entire response from the server. Timestamp at which Fiddler completed sending the response to the client. If buffering is disabled for this request, the response is returned from a client cache, bypassing Fiddler. Fiddler captures timestamps using the default system clock, which typically has a precision of 15.6 milliseconds. If a given operation takes less than 15.6ms to complete, it may appear that the operation took no time at all. Fiddler has not been updated to take advantage of high-precision timers because any task for which that level of precision is required should be accomplished via other means. The Observer Effect of introducing the Fiddler proxy between the client and server means that several high-level encodings. If the encoding is expected and understood, your extension may want to encode and decode the text itself--for instance, the WebForms Inspector will encode and decode form fields automatically. In other cases, however, the encoding may not be known in advance. In such cases, it can be helpful to TextWizard command which will send text to Fiddler's TextWizard tool. Logging Fiddler's Log tab collects logged message strings that are generated by extensions, FiddlerScript, and the following API: FiddlerApplication.UI.actShowTextWizard(string sDefaultInput) ... passing the selected text in the sDefaultInput parameter. An instance of the TextWizard will open, enabling the user to easily encode or decode the text. Logging Fiddler's Log tab collects logged messages including application events (e.g. when a SAZ file is saved or loaded) as well as system events (e.g. when the system's network connectivity is lost or restored). An extension can log text to the Log tab using either of the following two methods: FiddlerApplication.Log.LogString(string sMsg) FiddlerApplication.Log.LogFormat(string sFormat, params object[] args) These methods may be safely called from any thread and can be called before Fiddler itself has loaded; the Log object will collect any messages and publish them when a Log listener is first attached. If you are writing an extension of a Logger-Core-based application and wish to receive Logged messages, add a handler to the FiddlerApplication.Log.OnLogString event. Your handler will be passed the logged string as the LogEventArgs object. public event LogEventHandler OnLogString; public void LogString(string sMsg); public string LogFormat(string sFormat, params object[] args) { // Keep in mind that this event will often fire from background threads, so your handler must invoke BeginInvoke onto the UI thread if any visual updates are required. Fiddler's Log tab uses a very simple formatting mechanism when interpreting the strings it receives in its OnLogString handler: 207 | Extending Fiddler with .NET Code Strings prefixed with a bang (!) render in a bold font. Strings prefixed with a forward slash (/) render in an italic font. Your extension can also use simple markup features to draw attention to messages logged to the Log.Clear will clear the Log tab's contents. Logging the string @Log.Export filename will save the Log tab's text to the specified filename. Use a filename ending in .rtf to save in Rich Text Format, preserving text color and weight. Interacting with the FiddlerScript Engine Extensions may subscribe to events related to the compilation of FiddlerScript. For instance, this allows the FiddlerScript extension to capture and display compilation errors. The FiddlerApplication.scriptRules object exposes three events, two of which will fire as a script file is loaded and either compiles successfully or encounters an error: public event RulesBeforeCompileHandler BeforeRulesCompile; public event RulesAfterCompileHandler RulesAfterCompile; public event RulesCompileFailedHandler RulesCompileFailed; Because the script engine was developed before I knew how to use EventArgs properly, each event has its own custom delegate: public delegate void RulesBeforeCompileHandler(string sFilename); public delegate void RulesAfterCompileHandler(); public delegate void RulesCompileFailedHandler(string sDescription, int iLine, int iStartColumn, int iEndColumn); The BeforeRulesCompile event fires when Fiddler is instructed to load a new script file. The event handler delegate contains the name of the script file that is to be loaded and compiled. The AfterRulesCompile event fires if compilation succeeds against a script file, runs its Main function, and updates the Fiddler UI based on the contents of the script. The event handler delegate has no parameters. The RulesCompileFailed event fires if Fiddler encounters an error when compiling the script file. The event handler delegate has four parameters: the human-readable description of the problem, the line number of the error within the script file, and the starting and ending column on that line. If you plan to handle this event, please be aware that RulesCompileFailed usually fires very shortly after BeforeRulesCompile event. If your extension loads the 208 | Extending Fiddler with .NET Code new script's text in its handler for the BeforeRulesCompile event, it's best in its handler for the first event, ensure that you have successfully updated your copy of the script text before attempting to display the error information to the user. 209 | Extending Fiddler with .NET Code PROGRAMMING WITH P REFERENCES Fiddler includes a Preferences system which is accessible to FiddlerScript and extensions. The system is designed to be somewhat similar to the system used in Firefox (e.g. type about:config in Firefox to view its preferences). Fiddler's Preferences are automatically serialized to and deserialized from the registry. Preference Naming Keep the following in mind as you name your preferences: Preference names are case insensitive. Preference names must contain ASCII A-to-Z, numbers, dots, and dashes only. Preference names must be between 1 and 255 characters in length. Preference names improve performance using ephemeral will not be saved to or loaded from the registry. Preference internal names cannot be created or updated by extensions or script. You should name your preferences using a logical hierarchy, of the format: myCompany.myExtension.Feature.SettingName Careful naming makes it simpler to find your Preferences and improves performance for changes made (e.g. when subscribing to notifications). The FiddlerPreferences Interface Fiddler's preferences are accessed using FiddlerApplication.Prefs, an object which implements the IFiddlerPreferences interface. public interface IFiddlerPreferences { // Indexer string this[string sName] { get; set; } // Setters void SetBoolPref(string sPrefName, bool bValue); void SetInt32Pref(string sPrefName, Int32 iValue); void SetStringPref(string sPrefName, string sValue); void RemovePref(string sPrefName); // Getters bool GetBoolPref(string sPrefName, bool bDefault); string GetStringPref(string sPrefName, string sDefault); Int32 GetInt32Pref(string sPrefName, Int32 iDefault); // Methods to enable notifications 210 | Extending Fiddler with .NET Code public void SetPreferences.Bag.PrefMarker AddWatch(string sPrefixFilter, EventHandler preferHandler); void RemoveWatcher(PreferenceBag.PrefMarker wfToRemove); } Storing and Removing Preferences You can store preferences using the SetStringPref, SetBoolPref, and SetInt32Pref methods. If the preference with the same name and value to store directly: FiddlerApplication.Prefs["example.str"] = "value"; Internally, all preference values are stored as strings; the bool and int methods simply cast and parse the string's value. Call RemovePref or store a value of null to remove a preference's value. These three lines are equivalent: FiddlerApplication.Prefs.RemovePref("NameToRemove"); FiddlerApplication.Prefs.SetStringPref("NameToRemove", null); FiddlerApplication.Prefs["NameToRemove"] = null; Retrieving Preferences You can retrieve preferences using the GetStringPref, GetBoolPref, and GetInt32Pref methods. Each accepts the name of the preference and a default value to return if the requested preference does not exist. string sStr = FiddlerApplication.Prefs.GetStringPref("example.str", "demo"); bool bool1 = FiddlerApplication.Prefs.GetBoolPref("example.bool", false); int iNum = FiddlerApplication.Prefs.GetInt32Pref("example.int", 0); Alternatively, you can use the default indexer to retrieve a string preference directly: string sStr = FiddlerApplication.Prefs["example.str"]; If the requested preference does not exist, the indexer will return null. Watching for Preference Changes In many cases, it's desirable for extensions or FiddlerScript to be notified when a preference of a specified name is created, updated, or removed. To avoid the overhead of constantly polling preference values, you can instead "watch a branch" of preferences whose name contains a specified prefix. For instance, watching the fiddler.ui.toolbar prefix will raise an event whenever any preference whose name starts with that prefix is created, removed, or has its 211 | Extending Fiddler with .NET Code value changed. By thoughtfully naming your preferences, you can help ensure that your code is only notified of relevant changes. Notifications in Extensions Extensions may subscribe to receive notifications when preferences under a given prefix change by attaching a PrefWatcher. oWatcher = FiddlerApplication.Prefs.AddWatcher(string sPrefixToMatch, EventHandler fnToNotify); The event handler you supply in fnToNotify will be invoked for every preference change under the specified prefix. The ValueString property contains the name of the preference which has changed. The ValueString property contains the preference's new value, or null. You can permit garbage-collection of objects, be sure to remove your PrefWatcher when it is no longer needed: FiddlerApplication.Prefs.RemoveWatcher(oWatcher); Notifications in FiddlerScript FiddlerScript cannot easily use the AddWatcher method. // This callback function will be notified WatchPreference convenience method for this. Instead, use the new value as a boolean. You can permit garbage-collection of objects, be sure to remove your PrefWatcher when it is no longer needed: FiddlerApplication.Prefs.RemoveWatcher(oWatcher); // The event handler is notified when the watched preference changes. public static void ObservePrefChange(object sender, PrefChangeEventArgs oPCEA) { if (oPCEA.ValueBool) MessageBox.Show(oPCEA.PrefName + " changed to " + oPCEA.ValueString); } // Attach a callback to the preference change FiddlerObject.WatchPreference("example.str", FnChange); } FiddlerObject.WatchPreference(oPCEA.PrefName + " MessageBox.Show("Unexpected."); } } static function Main() { // Attach a callback to the preference change FiddlerObject.WatchPreference("example.str", FnChange); } FiddlerObject automatically disposes of the callback function when the FiddlerScript engine unloads on shutdown or because the script has been modified and must be recompiled. 212 | Extending Fiddler with .NET Code BUILDING EXTENSION INSTALLERS You must install your extensions using any technology you like. Fiddler simply requires that its Assembly .dll appear in the correct folder to load it next time that Fiddler launches. Behind all of the extensions I've written are installed using setup programs built using the NullSoft Scriptable Install System (NSIS). You can get this great freeware from . NSIS allows you to write a script that is compiled into a compressed executable file containing all of the binaries that make up your project. The resulting setup program is small and works properly across all versions of Windows. The only significant shortcoming I've encountered with NSIS is that it does not support Unicode, so you may need to use a different technology like WIX if you want your installer to use non-Latin characters (e.g. Japanese). A full explanation of how to use NSIS is beyond the scope of this book, but a simple example script you can use to get started...: In a NSIS Script, the semi-colon is a comment operator Name "MyExtension" ; TODO: Set a specific name for your installer's executable OutFile "InstallMyExtension.exe" ; Point to an icon to use for the installer, or omit to use the default Icon "C:\src\MyExt\MyExt.ico" XPStyle on ; Enable visual-styling for a prettier UI ; Explicitly demand permissions because we're going to write to ; Program Files. This prevents the "Program Compatibility Assistant" dialog. ; Note, you can use "user" here if you'd like, but then you must only write ; to HKCU and per-user writable locations on disk. RequestExecutionLevel "admin" ; Maximize compression SetCompressor /solid lzma BrandingText "v1.0.1.0" ; Text shown at the bottom of the Setup window ; TODO: Set the install directory to the proper folder ; ; To install to the Extensions folder, use: InstallDir "$PROGRAMFILES\Fiddler2\Inspectors" ; ; To install to the Inspectors folder, use: InstallDir "$PROGRAMFILES\Fiddler2\Inspectors" 213 | Extending Fiddler with .NET Code ; To install to the Inspectors folder, use: InstallDir "$PROGRAMFILES\Fiddler2\Inspectors" ; Registry keys used to track the install ; When run the Inspector's file under this key... then check that folder exists for files still removable ; if the registry key is found, then show the previous version number in the output folder into the installer ; When the installer runs, the file will be removed automatically. Set both SetOutPath "$INSTDIR\MyExt.dll" ; File "c:\src\MyExt\bin\release\MyExt.dll" ; ; TODO: List any other files your extension depends upon here. ; Be sure to also add those files to the list removed by the ; uninstaller at the bottom of this script ; ; ; Write information about the extension to the Add/Remove Programs dialog ; WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\MyExt" "DisplayName" "My Fiddler Extension" WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\MyExt" "Comments" "My Extension to Fiddler" WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\MyExt" "Publisher" "MyCo" WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\MyExt" "InstallLocation" "$INSTDIR" WriteRegDWORD HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\MyExt" "NoModify" 1 WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\MyExt" "NoRepair" 1 ; TODO: Update this line with the name of your uninstaller, set below ; WriteRegStr HKLM "Software\Microsoft\Windows\CurrentVersion\Uninstall\MyExt" "UninstallString" '"$INSTDIR\UninstallMyExt.exe"' SectionEnd ; end of default section ; ---------------------------- 214 | Extending Fiddler with .NET Code ; Perhaps surprisingly, this string cannot appear ; within the Uninstall section itself UninstallText "This will uninstall My Fiddler Extension." Section Uninstall Delete "$INSTDIR\MyExt.dll" ; ; TODO: Delete the other files you installed here. ; Remove the uninstaller regkey DeleteRegKey HKLM "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\MyExt" ; Remove the uninstaller ; exe SectionEnd ; end of Uninstall section ; Remove the Add/Remove Programs applet to allow time for user to uninstall the extension later. Save the script to a file named BuildInstaller.nsi. Next, create a batch file named BuildInstaller.bat to run the NSIS compiler and run the resulting setup program. If you do not have a code-signing certificate, you cannot run this batch file and MakeNSIS.exe @title Compile MyExtension Builder @cd c:\src\MyExt\ @c:\src\NSIS\makeNSIS.exe /V2 %BuildInstaller.nsi% @if %ERRORLEVEL%==1 goto done @Continue @Command Prompt Run BuildInstaller.bat and the NSIS compiler will compile your script into the InstallMyExtension.exe setup program. It will then sign that new program to prevent tampering and avoid security warnings in browsers. Warning: Windows keeps track of the origins of the DLL file using a data stream known as the "Mark of the Web." If you don't build and install instead ask the user to copy your extension's .DLL from a ZIP file or network share, you may find that it fails to load in Fiddler. Before your assembly is loaded, .NET v4 sees the "This file is from an untrusted Location" marker and will refuse to load the file by default. 215 | Extending Fiddler with .NET Code BUILDING INSPECTORS Developing custom Inspectors for Fiddler is simple-- you need only implement one class and one interface. Once the powerful Inspectors while maintaining backward compatibility, so be aware that some of the APIs you encounter will seem a bit redundant. There are two types of Inspectors: Request Inspectors and Response Inspectors; the former are used to view and/or modify a Session's request, while the latter are used to view and/or modify a Session's response. Both types of Inspector are derived from the abstract Inspector2 base class. Request Inspectors must also implement the IRequestInspector2 interface, while Response Inspectors must implement the IResponseInspector2 interface. The Inspector2 base class contains just two methods you must implement: public abstract void AddToTab(TabPage oPage); public abstract int GetOrder(); The AddToTab method is called when Fiddler has created a tabpage for your Inspector. In your implementation of this method, create your UI and add it to the TabPage. The UI could be as simple as a Text = "Raw"; // Create my UserControl and add it to my tab myControl = new RawView(this); oPage.Controls.Add(myControl); oPage.Dock = DockStyle.Fill; } The GetOrder method is called when Fiddler sorts the tabs for display within the UI. Inspectors with higher numbers will display to the left of the tab set while Inspectors with larger numbers will display toward the right of the tab set. Fiddler itself returns integers between -1000 and 110 for its own Inspectors. You may return any value, ordering is nondeterministic and can change between sessions. In addition to the abstract methods your Inspector must implement, the base class exposes a number of virtual methods that you may implement if you'd like. These include: public virtual void SetFontSize(float fSizeInPoints) public virtual int ScoreForSession(Session oS) public virtual int ScoreForContentType(string sMIMEType) 216 | Extending Fiddler with .NET Code public public virtual bool AssignSession(Session oS) bool CommitAnyChanges(Session oS) bool SessDirtyFlag() public virtual void SetInspectorFlags(InspectorFlags GetFlags) The ScoreForSession method runs when the user right-clicks on the Inspectors tab and selects the menu item, a message box will show the Inspector's right-click menu. If you don't override this method, when the user right-clicks on the Inspectors tab in the Session's UI font size. This method was created before the Fiddler Preference System was created, and thus this isn't used if you prefer. The ScoreForSession method runs when the user double-clicks a Session to display it. When this action occurs, each Inspector is polled to determine how eager it is to display a given request or response; the higher the value, the stronger the Inspector's desire to handle the Session. This method evaluates an aspect of a Session in deciding its score. AssignSession will be invoked on the most eager ScoreForContentType Inspectors return scores (meaning "I can't handle that at all") and should return 100 if it is confident that it is the best handler for a given request. Fiddler is configured so that the User-only aspect of a Session is a powerful one, as it enables your aspect of session to a powerful one. It enables your Inspector to render previews of WOFF font files, but most servers are not configured to return the correct Content-Type for such files. Therefore, the WebView Inspector checks the first 4 bytes of any response body to detect the WOFF signature bytes. static byte[] arr WOFF_MAGIC = new byte[4] { (byte)'w', (byte)'O', (byte)'F', (byte)'F' }; public virtual int ScoreForContentType(string sMIMEType) { // If not found, consult at the response's Content-Type header ScoreForContentType(oResponse.MIMEType); } // Avoid performing any expensive computations in your ScoreForSession method, as it runs often and users expect that inspecting a session will occur instantly. For example, your implementation of this method should not attempt to search a response body of arbitrary size for specific text, and should not perform network or disk operations. 217 | Extending Fiddler with .NET Code If you don't override the ScoreForSession method, the base implementation calls the ScoreForContentType method, passing the Content-Type header. The ScoreForContentType method predates the introduction of the ScoreForSession method, but it performs the same general function--it allows the Inspector to score how applicable it is for a given session. Again, scores between 0 and 100 should be returned. Avoid returning an overly-confident value for a general type; for instance, the JSON Inspector returns only 55 when the type is application/json, while other types are usually understood--for instance, if you have issued a custom MIME-type and no other handler is expected to be useful for your type. The WebView Inspector's implementation of ScoreForContentType is as follows: public virtual int ScoreForContentType(string sMIMEType) { if (sMIMEType.StartsWith("audio/", StringComparison.OrdinalIgnoreCase)) { return 60; } // Just in case a site sent a malformed MP3 type, check the whole string if (sMIMEType.IndexOf("mp3", StringComparison.OrdinalIgnoreCase) > -1) return 60; return 0; } If you do not implement the ScoreForContentType method, a default score of 0 is assumed and a different Inspector will be used for a general type. Handling the Session The IRequestInspector2 and IResponseInspector2 interfaces both inherit the IBaseInspector2 interface. public interface IBaseInspector2 { void Clear(); The inherited IRequestInspector2 interface adds one property: 218 | Extending Fiddler with .NET Code public interface IRequestInspector2 { HTTPRequestHeaders headers { get; set; } } Similarly, the inherited IResponseInspector2 interface adds one property: public interface IResponseInspector2 { HTTPResponseHeaders headers { get; set; } } bool bDirty { get; } // bReadOnly { get; set; } The headers property used to set or modify the response headers, the body property to set or modify the request or response body. The bReadOnly property is set to false, your Inspector should not permit the user to modify the headers or body. You can communicate the read-only status to the user by setting controls to readonly mode, and setting their background color to the color value stored in the Fiddler.CONFIG.colorDisabledEdit field. When bReadOnly is set to true, your Inspector should enable read/write mode for controls and allow the user to change the headers or body. If you set the bDirty property to false, your Inspector should not permit the user to edit the body or headers. You can communicate the read-only status to the user by setting controls to readonly mode, and setting their background color to the color value stored in the Fiddler.CONFIG.colorDisabledEdit field. When bReadOnly is set to true, your Inspector should enable read/write mode for controls and allow the user to change the headers or body. If you set the bDirty property to false, be sure that you've made no modifications within the Inspector before it's called. The headers property allows Fiddler to provide the HTTPRequestHeaders or HTTPResponseHeaders to your Inspector. The base implementation sets the property. When bReadOnly is set to true, your Inspector should enable read/write mode for controls and allow the user to change the headers or body. You must set the bDirty property to true whenever changes to the headers or body are made by the user. 219 | Extending Fiddler with .NET Code If you don't want to enable the user to modify the request or response in your Inspector, simply return null in the bDirty property. Warning: Fiddler does not currently clone the Session's headers when assigning to the headers property. Your Inspector must be coded to either clone the byte array when assigned, or must avoid making any changes to the bytes in the body array passed to your Inspector. When Fiddler attempts to commit any changes to your Inspector's bDirty property returns true, Fiddler will query the headers property for the new array to use. If the body array returned is different than the byte array that you got in your setter, close the body array using AssignSession method is called and must examine the Session's request or response to determine whether it is able to and wishes to display it. To store an empty body, return new byte[0]. Warning: Fiddler does not currently clone the Session's body when assigning to the body property. Your Inspector must be coded to either clone the byte array when assigned, or must avoid making any changes to the bytes in the body array passed to your Inspector. The Clear method is called if your Inspector should clear the current Session and your Inspector is invoked. In response to this method call, clear all data from the Inspector's UI, and indicate to the user that no Session is selected. 219 | Extending Fiddler with .NET Code When Fiddler is finished with an Inspector, the bReadOnly property is set. If you provide a Request Inspector, you will implement the IRequestInspector2 interface; if you provide a simple, easily understood API contract. But this simplicity prevents a number of shortcomings. For instance, it was impossible to provide an Inspector to get or set flags on the Session object, and even impossible to provide a Response Inspector that could view the Session's transaction timers. To resolve these shortcomings, the Inspector2 base class was extended with a new virtual method: public virtual void AssignSession(Session oS) { this.session = oS; } By default, this virtual method simply stores the reference to the Session in a field; the base class then calls the existing AssignSession and SetInspectorFlags methods. If your Inspector does not override these virtual methods, the legacy properties because not all codepaths in Fiddler call the newer virtual methods. Specifically, when editing a response using the AutoResponder tab, or using SetInspectorFlags. If your Inspector is designed to show or modify these virtual methods, and you need to implement any of the newer virtual method's behavior. If your Inspector does override the new AssignSession method, it must still implement all of the legacy properties because not all codepaths in Fiddler call the newer virtual methods. Specifically, when editing a response using the AutoResponder tab, or inserting a response using the uitlPrepareResponse* methods, the Inspector's legacy properties are used instead of the newer virtual methods. If your Inspector is designed to show or modify a Session's request or response, consider overriding the new AssignSession method. Specifically, when editing a response using the AutoResponder tab, the legacy methods are called. When this happens, the default implementation of AssignSession takes care of calling the legacy methods. 220 | Extending Fiddler with .NET Code If you do not override the AssignSession method, the base Inspector2 implementation will automatically assign to the headers, body, and bReadOnly properties based upon whether the Inspector is a Request or Response Inspector. virtual public HTTPRequestHeaders headers { (this is IRequestInspector2) ? IRequestInspector2(this).headers : IResponseInspector2(this).headers; } virtual public byte[] body { (this is IRequestInspector2) ? IRequestInspector2(this).body : IResponseInspector2(this).body; } virtual public bool bReadOnly { (this.state > SessionStates.HandTamperRequest && <oS.oFlags.ContainsKey("x-Unlocked"); return; } if (this is IResponseInspector2) { ... } any uncommitted edits, they should be stored to the Session with this method is called and your Inspector must change to the empty state. In no changes were committed, return false. If you do not override this method, after Fiddler consults the bDirty property of the Inspector, and if it is true, and then updates the Session's request or response using the headers and body using the CommitAnyChanges method. If your Inspector overrides this method and can successfully clear its dirty state.

do so and return true. This method exists because some Inspectors may not be able to easily reset their dirty state, and thus the bDirty property is readonly-- only the Inspector may change it. The benefit of implementing this method is that it prevents unneeded "Do you want to save changes?" prompts when the user edits an AutoResponder rule using a popup Inspectors window. The GetFlags method retrieves flags that describe the Inspector's behavior. Currently, this method is only overridden by the Transformer Inspector. You should not override this method for your own Inspectors. 221 | Extending Fiddler with .NET Code DEALING WITH HTTP COMPRESSION AND CHUNKING Most Inspectors cannot properly parse a chunked or compressed response body into a higher-level format like XML or JSON text. For most Inspectors, the user is simply encouraged to decode the response manually before inspecting: if (oHeaders.Exists("Transfer-Encoding") || oHeaders.Exists("Content-Encoding")) lblDisplayMyEncodingWarning.Visible = true; return; } Fiddler allows the user to easily remove the compression and chunking by clicking the yellow notification bar or using the Transformer Inspector. However, it is possible to make a copy of the body and decode it for your Inspector's private use. Be wary of the cost of decoding large bodies; you shut yourself off having the user do it manually just once instead of every time the response is loaded in your Inspector. There are three ways to get the uncompressed and unchunked body. Decoding a Copy of the Headers and Body. Using this information, the body may be cloned and decoded using the Utilities.utilDecodeHTTPBody static method. If the body needs to be interpreted as a string, the getEntityBodyEncoding and GetStringFromArrayRemovingBOM methods can be used to convert the body bytes to a string. Here's what the JSONView Inspector does in its body property setter, and the incoming value object refers to the new body byte array that is being assigned to the Inspector: if (null != oHeaders) { // Check for no body if ((null == value) || (value.Length == 0)) return; if (!oHeaders.ExistsAndContains("Content-Type", "application/json") && !oHeaders.ExistsAndContains("Content-Type", "javascript")) { return; } if (oHeaders.Exists("Transfer-Encoding") || (oHeaders.Exists("Content-Encoding")) { // Body is oversized to avoid corrupting the original byte[] arrCopy = (byte[])value.Clone(); 222 | Extending Fiddler with .NET Code try { // Decode. Warning: Will throw if value cannot be decoded Utilities.utilDecodeHTTPBody(oHeaders, ref arrCopy); value = arrCopy; } catch { // Leave value alone. } } } } // Okay, now the body stored in "value" is unchunked and uncompressed. We need to convert it's to a .NET string (Unicode) and returned as the result of the function. .NET strings are immutable. You don't need to worry about corrupting the original HTTP body by manipulating the returned string, but update the Session's headers to remove the reference to the now-removed encodings. 223 | Extending Fiddler with .NET Code Your Inspector should not automatically call these methods when a Session is assigned to your Inspector. If your Inspector would permanently change the response so it's meant to be in a readonly state. Instead, call these methods only if the user takes an explicit action, like clicking on a button or link that indicates that a permanent change will be made. Inspector Assemblies When looking for Inspectors, Fiddler loads all assembly .DLLs from the %PROGRAMFILES% Fiddler2 Inspectors and %USERPROFILE% My Documents Fiddler2 Inspectors folders and enumerates all public classes within them to find Request and Response Inspectors. Install to the %PROGRAMFILES% location to make your Inspectors available to all users on the machine, or the %USERPROFILE% folder to make them available only to the current user. In addition to placing your Inspector assemblies in the appropriate folder, you must also mark your assembly to indicate the minimum version of Fiddler required by your Inspectors. Set the RequiredVersion attribute as follows: // Inspector requires methods introduced in v2.3.9.0... [assembly: Fiddler.RequiredVersion("2.3.9.0")] If Fiddler loads an assembly for which the RequiredVersion attribute indicates a later version of Fiddler is needed, the user will be notified that an update is required and no Inspectors will be loaded from the assembly. Assemblies in the Inspectors folder which lack a RequiredVersion attribute are silently ignored. Fiddler skips enumeration of DLLs named with a leading underscore character (_) when loading Inspectors. Many Inspectors may rely upon 3rd party assemblies that do not themselves contain any Fiddler-related code. To avoid the performance cost of Fiddler enumerating such assemblies to look for Inspectors, name those assemblies using a leading underscore. 224 | Extending Fiddler with .NET Code BUILDING EXTENSIONS When you need to add functionality to Fiddler and an Inspector isn't the appropriate way to expose that functionality, build an extension instead. Extensions can enhance Fiddler with anything from simple features (like a new menu item) to extremely complicated business logic and reporting infrastructures. Getting started is easy. The simplest Fiddler extension might implement only one interface whose methods are called to load and unload the extension: public interface IFiddlerExtension { void OnLoad(); void BeforeUnload(); } The OnLoad function is 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 inherits from the IFiddlerExtension interface, and permits your extension to receive callbacks as each Session is processed, enabling modifications, logging, or other operations. public interface IAutoTamper : IFiddlerExtension { void AutoTamperRequestBefore(Session oSession); void AutoTamperRequestAfter(Session oSession); void AutoTamperResponseBefore(Session oSession); void AutoTamperResponseAfter(Session oSession); void OnBeforeReturningError(Session oSession); } The difference between the *Before and *After methods is that the former execute before the user has a chance to manually tamper with a breakpointed Session using the Inspectors, while the latter methods execute after any such modifications are completed. If you permit the user to manually undo any modifications your method makes, use the *Before methods, otherwise use the *After methods. The IAutoTamper2 interface inherits from IAutoTamper and adds one additional method that is called when Fiddler has read the server's response headers, (usually) before the response body is available: public interface IAutoTamper2 : IAutoTamper { void OnPeekAtResponseHeaders(Session oSession); } Code A common use of the OnPeekAtResponseHeaders method is to examine the response's headers to decide whether the response should be buffered or streamed. For instance, to stream all image files, you could write the following code: void OnPeekAtResponseHeaders(Session oSession) { if (oSession.oResponse.MIMEType.StartsWith("image/")) { bBufferResponse = false; } } Be careful not to attempt to read the responseBodyBytes array in this method, because a Null Reference Exception is likely to ensue. Additionally, you should avoid manipulating headers related to the response's length (e.g. Content-Length, Transfer-Encoding, Connection) because the subsequent attempt to read the response body from the server will likely fail. The IAutoTamper3 interface inherits from IAutoTamper2 and adds one additional method that is called when Fiddler has read the client's request headers, (usually) before the request body is available: public interface IAutoTamper3 : IAutoTamper2 { void OnPeekAtRequestHeaders(Session oSession); } This method is only useful in certain niche scenarios; in most cases, the availability of the request's headers alone is not sufficient to do anything useful. Understanding Threading The IAutoTamper interfaces' methods 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 interfaces' methods can be called before the OnLoad event fires-- Fiddler allows traffic to flow before the UI is fully available. Therefore, most extensions will initialize objects and UI inside their constructor and wait until OnLoad is called before they implement the IHandleExecAction interface are called when the user has entered a command into the QuickExec box or invoked a command using the ExecAction.exe command-line tool. public interface IHandleExecAction { bool OnExecAction(string sCommand); } 226 | Extending Fiddler with .NET Code If your extension would like to handle the command (and prevent further processing by other extensions and Fiddler itself) return true from your implementation of this method. The Utilities class includes a helper function for interpreting the sCommand parameter: public static string[] Parameterize(string sCommand) The Parameterize method tokenizes a string into tokens. The string is split by whitespace characters, unless the spaces are contained within quotation marks, in which case the entire quoted string is stored as a token. Use a backslash character (\) to escape quotation marks so that they may appear literally within a string. Example Extension The following is a simple extension which examines all responses as they are received. It determines whether the responses attempt to set cookies, and if so, whether a valid P3P Compact Policy statement has been provided. This example adds an item to the top-level menu, adds a column to the Web Sessions list, and modifies Sessions as they are processed. using using using using Fiddler; System; System.Text; System.IO; System.Collections; System.Globalization; System.Collections.Generic; System.Windows.Forms; System.Diagnostics; System.Text.RegularExpressions; [assembly: Fiddler.RequiredVersion("2.3.9.0")] [assembly: AssemblyVersion("1.0.1.0")] [assembly: AssemblyDescription("Scans for Cookies and P3P")] public class TagCookies : IAutoTamper2 { private bool bEnabled = false; private bool bEnforceP3PValidity = false; private bool bCreatedColumn = false; private MenuItem mnEnabled; private MenuItem miEnforceP3PValidity; private MenuItem mnuEnforceP3PValidity; private MenuItem mnuCookieTag; private enum P3PState { NoCookies, NoP3PAndSetsCookies, P3POk, P3PUnsatisfactory, P3PMalformed } 227 | Extending Fiddler with .NET Code public void InitializeMenu() { mnuCookieTag = new MenuItem("&Cookies"); mnuCookieTag.MenuItems.Add(mnEnabled = new MenuItem("&Enabled")); mnEnabled.Click = new MenuItem; this.mnEnabled.Click); this.mnEnabled.Checked = bEnabled; this.mnEnabled.Index = 0; this.miEnforceP3PValidity.Index = 1; this.miEnforceP3PValidity mi = new System.EventHandler(this.miEnforceP3PValidity_Click); this.miEnforceP3PValidity.Enabled = bEnabled; if (bEnabled) { EnsureColumn(); } FiddlerApplication.Prefs.SetBoolPref("extensions.tagcookies.enabled", bEnabled); } public void miEnforceP3PValidity_Click(object sender, EventArgs e) { miEnforceP3PValidity.Checked; } 228 | Extending Fiddler with .NET Code bEnforceP3PValidity = miEnforceP3PValidity.Checked; FiddlerApplication.Prefs.SetBoolPref("extensions.tagcookies.EnforceP3PValidity", bEnforceP3PValidity); } private void EnsureColumn() { // If we already created the column, bail out. if (bCreatedColumn) return; // Add a new Column to the Web Sessions list, titled "Privacy Info", // that will automatically fill with each Session's X-Privacy flag string FiddlerApplication.UI.lvSessions.AddBoundColumn("Privacy Info", 1, 120, "X-Privacy"); bCreatedColumn = true; } public TagCookies() { this.bEnabled = FiddlerApplication.Prefs.GetBoolPref("extensions.tagcookies.enabled", false); this.bEnforceP3PValidity = FiddlerApplication.Prefs.GetBoolPref("extensions.tagcookies.EnforceP3PValidity", true); InitializeMenu(); if (bEnabled) { EnsureColumn(); } else { this.bEnforceP3PValidity.Enabled = false; } } private void SetP3PStateFromHeader(string sValue, ref P3PState oP3PState) { // If there is a string sUnsatCat = string.Empty; string sUnsatChar = String.Empty; sValue = sValue.Replace("\", ""); string sCP = null; // Use a Regular Expression to search the header for a CP attribute Regex r = new Regex("CP\s*=\s*[\"']?(?<P3P>[^\"']*)"); Match m = r.Match(sValue); if (m.Success && (null != m.Groups["P3P"])) { sCP = m.Groups["P3P"].Value; } if (null == sCP) { // If this header didn't contain a CP attribute, it's unsatisfactory. oP3PState = P3PState.P3PMalformed; return; } // Okay, we've got a compact policy string. Evaluate each token. sCP.Splitlstar char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries); foreach (string sToken in sTokens) { // Reject clearly invalid tokens if ((sToken.Length < 3) || (sToken.Length > 4)) // oP3PState = P3PState.P3PMalformed; return; } // Track any tokens with "Unacceptable" privacy category if ("PHY,ONL,GOV,FIN,".IndexOf("," + sToken + ",", StringComparison.OrdinalIgnoreCase) > -1) { sUnsatCat += (sToken + ""); continue; } // Track any tokens with "Unacceptable" privacy purposes if (",COR,OTR,UNR,PUB,IVA,IVD,CON,TEL,OTP,".IndexOf("," + sToken + ",", StringComparison.OrdinalIgnoreCase) > -1) { sUnsatPurpose += (sToken + ""); continue; } // TODO: Check each token against the list of 70-some valid tokens and / reject if it's not found. } // // if { // A comma HTTP response, examine the response headers for attempts to set cookies, and check for a P3P header too. We do this in OnReadAtResponseHeaders rather than OnBeforeResponse because we only need the headers and do not need to read the bodies and thus do not need to use the bufferResponse in 230 | Extending Fiddler with .NET Code if (oSession.Response.headers.Exists("P3P")) { SetP3PStateFromHeader(oSession.oResponse.headers["P3P"], ref oP3PState); } // Based on the cookie/P3P state, set the background color of item in the // Web Sessions list. Also set the X-Privacy flag which is shown in the / column that we created. switch (oP3PState) { case P3PState.P3POk: oSession["ui-backcolor"] = "#ACDC85"; oSession["X-Privacy"] = "Sets cookies & P3P"; break; case P3PState.NoP3PAndSetsCookies: oSession["ui-backcolor"] = "#FAFDA4"; oSession["X-Privacy"] = "Sets cookies without P3P"; break; case P3PState.P3PUnsatisfactory: oSession["ui-backcolor"] = "#EC921A"; oSession["X-Privacy"] = "Sets cookies; P3P unsat. for 3rd-party use"; break; case P3PState.P3PMalformed: oSession["ui-backcolor"] = "#E90A05"; if (bEnforceP3PValidity) { oSession.oResponse.headers.Remove("P3P"); } break; } } // Our extension doesn't need any of the other AutoTamper* methods. public void AutoTamperRequestBefore(Session oSession) {/*noop*/} public void AutoTamperRequestAfter(Session oSession) {/*noop*/} AutoTamperResponseBefore(Session oSession) {/*noop*/} OnBeforeReturningError(Session oSession) {/*noop*/} } If your extension needs to extend to the Web Sessions list's context menu, add your new menu items to FiddlerApplication.UI.lmnuSessionContext. If your extension needs to create a new top-level tab, it can add one to the tabsViews control. Here's how the Timeline tab adds its tab with a title and icon: public void OnLoad() { oPage = new TabPage("Timeline"); oPage.ImageIndex = (int)FiddlerIconseum.Timeline; FiddlerApplication.UI.tabsViews.TabPages.Add(oPage); } If your tab contains any non-trivial UI, you should use the Delay Load pattern described in the Best Practices for Extensions section to ensure that the heavyweight initialization only occurs if and when the user switches to your new tab. Extension Assemblies When looking for extensions, Fiddler loads all assembly .DLLs from the %PROGRAMFILES% Fiddler2 Scripts and %USERPROFILE% My Documents Fiddler2 Scripts folders and enumerates all public classes within them to find extensions. Install to the %PROGRAMFILES% location to make your extension available to all users on the machine, or the %USERPROFILE% folder to install it for the current user only. In addition to placing your extension assemblies in the appropriate folder, you must also mark your assembly to indicate the minimum version of Fiddler required by your extension. Set the RequiredVersion attribute as follows: // Transcoders require methods introduced in v2.3.9.0... [assembly: Fiddler.RequiredVersion("2.3.9.0")] If Fiddler loads an assembly for which the RequiredVersion attribute indicates a later version of Fiddler is needed, the user will be notified that an update is required and no extensions will be loaded from the assembly. Assemblies in the Scripts folder which lack a RequiredVersion attribute are silently ignored. Fiddler skips enumeration of DLLs named with a leading underscore character (_) when loading extensions. Many extensions may rely upon 3rd party assemblies that do not themselves contain any Fiddler-related code. To avoid the performance cost of Fiddler enumerating such assemblies to look for extensions, name those assemblies using a leading underscore. 232 | Extending Fiddler with .NET Code BUILDING IMPORT AND EXPORT TRANSCODE RS One of the most common goals when building an extension is to export captured data out of Fiddler, or import data captured by other tools. While there's no reason you can't build such an extension using the general IFiddlerExtension interface, there are shortcomings in doing so: Such extensions have a performance cost, even when unused Such extensions require a lot of boilerplate code Inconsistent import/export UI leads to user confusion To combat these shortcomings, Fiddler offers a specific extension type called a Transcoder. Fiddler ships with a number of these Transcoders by default, and developers can build new Transcoders easily. A Transcoder is simply an object that implements either or both of the ISessionImporter and ISessionExporter interfaces. Transcoder Assemblies are only loaded when invoked by the user via the Import Sessions and Export Sessions entry points on the File menu. Delay-loading means that Transcoders do not have a performance impact on Fiddler boot time or memory usage. Direct Fiddler to load your Transcoder assemblies Fiddler builds Transcoder assembly DLLs from the %PROGRAMFILES% folder to install it for the current user only. Be sure to set the RequiredVersion attribute as follows: // Transcoders require methods introduced in v2.3.9.0... [assembly: Fiddler.RequiredVersion("2.3.9.0")] ...and ensure that your Transcoder classes are marked public. The ProfferFormat Attribute Each class must be decorated with one or more ProfferFormat attributes that specify FormatName and FormatDescription strings that describe the Transcoder. These strings are shown to the user in the Select Format window. For instance, the HTTPArchive format exporter is decorated with two ProfferFormat attributes: [ProfferFormat("HTTPArchive v1.1", "A lossy JSON-based HTTP traffic archive format. Standard is documented @)] [ProfferFormat("HTTPArchiveFormat;Export: ISessionExporter 233 | Extending Fiddler with .NET Code { //... Each of these two attributes identifies a format supported by the Transcoder, and each is displayed in the Select Format window's dropdown. When the user invokes the Transcoder, the FormatName is provided to the ImportSessions or ExportSessions method so that a Transcoder that supports multiple formats may know which format the user has requested. The ISessionImporter interface Transcoders that implement the ISessionImporter interface are invoked when the user invokes the File > Import Sessions... menu command. The interface defines one method: public interface ISessionImporter : IDisposable { Session[] ImportSessions(string sFormatName, Dictionary dictOptions, EventHandler evtProgressNotifications); } ... and it also inherits the Dispose method from IDisposable. The ImportSessions method should return an array of Session objects imported from the import of the data, or null if there was an error in loading the data. When the user chooses File > Import Sessions..., Fiddler will call the Transcoder and then add the Sessions returned by the Transcoder to the Web Sessions list. The ISessionExporter interface Transcoders that implement the ISessionExporter interface are called when the user invokes the File > Export Sessions menu command. The interface defines one method: public interface ISessionExporter : IDisposable { bool ExportSessions(string sFormatName, Session[] oSessions, Dictionary dictOptions, EventHandler evtProgressNotifications); } 234 | Extending Fiddler with .NET Code ... and it also inherits the Dispose method from IDisposable. The ExportSessions method should return true if the export of the Sessions passed in the oSessions parameter is successful. If the data could not be exported for some reason, the Transcoder should log an explanation to Fiddler's Log and return false. Handling Options The ImportSessions and ExportSessions methods' second parameter is a Dictionary that permits callers (FiddlerScript, other Extensions, or a FiddlerCore-based application) to pass options into the Transcoder. The provided dictOptions dictionary may be null, or may contain a set of string-keyed objects. Most Importers support specification of a filename like this: dictOptions["Filename"] = "C:\test.file"; When a filename is provided through this mechanism, the Transcoder can suppress its normal UI and respect the provided options. Similarly, many Transcoders support a Silent key, which, if present, ensures that no warning or error UI is shown to the user. Other Transcoders have other options, depending on their individual needs. For instance, FiddlerScript can invoke the HTTPArchive Transcoder, passing in the filename string and maximum response size integers thusly: var oSessions = FiddlerApplication.UI.GetAllSessions(); var oExportOptions = FiddlerObject.createDictionary(); oExportOptions.Add("Filename", "C:\users\ericlaw\desktoplout1.har"); oExportOptions.Add("MaxTextBodyLength", 1024); oExportOptions.Add("MaxBinaryBodyLength", 16384); FiddlerApplication.DoExport("HTTPArchive v1.1", oSessions, oExportOptions, null); As you can see in this example, the DoExport method on the FiddlerApplication object enables script or an extension to invoke a Transcoder by passing the FormatName. Similarly, the DoImport method permits invocation of a Transcoder to import Sessions. The HTTPArchive Transcoder's code checks whether its caller has set any options like so: public bool ExportSessions(string sFormat, Session[] oSessions, Dictionary dictOptions, EventHandler evtProgressNotifications) { //... subscribe to progress notifications from your Transcoder by passing in an EventHandler delegate via evtProgressNotifications. As the import or export proceeds, your Transcoder should construct ProgressCallbackEventArgs objects representing the current progress and pass them to the delegate: public string ProgressText { get; } public string ProgressText { get; set; } When constructing the EventArgs, set fICompletionRatio to the fraction of the operation that is complete (ranging from 0 to 1.0). If the completion ratio cannot be determined, simply set fICompletionRatio to 0 or a "guess" between 0 and 1. Set srProgressText to provide status text (for instance "Adding session #1234" or "Compressing output." 236 | Extending Fiddler with .NET Code Notes on Threading and Transcoders in FiddlerCore Currently, methods of the ISessionImporter and ISessionExporter interfaces are called on the application's UI thread. This is almost certain to change in the future, so you should ensure that your classes are thread-safe and that they do not attempt to directly manipulate any UI. Transcoders may be cross-coupled to work with FiddlerCore which has no UI and thus any attempt to manipulate the Fiddler UI will throw an exception. To cross-compile with FiddlerCore, you must either abort the current operation as soon as possible, returning null or false to the caller. 236 | Extending Fiddler with .NET Code Notes on Threading and Transcoders in FiddlerCore Currently, methods of the ISessionImporter and ISessionExporter interfaces are called on the application's UI thread. This is almost certain to change in the future, so you should ensure that your classes are thread-safe and that they do not attempt to directly manipulate any UI. Transcoders may be cross-coupled to work with FiddlerCore which has no UI and thus any attempt to manipulate the Fiddler UI will throw an exception. Beyond Files While most Transcoder objects are designed to import or export files of various formats, there's no requirement that files are the source or destination of a Transcoding. For instance, you can build a Transcoder that reads variants of your export, which database and adds it back to the Web Sessions list. Example Transcoder To build a simple Transcoder, follow these steps: 1. Start Visual Studio (version 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 %PROGRAMFILES% Fiddler2 folder. If you want to build a version of your Transcoder that works with FiddlerCore instead, reference FiddlerCore.dll. 5. Click Ok to add the reference. 6. In the Solution Explorer, right click the project's Properties. 7. On the Application tab, ensure that the project's Target Framework is Any CPU. 9. On the Build Events tab, add the following as the Post-build event command line: copy "$(TargetPath)" "%USERPROFILE% My Documents Fiddler2 Scripts" 237 | Extending Fiddler with .NET Code This will ensure that every time you successfully build your project, your new assembly is copied to the proper location for Fiddler to load. You will, of course, need to close Fiddler between each build or the command to copy the assembly to the target folder will fail with a "File in use" error. 10. On the Debug tab, click Start external program and enter %PROGRAMFILES% Fiddler2 Fiddler.exe as the target program. When you hit F5 to test your Transcoder, Fiddler will launch. 11. Save the project. Modify the default class1.cs (or create a new class) in your project to look like the following: using System; using System.Collections.Generic; using System.Windows.Forms; using Fiddler; As noted in the introductory text 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 dictOptions, EventHandler 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 238 | Extending Fiddler with .NET Code // for the export format they selected if (sFormat == "Comma-Separated Values") { chSplit = ","; if // As = sFormat, "TSV Files (*.tsv)|*.tsv"; } // We didn't get a filename, user cancelled. If so, bail out. if (string.IsNullOrEmpty(sFilename)) return false; try { StreamWriter swOutput = new StreamWriter(sFilename, false, Encoding.UTF8); int iCount = 0; int iMax = oSessions.Length; foreach (Session oS in oSessions) { iCount++; // The ViewItem object is the ListViewItem in the WebSessions list 239 | Extending Fiddler with .NET Code // Obviously, this doesn't exist in FiddlerCore-based applications if (null != oS.ViewItem) { bFirstCol = false; } else { bFirstCol = false; } // Remove any delimiter characters from the value swOutput.Write(oLVC.Text.Replace(chSplit, "")); } else { bFirstCol = false; } // Remove any delimiter characters from the value swOutput.Write(oLVC.Text.Replace(chSplit, "")); } swOutput.WriteLine(); #endregion WriteColHeaders #region WriteEachSession foreach (Session oS in oSessions) { iCount++; // The ViewItem object is the ListViewItem in the WebSessions list if (null != oS.ViewItem) { bFirstCol = true; } foreach (ListViewItem.ListViewSubItem oLVC in oS.ViewItem.SubItems) { if (!bFirstCol) { swOutput.Write(chSplit); } else { bFirstCol = false; } // Remove any delimiter characters from the value ProgressCallbackEventArgs PCEA = new ProgressCallbackEventArgs((iCount / (float)iMax), "wrote " + iCount.ToString() + " records."); evtProgressNotifications(null, PCEA); // If the caller tells us to cancel, abort quickly if (PCEA.Cancel) { swOutput.Close(); return false; } } } #endregion WriteEachSession swOutput.Close(); bResult = true; } catch (Exception eX) { // TODO: Replace alert with FiddlerApplication.Log.LogFormat(... MessageBox.Show(eX.Message, "Failed to export"); bResult = false; } return bResult; } public void Dispose() { /*no-op*/ } } 240 | Extending Fiddler with .NET Code This simple example defines an ISessionExporter that offers two text formats: "comma-separated values" and "tab-separated values." When ExportSessions is called, the method examines the dictOptions dictionary and prompts the user for any missing information (like the target filename) required to complete the export. It then processes each Session in the oSessions array, reporting progress of the export back to Fiddler using the callback event handler provided. 241 | Extending Fiddler with .NET Code FiddlerCore OVERVIEW As you've seen in prior chapters, you can extend Fiddler's work with script and .NET code, and this is the best approach for building new functionality for most users. However, in some scenarios, like test automation, it would be more natural to add proxy functionality into an existing tool or test harness instead of using the entirety of Fiddler for the job. Enter FiddlerCore. FiddlerCore is a class library that you can reference in your .NET applications to add proxy functionality to .NET programs with none of the Fiddler user-interface. This diagram shows the difference between extending Fiddler with your code and extending your code with FiddlerCore: If you've previously built a Fiddler extension, you'll find that programming against FiddlerCore is an easy adjustment. Many FiddlerCore-based applications are first prototyped as a Fiddler extension being moved into a standalone program. Building your code on FiddlerCore first allows you to easily see what is happening to web traffic using Fiddler's Inspectors. Once you're using FiddlerCore, you can only see the web traffic by adding logging or Inspectors to your application (unless you chain your FiddlerCore application to an upstream or downstream Fiddler instance!). Legalities Before adopting any library, you should always understand its license terms to ensure that your use of the library is permitted. Fortunately, FiddlerCore has a simple license, and you're free to use it for any legal purpose. FiddlerCore's certificate generation code used to intercept HTTPS traffic depends upon Microsoft's makecert.exe. If you do not need to decrypt HTTPS traffic, your program does not need to take a dependency on makecert. The utility is a licensed Visual Studio 2008 redistributable1, so if you have a license to that tool you may redistribute makecert with your FiddlerCore-based applications. If you do not have the appropriate license, you should either speak to your lawyer or supply a different certificate generation tool. Getting Started with FiddlerCore You can download FiddlerCore from . The download package contains two copies of the library, one compiled for the v4 .NET Common Language Runtime, and one compiled for the v2 .NET CLR. The package also contains source code for a demonstration program which exercises FiddlerCore's functionality, and the makecert.exe utility. You should also take the opportunity to download the separate documentation package, which will install the FiddlerCore.chm file. This file is an HTML Help file2 that shows all of the classes, properties, methods, and fields available in FiddlerCore for your application to call. After installing these packages, you'll have a new folder named FiddlerCoreAPI on your desktop. Compiling the Sample Application Explore the SampleApp subfolder of the FiddlerCoreAPI folder, and open the Demo.csproj file to launch Visual Studio. The Demo project contains a Console Application which demonstrates FiddlerCore's basic functionality. Scroll to the static void Main function in Program.cs to see how FiddlerCore is configured and used. The very first thing you'll need to do is attach the event listeners that handle events raised by FiddlerCore. A key thing you must keep in mind is that FiddlerCore typically calls event handlers on the background threads in which Sessions are processed, so if you make any calls that modify shared objects (e.g. a list of Sessions) or update the application's UI, you must do so using thread-safe mechanisms. To update shared data-structures, use a Mutex or lock; to update WinForms-based UI, use the BeginInvoke method on the control being updated. Supplying event handlers for FiddlerCore's OnNotification and LogString events allows you to see error messages and other information which is very useful for troubleshooting problems in your application: FiddlerApplication.OnNotification += delegate(object s, NotificationEventArgs oNEA) { Console.WriteLine("** NotifyUser: " + oNEA.NotifyString); }; 248 | FiddlerCore FiddlerApplication.Log.OnLogString += delegate(object s, LogEventArgs oLEA) { Console.WriteLine("** LogString: " + oLEA.LogString); }; The next step is to attach event handlers that fire as each Session is processed. It's common to attach event handlers for the BeforeRequest and BeforeResponse events. One of the most common behaviors of the BeforeRequest event handler is to set the bBufferResponse property to false if you want responses to stream to the client, or to true if you want the Response to be fully buffered before the client gets it. If your application is thread-safe, you must take care to ensure that your event handlers are thread-safe. Use BeginInvoke or Invoke to marshal the work to your list in the BeforeRequest handler. FiddlerApplication.BeforeRequest += delegate(Session oS) { // Buffer response to allow response tampering oS.bBufferResponse = true; // Use a thread-safe mechanism to update my List Monitor.Enter(oAllSessions); oAllSessions.Add(oS); Monitor.Exit(oAllSessions); }; A BeforeResponse handler enables you to modify a Session's response. FiddlerApplication.BeforeResponse += delegate(Session oS) { // Uncomment the following two statements to decompress/unchunk the // HTTP response and thus make it easier for us to search for any strings // oS.utilDecodeResponse(); if you want to modify the bytes before they come to the // client oS.utilDecodeResponse(); }; 247 | FiddlerCore To begin the decryption process, you must set properties on the FiddlerCoreStartupFlags object. Default; // ...but if, say, we won't want FiddlerCore to Decrypt // HTTPS traffic, we can unset that flag at this point oFCSF = oFCSF & ~FiddlerCoreStartupFlags.DecryptSSL); Then call the Startup method, passing in the desired port and your flags: // Start listening on port 8877 247 | FiddlerCore FiddlerApplication.Startup(8877, oFCSF); If you don't watch FiddlerCore listens on, freeing you from handling errors when that port is already in use, pass 0 for the port and FiddlerCore will automatically select an available port. The port chosen can be retrieved using the property FiddlerApplication.oProxy.ListenPort. When using the default flags, the Startup method will immediately register the new proxy as the system's default proxy. Any BeforeRequest and BeforeResponse event handlers will begin to fire as traffic is captured. After you're done with FiddlerCore, you must call the Shutdown method to unregister as the system proxy and close the listening socket. FiddlerApplication.Shutdown(); Note that the Shutdown method cannot gracefully terminate any in-progress Web Sessions. Those sessions' background threads will continue to run (potentially calling into your event handlers) but will likely eventually be aborted by either an ObjectDisposedException or NullReferenceException when they attempt to access the now-null FiddlerApplication.oProxy object. If you'd prefer, you may instead call: FiddlerApplication.oProxy.Detach(); ...to unregister as the system proxy. What a few seconds to permit in-progress Sessions to complete before calling the Shutdown method. FiddlerCoreStartupFlags FiddlerCoreStartupFlags enumeration allows you to specify the behavior of the FiddlerCore proxy endpoint. The enumeration exposes the following flags: None No options are set. RegisterAsSystemProxy When Startup is called, FiddlerCore will attempt to register itself as the default proxy for all HTTP traffic that flows through the proxy. AllowRemoteClients Permit connections from remote PCs or devices. You may need to enable a firewall exception for your application to receive remote traffic. ChainToUpstreamGateway Automatically adopt the system's existing proxy settings as an upstream proxy server. MonitorAllConnections Register as the default proxy for all WinINET connections, including dialup and VPN connections. When not set, only the LAN connection is set to point at the FiddlerCore proxy instance. HookUsingPAC File this option is not implemented. CaptureLocalhostTraffic Add to the system's proxy exceptions list, enabling IE9+ to send loopback traffic to FiddlerCore. The FiddlerCore Default This flag encompasses the following flags: RegisterAsSystemProxy | DecryptSSL | AllowRemoteClients | ChainToUpstreamGateway | MonitorAllConnections | CaptureLocalhostTraffic Using FiddlerCoreStartupFlags.Default is the recommended approach as it enables the features desired by most applications. Even if you don't want all of the flags in the default set, your best bet is to start with Default and use bit-manipulation to mask off unwanted options: FiddlerCoreStartupFlags oFlags = FiddlerCoreStartupFlags.Default & ~FiddlerCoreStartupFlags.DecryptSSL); This approach has the benefit that as new options are introduced in future versions of FiddlerCore, your application will automatically be opted-in (upon recompile) to new default options. 249 | FiddlerCore THE FIDDLERAPPLICATION CLASS The static FiddlerApplication class provides many useful event handlers, static properties, and methods that control FiddlerCore's behavior. Most of these APIs are also available for use by extensions running inside Fiddler itself. FiddlerApplication Events FiddlerApplication fires events raised by the FiddlerCore engine and much handlers are thread-safe. Use BeginInvoke or Invoke to update user-interface elements on the UI thread. RequestHeadersAvailable Event This event fires after FiddlerCore has read the complete set of request headers from the client. It is rarely useful because it fires so early in request processing. BeforeRequest Event This event fires after the request has been completely read from the client. OnValidateServerCertificate Event This event fires when FiddlerCore evaluates a HTTPS server's certificate. Your event handler may accept or reject the server's certificate based on the criteria of your choosing. FiddlerApplication.OnValidateServerCertificate += new System.EventHandler(CheckCert); void { // // if { // CheckCert(object sender, ValidateServerCertificateEventArgs e) If there's an obvious issue with the presented certificate, it will be rejected unless overridden here. (SslPolicyErrors.None != e.CertificatePolicyErrors) // Perhaps you have // an opinion about this certificate chain, load information on this certificate chain; FiddlerApplication.Log.LogFormat("Notaries have indicated that the "+ "certificate presented for {0} is {1}", e.ExpectedCN, bNotariesAffirm ? "VALID" : "INVALID"); if (bNotariesAffirm) { 250 | FiddlerCore e.ValidityState = CertificateValidity.ForceValid; return; } e.ValidityState = CertificateValidity.ForceInvalid; return; } OnResponseBuffer Event This event fires on every read of the server's response stream. It is generally not handled because the event it's raised on background threads. FiddlerApplication.OnReadResponseBuffer += new EventHandler(OnRead); static void ReadEventArgs e) { Console.WriteLine(String.Format("Read {0} response bytes for session {1}", e.iCountOfBytes, e.sessionOwner.id)); // NOTE: arrDataBuffer is a fixed-size array. Only bytes 0 to iCountOfBytes should be read/manipulated. // Just for kicks, lowercase every ASCII char. Note that this will // obviously mangle any binary MIME files and break many types of markup for (int i = 0; i < e.iCountOfBytes; i++) { if (e.arrDataBuffer[i] > 0x40) && (e.arrDataBuffer[i] < 0x5b)) { e.arrDataBuffer[i] = (byte)(e.arrDataBuffer[i] + (byte)0x20); } } 251 | FiddlerCore ResponseHeadersAvailable Event This event fires when FiddlerCore has read the complete set of response headers from the server. It provides the opportunity to set the Session's // oS.bBufferResponse = true; } 251 | FiddlerCore ResponseHeadersAvailable Event This event fires when FiddlerCore has read the complete set of response headers from the server. It provides the opportunity to set the Session's bBufferResponse property based upon the headers of the response. BeforeResponse Event This event fires after the response has been completely read from the server. If you plan to modify the server's response in your handler for this event, you must first disable streaming of the response. bBufferResponse to true set this event handler will not be called if the response was streamed to the client. If you plan to modify the server's response in your handler for this event, you must first disable streaming of the response. bBufferResponse to true set this event handler will not be called if the response was streamed to the client. ResponseHeadersAvailable and BeforeResponse events, which is a default in FiddlerCore. This change only takes effect properly if oS.bBufferResponse is set at the earliest! oS.utilReplaceInResponse("", "INJECTED!!!"); } After event handlers are ready, your next step is to start the FiddlerCore proxy. Just calculate the desired behavior flags: // The default flags are your best bet! FiddlerCoreStartupFlags oFCSF = FiddlerCoreStartupFlags.Default; // ...but if, say, we won't want FiddlerCore to Decrypt // HTTPS traffic, we can unset that flag at this point oFCSF = oFCSF & ~FiddlerCoreStartupFlags.DecryptSSL); AfterSessionComplete Event This event fires when the processing of a Session is complete. FiddlerAttach Event This event fires when FiddlerCore has registered as the system's default proxy. FiddlerDetach Event This event fires when FiddlerCore has unregistered as the system's default proxy. OnClearCache Event This event fires when the Fiddler.WinINETCache.ClearCacheItems method is called; properties of the EventArgs object indicate what elements of the cache should be cleared. The event object is silent and the cache should be cleared. The event object is silent mode by default, most notifications are raised using this event. Developers often contact me to complain that "FiddlerCore isn't working properly." After attaching a handler for this event, the source of the problem is often obvious. FiddlerApplication Methods Startup() This method is used to configure the default FiddlerCore proxy listener and instruct it to begin capturing traffic. Various overloaded versions are available-- I recommend that you use the overload that accepts FiddlerCoreStartupFlags. Shutdown() This method detaches the default FiddlerCore listener (if attached) and disposes of it. IsStarted() This method returns true if the default FiddlerCore listener is running and attached as the system proxy. CreateProxyEndpoint() This method allows you to create additional FiddlerCore proxy endpoints that can accept requests from clients. When calling this method, supply a port number and optionally either a string containing a hostname or an X509Certificate2. If either is supplied, the endpoint will automatically perform a HTTPS handshake with the client on each connection, presenting either the provided certificate or an automatically generated certificate containing the specified hostname. Use this overload when running a reverse proxy for a HTTPS site. Proxy oSecureEP = FiddlerApplication.CreateProxyEndpoint(8877, true, "localhost"); if (null != oSecureEP) { FiddlerApplication.Log.LogString("Created secure endpoint listening " + "on port 8877, which will use a HTTPS certificate for 'localhost'"); } If you do not supply a hostname or certificate, the endpoint will act as a normal proxy endpoint and will not masquerade as a HTTPS server. 253 | FiddlerCore These additional endpoints created using this method will not be processed using the normal Session-processing event handlers. Your Session-processing event handlers can determine which endpoint was received upon by examining its LocalPort property. FiddlerApplication.Log.LogFormat("Session {0} received by EndPoint on Port #{1}", oSession.id, (int) oSession.oRequest.pipeClient.LocalPort); DoImport() This method enables import of Sessions using the Importer architecture. When calling this method, supply an Import Format name, a bool controlling options, and a progress event callback to receive updates on the progress of the import. DoExport() This method enables export of Sessions using the Exporter architecture. When calling this method, supply an Export Format name, the array of Sessions, a StringDictionary containing options, and a progress event callback to receive updates on the progress of the export. GetVersionString() This method returns a string describing the FiddlerCore version. For instance: Fiddler/2.3.9.9 (+GetDetailedInfo) This method returns a string describing the configuration of the AddRef / AddRefmatic subsystem. 254 | FiddlerCore FiddlerCore DefaultClientCertificate When set, FiddlerCore will use the supplied X509Certificate when performing client authentication to a HTTPS server. oProxy This object represents Fiddler's default listening proxy endpoint. Its SendRequest methods allow your application to generate new Sessions. uTranscoders This object represents Fiddler's ImportExport subsystem. Your application may register Transcoder assemblies using the ImportTranscoders method on this object. Invoke Import or Export operations using the DoImport and DoExport methods of the FiddlerApplication object. Prefs This object provides access to Fiddler's Preferences system. See Programming with Preferences on page 210 for more information. The Rest of the FiddlerAPI With the exception of a few FiddlerApplication-specific APIs, Fiddler and FiddlerCore share a common object model. Code developed as a Fiddler extension can often be ported to run in a FiddlerCore-based application simply by eliminating any dependencies upon Fiddler user-interface objects. 255 | FiddlerCore COMMON TAS KS WITH FIDDLERCORE Keeping track of Sessions Unlike Fiddler, FiddlerCore does not keep a Session list automatically. If you want a Session list, simply create a List and add new Sessions to it as they are captured. FiddlerCore does not keep a Session list automatically. If you want a Session list, simply create a List and add new Sessions to it as they are captured. System.Collections.Generic List oAllSessions = new List(); // Add Sessions to the list as they are captured FiddlerApplication.BeforeRequest += delegate(Session oSession) { Monitor.Enter(oAllSessions); oAllSessions.Add(oS); Monitor.Exit(oAllSessions); }; application because Sessions cannot be garbage-collected while they are referenced in the list. You should periodically trim the list to keep it of a reasonable size. Alternatively, if you only care about request URLs or headers, you could keep a list of those types rather than storing references to Session objects. Getting Traffic to FiddlerCore Like Fiddler itself, FiddlerCore runs as a local proxy server, and it only sees traffic that is sent to it. The RegisterAsSystemProxy flag is set when you call the FiddlerApplication.Startup method, FiddlerCore will automatically register as the system proxy, just like Fiddler does. If you do not configure mange endpoints to point at your FiddlerCore instance. FiddlerCore offers the unique ability to register as the WinINET proxy for just the process in which it is running. This is a useful capability for scenarios where you have a Web Browser control in your .NET application and want to use FiddlerCore to capture and manipulate the traffic from that Web Browser control (and any other WinINET APIs). To make use of this capability, call the SetProxyInProcess method with your FiddlerCore endpoint's address and port, and a proxy-bypass list: Fiddler.URLMonInterop.ResetProxyInProcessToDefault(); 256 | FiddlerCore Trusting the FiddlerCore Certificate If your FiddlerCore application needs to decrypt HTTPS traffic, you can use the Fiddler.CertMaker class to generate a root certificate and prompt the user to trust it: private bool CreateAndTrustRoot() { // Ensure root exists if (!FiddlerApplication.CertMaker.rootCertExists()) { if (!FiddlerApplication.CertMaker.createRootCert()) return false; } // Ensure root is trusted if (!CertMaker.rootCertIsTrusted()) { if (!CertMaker.trustRootCert()) { if (!CertMaker.rootCertIsTrusted()) return false; } } return true; } Note that the following method: if (!CertMaker.rootCertExists()) return false; } // Ensure root exists if (!CertMaker.rootCertExists()) { if (!CertMaker.createRootCert()) return false; } The FiddlerCore root certificate on a machine-wide basis instead, call rootCertIsMachineTrusted() and trustRootCertMachine() methods instead. Both of these methods require that the application be running with Administrator privileges so all certificates to the machine-wide root certificate store. 257 | FiddlerCore Generating Responses FiddlerCore's AutoResponder API allows users to replay a previously captured Session. Hosts of FiddlerCore can reproduce the behavior of this feature in either of two ways. The following snippet identifies requests for replacement.txt and returns a previously stored SessionIWantToReturn: FiddlerApplication.BeforeRequest += delegate(Session oS) { if (oS.uriContains("replacement.txt")) { oS.utilCreateResponseAndBypassServer(); oS.responseBodyBytes = SessionIWantToReturn.responseBodyBytes; oS.oResponse.headers = (HTTPResponseHeaders) SessionIWantToReturn.oResponse.headers.Clone(); }; Other Resources See for more information on building FiddlerCore-based applications. 258 | FiddlerCore APPENDIX A: TROUBLESHOOTING Over the years, Fiddler has been refined to address most of the things that commonly caused problems for users. The single most common complaint I hear from users is: "Fiddler used to work but now it doesn't show anything. Please help!" In virtually all cases, the problem is that a filter is set that is causing the traffic to be hidden. Sometimes, the filter was set inadvertently, and sometimes the user merely set a filter weeks ago and forgot about it. Fortunately, it's easy to troubleshoot filters--just click the Troubleshoot Filters option on the Help menu. When Filter Troubleshooting is enabled, the Web Sessions list will include all Sessions that would ordinarily be hidden, rendering each filtered Session using the strikethrough font. The Comments column for each filtered Session will note which filter was responsible for hiding the traffic, allowing you to adjust filters as needed. If none of your traffic is shown 259 | APPENDIX A: Troubleshooting For instance, the following Sessions were hidden because they had a web browser process, and the status bar's Process Filter is set to show only Non-Browser traffic: If you don't see the missing Sessions even after enabling Filter Troubleshooting, then your next step is to try to catch the traffic in the application. If your client typically has a proxy setting in the Web Sessions list and the browser using an "echo" page if you don't see this page, that suggests that the client application is not using Fiddler as its proxy. Ensure that you are not inadvertently blocking the target network using Microsoft DirectAccess--that technology will bypass the local Fiddler proxy server. Your next step should be to try the URL using the same computer. If that URL cannot be reached, it means that your client application is unable to connect Fiddler, either because Fiddler is not properly running, or there is a firewall or other problem preventing Fiddler from receiving the traffic. You should check your firewall configuration, and see if you get different results from other clients or when running a different browser. If you don't see the traffic in Fiddler, the most likely explanation is that the client is not configured to send Firefox's Tools > Options > Network does not require a proxy. If you have configured Firefox to either use the system proxy or to send traffic to Fiddler, but you are still not seeing its traffic, you may have hit a known bug with Firefox and reinstalling Fiddler. If you wish to reset Fiddler's configuration manually, hold the SHIFT key while starting Fiddler, and click Yes in the Warning prompt. 262 | Appendices Wiping all traces of Fiddler Warning: As these instructions involve modifying sensitive areas of your system, they should only be undertaken by experts. You may be able to fix broken MakeCert.exe functionality by using the Systinternals Process Monitor utility to find MakeCert.exe file accesses within the folder path %USERPROFILE% Application Data Microsoft Crypto RSA whose Fiddler fails to generate a certificate. Moving the accessed files may restore the certificate generation functionality. If problems persist, you may be able to avoid the problem by switching to the Certificate Maker plugin described on page 103. This plugin does not utilize the Windows Certificate APIs for certificate generation, and thus may workaround the problem. 263 | Appendices Wiping all remnants of Fiddler from your system: 1. Uninstall Fiddler using the Add/Remove Programs list. 2. Delete Fiddler-related entries in the registry, listed in the Add/Remove Programs list. 3. Delete the %USERPROFILE% Application Data Microsoft Fiddler2 folder. 4. Delete the %USERPROFILE% My Documents Fiddler2 folder. 5. Use the Windows Certificate Manager (CertMgr.msc) to remove all of Fiddler's certificates from your Personal and Trusted Root Certification Authorities stores. 6. If you installed FiddlerCap, uninstall it too. 7. Inside IE's Tools > Internet Options > Connections > LAN Settings, ensure that none of the settings point at Fiddler. 8. Inside Firefox or Opera's Connection Settings, ensure that none of the settings point to Fiddler. Fiddler crashes complaining about the "Configuration System" The following error message indicates that one of the .NET Framework's configuration files is corrupt: -------- Sorry, you may have found a bug... ---------- Your Microsoft .NET Configuration file is corrupt and contains invalid data. You can often correct this problem by deleting the invalid .NET Framework configuration file. This error can occur if one or more machine.config or system.config files became corrupt or contain invalid XML. To fix this problem, you should try re-installing the .NET Framework. If that doesn't work, try re-installing the .NET Framework. If that doesn't work, try editing the XML file (e.g. machine.config) specified by the error message, to correct whatever the problem is in complaining about. 264 | Appendices Fiddler randomly stops capturing traffic The Microsoft Firewall client for ISA / Forefront TMG has an option to automatically reconfigure Internet Explorer to detect Fiddler at random times. To prevent this problem, disable automatic configuration in the Microsoft Firewall client settings. A Firewall client icon in your system tray, right click it and choose Configure... from the popup menu. If you don't see the icon, type forefront in the Start Menu search box to find the applet. On the Web Browser tab, uncheck the Enable Web browser automatic configuration checkbox. Fiddler stalls when streaming RPC-over-HTTPS traffic The HTTP protocols are extremely popular data transports; even scenarios that would otherwise use a straight TCP/IP connection often use HTTP or HTTPS because these protocols are usually allowed to traverse firewalls. Fiddler may encounter problems if some traffic is masquerading as HTTP. For example, one scenario where you will encounter

this is when using Microsoft Outlook with an Exchange Server configured to deliver mail using the RPC-over-HTTPS feature. You will see a request in Fiddler like so: RPC_OUT_DATA /rpc/rpcproxy.dll?ex.example.com HTTP/1.1 Host: ex.example.com Content-Type: application/rpc Content-Length: 1073741824 Outlook is attempting to use the HTTPS protocol to establish a bytestream over which a bidirectional TCP/IP stream will be layered. The problem is that the client isn't really planning to send a one gigabyte request body - instead just sending a huge value to indicate that it might send a lot of data. Fiddler dutifully waits for the client to send the promised gigabyte of data before proceeding with the network request. Since the data is not forthcoming from the client, no connection is made, and Outlook will indicate that it's "Trying to connect..." forever. While enabling the Stream option in Fiddler's toolbar enables responses to stream to the client, Fiddler does not currently offer a way to stream incomplete requests to the server. To enable Outlook's RPC-over-HTTPS feature to work correctly while Fiddler is running, you can: 1. Configure the target hostname to bypass Fiddler entirely: Add exchange.example.com to the Bypass List box on the Tools > Fiddler Options > Connections tab, or 265 | Appendices 2. Configure Fiddler to skip decrypting connections to this host: Add exchange.example.com to the Skip Decryption box on the Tools > Fiddler Options > Connections tab, or 3. Configure Fiddler to skip decrypting of traffic from the client application: Use the dropdown box on the Tools > Fiddler Options > Connections tab to enable decryption for browsers only. 266 | Appendices APPENDIX B : COMMAND LINE SYNTAX Fiddler.exe accepts zero or more command-line arguments, consisting of zero or more option flag arguments and a single filename of a SAZ file to load on startup. fiddler.exe [options] [File/Load.saz] Fiddler also registers itself in the Windows' AppPaths key so that you can launch it by typing fiddler2 in the Start > Run prompt instead of specifying a fully-qualified path to fiddler.exe. Option Flags Option flags may be preceded by either a / or - character. Flag /? /viewer /quiet /noattach /noversioncheck /extoff /noscript /port:#### Description Show the list of available command line arguments. Open a Viewer mode instance of Fiddler which can be used to display previously captured traffic but cannot capture new traffic. 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. 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. Do not send a write request to check for updates on startup. Do not load any Fiddler Inspectors or Extensions. This flag is used to troubleshoot problems related to buggy extensions. Do not load any FiddlerScript. This flag is used to determine whether your FiddlerScript is causing some problem. 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: fiddler2 /port:1234 -quiet Open a SAZ file in a new Fiddler Viewer instance: fiddler2 -viewer C:\users\joe\desktop\Sample.saz 267 | Appendices A P P E N D I X C : S E S S I O N F L A G S A StringDictionary field in each Session object contains flags that control the processing or display of the session. Some flags are set by Fiddler itself, but most are set script or extensions. The list of supported flags grows with each update to Fiddler, and extensions may use their own flags (which have no meaning to Fiddler) to add state information to a given Session. The flags can be accessed by oSession.oFlags["flagname"] or by using the default indexer on the Session object: oSession["flagname"]. Flag names are case-insensitive strings, and most flag values are interpreted case-insensitively. Most of Fiddler's flags are simply checked for their existence, such that setting any value (even misleading strings like 0, false, and heck no!), enables the named behavior. To disable a flag, remove the flag from the Session like so: oSession.oFlags.Remove("flagname"); Because most flags are simply tested for existence, a best practice is to use the flag's value to store a terse explanation of why the flag was set. For instance: oSession["ui-hide"] = "hidden by Hide Images rule"; You can view a Session's flags by using the Properties item on the Web Sessions list's context menu. Session Display Flags The following flags control how a session appears within the Web Sessions list. Flag Name Explanation Supported Values ui-hide The session will not appear within the Web Sessions list. One of the most commonly used flags, ui-hide is used by script or extensions to avoid cluttering the Web Sessions list with uninteresting traffic. Any value will hide the Session. Typically, you should provide a terse explanation of why the Session was hidden, so that if the user activates the Troubleshoot Filters feature on the Help menu, the UI will explain why the Session was hidden. By default, Fiddler will not hide requests that it itself generated (e.g. using the Composer). However, if the ui-hide flag's value contains the word stealth, the Session will be hidden unconditionally. 268 | Appendices Flag Name Explanation Supported Values ui-bold The Session will be rendered using bold text. Any value triggers the bold formatting. ui-italic Flag Name Explanation Supported Values ui-comments Supported Values Flag Name Explanation Supported Values The Session's list item in the Web Sessions list is rendered using bold text. Any value triggers this font formatting. The Session is rendered in italicized text. Any value triggers the font formatting. The Session is rendered in struck text. Any value triggers the font formatting. The Session's text of the specified color. The color string value may be a .NET Color constant like red, or may be a HTMLformatted color code, like #ff0000. The Session is rendered in text of the specified color. The color string value may be a .NET Color constant like red, or may be a HTMLformatted color code, like #ff0000. This string, set by FiddlerScript, extensions, or the Comment button on the toolbar, is shown in the Comments column in the Web Sessions list. The string that you'd like to display in the column. ui-customcolumn This string, set by FiddlerScript or extensions is shown in the Custom column in the Web Sessions list. The string that you'd like to display in the column. Breakpoint and Editing Flags Breakpointing and editing features are triggered by setting flags during the processing of the Session. Flag Name Explanation Supported Values x-BreakRequest When this flag is set on a Session before the request is sent to the server, Fiddler will pause the Session to allow you to use Inspectors to modify the request. 269 | Appendices Supported Values Any value will enable a request breakpoint. It is customary to set this string to a terse explanation about why this request was chosen to be breakpointed. Flag Name Explanation x-BreakResponse Supported Values When this flag is set on a Session before the response is sent to the client, Fiddler will pause the Session to allow you to use Inspectors to modify the response. Response tampering is incompatible with streaming. The response breakpoint is only checked after a streamed response has been returned to the client. Any value will enable a response breakpoint. It is customary to set this string to a terse explanation about why this response was breakpointed. x-Unlocked This flag, set by Fiddler's Unlock for Editing option, allows you to use Inspectors to modify a completed Session as if it were paused at a breakpoint. Any value will unlock the Session. Flag Name Explanation x-ClientIP Flag Name Explanation Supported Values x-overrideHost 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. Specify either an alternate hostname or an IP Address to which this request should be targeted. Setting this flag will have no effect if the request is sent to an upstream gateway, because the upstream gateway will perform its own DNS resolution. x-OverrideGateway Controls which upstream gateway proxy, if any, this request is sent to. 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. 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 Explanation 270 | Appendices x-ReplyWithTunnel 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. Supported Values Flag Name Explanation Supported Values Flag Name Explanation Supported Values Flag Name Explanation x-SecurePipe Set by Fiddler, this flag contains the IP address of the server to which this request was sent. Set by the AutoResponder, this flag contains the original request URL in the event that the URL was changed by an AutoResponder rule. Set by Fiddler, this flag contains the local port number used to establish the connection to the server or gateway proxy. Set by Fiddler, this flag contains diagnostic information about the connection to the server or gateway proxy, including information about whether the connection was made. Supported Values x-EgressPort Flag Name Explanation x-ServerSocket Flag Name Explanation When a CONNECT tunnel is used. You've used either flag's use to expose HTTPS traffic to a different server and do not want the client to encounter an unexpected certificate error. Specify the hostname to use for the self-signed certificate generated by Fiddler to send to the client when a CONNECT tunnel is created. This flag is most useful when you've used either flag's use to expose HTTPS traffic to a different server and do not want the client to encounter an unexpected certificate error. Specify the hostname to use for the self-signed certificate generated by Fiddler to send to the client and HTTP/401 challenges sent by upstream proxies. This flag supports the Digest, NTLM, and Negotiate protocols. When set to (default), the current logon user's credentials will be used. To use different 271 | Appendices Supported Values credentials, set the flag to a string in the format username:password. Warning: If Fiddler is configured to accept requests from other devices or user-accounts, use of (default) introduces a security vulnerability. That's because those requests will be authenticated using the credentials of the account in which Fiddler is running. Flag Name Explanation Supported Values Flag Name Explanation x-AutoAuth-Retries When the x-AutoAuth is set, this flag specifies the maximum number of attempts Fiddler will make to supply the credentials to HTTP/401 authentication challenges. Any positive integer; the default value is 5. x-AutoAuth-Failed This flag is set if Fiddler fails to successfully authenticate to the server because the X-AutoAuth-Retries limit was reached. Client Information Flags The following flags control contacting the server or gateway. Supported Values Flag Name Explanation Supported Values x-ProcessInfo Flag Name Explanation Supported Values Typically, this will be the current PC's IP address, unless remote connections have been permitted using the option in the Fiddler Options > Connections tab. Set by Fiddler, this flag contains the name and ID of the local process which issued the request to Fiddler. Set by Fiddler, this flag contains the ID of the process which issued the request to Fiddler. Performance Simulation Flags Fiddler offers simple flags that allow you to roughly simulate limited-bandwidth connections like modems, DSL, or Satellite connections. Flag Name Explanation Supported Values 272 | Appendices request-trickle-delay Set this flag to control the rate at which Fiddler writes the request headers and body to the server. Fiddler will ensure that every 1kb of data written to the server is delayed by the amount specified. Specify this flag's value as an integer number of milliseconds. For instance, the value 300 will result in Fiddler sending the request to the server in 1kb chunks, with a 150ms delay before the chunk and a 150ms delay after the chunk. Flag Name Explanation Supported Values response-trickle-delay Set this flag to control the rate at which Fiddler writes the response headers and body to the client. Fiddler will ensure that every 1kb of data written to the client is delayed by the amount specified. Specify this flag's value as an integer number of milliseconds. For instance, the value 300 will result in Fiddler sending the response to the client in 1kb chunks, with a 150ms delay before the chunk and a 150ms delay after the chunk. HTTPS Flags The following flags are related to HTTPS handling in Fiddler. Several of the flags are set automatically by Fiddler to reflect information about the HTTPS connection, while others can be set by script or extensions to influence the connection. Flag Name Explanation Supported Values x-no-decrypt When set on a CONNECT tunnel, this flag instructs Fiddler to not attempt to decrypt the HTTPS traffic that flows through the tunnel. Any value will disable decryption for the CONNECT tunnel. Typically, you should provide a terse explanation of why the CONNECT's traffic will not be decrypted so that a user examining the CONNECT Session's properties can understand why its traffic is not being decrypted. x-OverrideCertCN When HTTPS decryption is enabled, this value controls the hostname that is used for the self-signed certificate generated by Fiddler to send to the client when a CONNECT tunnel is created. This flag is most useful when you've used either flag's use to expose HTTPS traffic to a different server and do not want the client to encounter an unexpected certificate error. Specify the hostname to use for the certificate. For instance www.example.com. x-IgnoreCertCNMismatch This value controls whether Fiddler validates that the server's certificate's Subject Common Name (CN) field matches the hostname expected for the target server. This flag is meaningful only for CONNECT tunnels and only when HTTPS decryption is enabled. Set this flag when you have encrypted HTTPS traffic to a different server and do not want Fiddler to flag a certificate-hostname mismatch. Any value will disable Certificate Subject CN validation. Typically, you should provide a terse explanation of why Subject CN validation was disabled, so that a user examining the CONNECT Session's properties will understand why the flag was present. x-IgnoreCertErrors When HTTPS decryption is enabled, this value controls whether Fiddler validates that the server's certificate is unexpired, chains to a trusted root, and is valid for the target hostname. This flag is meaningful only for CONNECT tunnels and only when HTTPS decryption is enabled. Set this flag when interacting with Test or Development servers which are using self-signed or otherwise invalid certificates. To enable this behavior globally (unsafe), tick the Ignore server certificate errors checkbox inside Tools > Fiddler Options > HTTPS. Any value will disable warnings about Certificate Errors on the CONNECT tunnel. Typically, you should provide a terse explanation of why certificate errors are being ignored, so that a user examining the CONNECT session's properties will understand why the flag was present. x-OverrideSslProtocols Controls what SSL and TLS versions will be advertised when talking to the server Provide a semi-colon delimited string containing one or more of the following tokens: ssl2 ssl3 tls1.0 For example: ssl2;ssl3. The provided versions will be used instead of the default versions controlled by the CONFIG.oAcceptedServerHTTPSProtocols value (Ssl3 and Tls1.0 by default). Flag Name Explanation Supported Values https-Client-Certificate Set this flag on a CONNECT tunnel containing the location of a certificate file that Fiddler should use if the server prompts for HTTPS Client Authentication. The .CER file must have a matching private key in the Windows Certificate store. If this flag is not present, Fiddler will look for the file %USERPROFILE%\My Documents\Fiddler2 \ClientCertificate.cer and attach any certificate it finds. Specify a fully-qualified path to the file. If you're setting this property from script or C#, remember to escape backslashes by doubling them up. E.g. oS["https-Client-Certificate"] = "C:\\test\\someCert.cer"; Flag Name Explanation Supported Values https-Client-SNIHostname 274 | Appendices Set by Fiddler on a CONNECT tunnel, this flag reports the HTTPS Session ID sent by the client in the HTTPS handshake. Set by Fiddler on a CONNECT tunnel, this flag contains the hostname sent by the client in a TLS Handshake's Server Name Indication TLS extension, if any. Flag Name Explanation X-HTTPS-Decryption-Error Set by Fiddler when a given connection is secured by a client certificate, this string contains that certificate's Subject Field and Serial number. This flag is set by Fiddler when it is unable to find or generate a self-signed HTTPS interception certificate. Request Composer Flags While these flags were designed for use by the Request Composer, your script or extensions may use them as well. Flag Name Explanation Supported Values x-Builder-Inspect When this flag is set by the Composer when the Inspect Session option is enabled) the Session will be automatically inspected when added to the Web Sessions list. Any value will trigger the inspection. X-Builder-MaxRedir Controls the number of HTTP/3xx redirections Fiddler will follow before giving up. Any positive integer value. The default is 0. X-From-Builder This flag is set if the request was built and issued by the Request Composer. Other Flags Flag Name Explanation Supported Values Flag Name Explanation Supported Values memory when Session processing completes. Any value will cause the Session to be dropped after Session processing completes. log-drop-request-body When this flag is set, Fiddler "forgets" the request body to save memory when Session processing completes. Any value will cause the request body to be dropped after Session processing completes. log-drop-response-body When this flag is set, Fiddler "forgets" the response body to save memory when Session processing completes. X-Fiddler-Stream1xx When the fiddler.network.leakhttp1xx preference is set to true (its default), Fiddler will stream any HTTP/1xx intermediate responses from the server to the client. This flag's value, if present, indicates which HTTP/1xx codes were returned. 275 | Appendices Flag Name Explanation x-Fiddler-Streaming Flag Name Explanation Supported Values X-Divorced-ServerPipe Flag Name Explanation x-Original-Host Set by Fiddler on a CONNECT tunnel, this flag reports the HTTPS Session ID sent by the server in the HTTPS handshake. Set by Fiddler on a CONNECT tunnel, this flag contains the hostname sent by the client in a TLS Handshake's Server Name Indication TLS extension, if any. Flag Name Explanation X-HTTPS-Decryption-Error Set by Fiddler when a given connection is secured by a client certificate, this string contains that certificate's Subject Field and Serial number. X-Fiddler-Aborted Flag Name X-TTFB 276 | Appendices When the fiddler.network.leakhttp1xx preference is set to false, Fiddler will "swallow" any HTTP/1xx intermediate responses from the server. This flag's value, if present, indicates which HTTP/1xx codes were swallowed. When set, this flag indicates that Fiddler was forced to use a different connection to the server to process the request. This might occur, for instance, if FiddlerScript or an extension changed a HTTP request to a HTTPS request, or an existing secure connection cannot be reused because it has been closed. Set by Fiddler, this flag is present if Fiddler itself generated the response with the utilCreateResponseAndBypassServer method. If present, this flag will contain a string indicating the number of times that Fiddler failed over to avoid DNS record when attempting to contact the host. A +1 is added to the string each time this occurs, up to the maximum number of DNS failover attempts. This flag is set if the inbound request violated the HTTP protocol by specifying a different hostname in the request URL and the Host request header. This flag will contain the original Host header value. The request's Host header will be overwritten with the hostname specified in the request URL. This flag is set to the virtual hostname value that was specified when the client issued the CONNECT request. x-Fiddler-SessionID Flag Name Explanation Supported Values Session and choosing Abort Session from the context menu will call Abort. Explanation Legacy flag reports the number of milliseconds until the first byte was read from the server. Flag Name Explanation X-TTLB Flag Name Explanation Supported Values x-ReplyWithFile Supported Values when Session processing finishes. The X-RequestBodyLength flag records the request body bytes count at the point that the body is dropped. If the log-drop-response-body flag is set, Fiddler "forgets" the response body to save memory when Session processing finishes. The X-ResponseBodyFinalLength flag records the response body byte count at the point that the body is dropped. This flag controls the automatic reformatting of JavaScript performed by the JavaScript Formatter extension. When this flag is set to a value of 0, the JavaScript Formatter extension will not format the response body, even if the Make JavaScript Pretty option is enabled on the Rules menu. Any other value will cause the JavaScript response body to be formatted, even if the Make JavaScript Pretty option is not enabled on the Rules menu. 277 | Appendices APPENDIX D : P REFERENCES 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 network behavior: Name Default Explanation fiddler.network.timeouts.dnscache.reuse Name Default Explanation fiddler.network.timeouts.clientpipe.receive.reuse Name Default Explanation fiddler.network.timeouts.serverpipe.receive Name Default Explanation Initial Name Default Explanation fiddler.network.timeouts.serverpipe.receive.reuse 278 | Appendices Number of milliseconds for which Fiddler should cache DNS lookup results. 120000 (2 minutes) 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. 60000 (1 minute) Number of milliseconds for which Fiddler is willing to wait for a client to begin sending a previously used connection. After this timeout expires, Fiddler will close the connection from the client. 30000 (30 seconds) Number of milliseconds for which Fiddler is willing to wait for a server to begin sending a response on a newly established connection. After this timeout expires, Fiddler will close the connection to the server. Name Default Explanation fiddler.network.egress.IP Set this preference if your computer has multiple outbound connections and you need requests to be sent on a particular outbound connection. This preference is rarely used, but is sometimes needed if you have particular routing needs. For instance, if you have tethered a 3G phone to your PC, you may want Fiddler to send outbound requests from its IP address so that traffic is sent from the 3G connection. Warning: If you set this preference to an IP address that is not currently bound to one of your network connections, all outbound requests will fail. Name Default Explanation fiddler.network.auth.ReuseMode 0 This preference controls whether Fiddler is willing to reuse server connections upon which an authentication (HTTP header authentication or HTTPS client certificate authentication) has taken place. The default value of 0 requires "Process Affinity." Fiddler will only reuse the authenticated connection to service requests that originated from the same process that authenticated the connection originally. The value 1 requires "Client Connection Marriage." Fiddler will only reuse the authenticated connection to service any request from any client. Warning: This value will yield better performance but unexpected and insecure behavior. Name Default Explanation fiddler.ftp.UseBinary When set to true, Fiddler (acting as a FTP gateway) will use BINARY mode for file transfers. When set to false, Fiddler will use ASCII mode instead. To control this option on a per-request basis, set the FTP-UseASCII flag on the Session object. Name Explanation Name Default Explanation fiddler.ftp.UsePassive Use When set to true, Fiddler will use the PASV option when acting as a FTP gateway. When set to false, Fiddler will make active FTP connections, which are not compatible with most firewalls. fiddler.network.sockets.ClientReadBufferSize 8192 279 | Appendices This preference controls the size, in bytes, of the buffer used to read from client connections. Name Default Explanation fiddler.network.sockets.ServerReadBufferSize 32768 Name Default Explanation fiddler.network.proxy.RegistrationHostName 127.0.0.1 This preference controls the size, in bytes, of the buffer used to read from server connections. This preference controls the hostname used to register as the system proxy. There are some cases where you will need to register as the system proxy using a different hostname. For instance, the Windows Phone Fiddler will build a connection if the first IP address cannot be reached. When this preference is set to true, Fiddler will try up to MaxAddressCount addresses before giving up. Name Default Explanation fiddler.network.dns.MaxAddressCount 5 Name Default Explanation This preference allows you to control the maximum number of addresses returned, which in turn limits the number of DNS fallbacks that are permitted when Fiddler cannot reach the machine name of the current machine. Name Default Explanation fiddler.network.dns.fallback true A DNS server may return multiple addresses for a single hostname resolution. This preference controls whether Fiddler will automatically pass HTTP/1xx messages through to the client connection as they are read from the server. HTTP/1xx responses are "non-final" headers-only responses and they are not shown in Fiddler's UI. HTTP/100 Continue responses are sent by a server in response to a Continue header on POST requests. Name Default Explanation fiddler.network.streaming.AbortIfClientAborts false This preference controls whether Fiddler will close the server connection when the client connection closes during streaming. This configuration can significantly reduce the amount of memory used by Fiddler by aborting downloads that the client application has cancelled. Name Default Explanation fiddler.network.streaming.ForgetStreamedData When set to true, Fiddler will "forget" response data from the server as soon as that data is sent to the client. This precludes you from using Fiddler to examine streamed responses, but significantly decreases memory usage when streaming video or other large responses through Fiddler. Name Default Explanation fiddler.network.gateway.UseFiddlerAutoProxy false Fiddler uses the WinHTTP library for proxy auto-configuration support (either via WPAD or a manually-specified PAC script). When this preference is set to true, Fiddler will direct WinHTTP to perform proxy determination in-process rather than calling out to the WinHTTP service. By default, Fiddler will begin to ignore an automatically-detected upstream proxy script if it returns errors; requests will instead be sent directly to the target. Set this preference to true to disable that optimization and always attempt to use a proxy script even if it is non-responsive. Name Default Explanation fiddler.proxy.pacfile.UseFileProtocol true When the Use PAC Script option is enabled on the Fiddler Options > Connections tab, Fiddler will register as the system proxy by providing a proxy auto-configuration script URL. By default, the URL points to the BrowserPAC.js file in your Documents folder. When this preference is set to false, Fiddler will instead register using a HTTP URL, in the form then respond to inbound requests for the script using the AutoResponder or FiddlerScript. Name Default Explanation fiddler.proxy.pacfile.text return 'PROXY 127.0.0.1:8888'; 281 | Appendices This preference controls the body of the FindProxyForURL function stored in the %UserProfile%\My Documents\Fiddler2\Scripts\BrowserPAC.js file. HTTPS Preferences The following preferences control Fiddler's behavior when interacting with HTTPS connections. Name Default Explanation fiddler.network.https.SetCNPromSslChainValueIdentificationRevocation false Name Default Explanation true, Fiddler will walk the entire certificate chain provided by the server and add information about each certificate in the chain to the text shown in the Response Inspector for the CONNECT tunnel. When set to true, Fiddler will perform a revocation check to determine whether the server's certificate has been revoked by the issuer. While validation enhances security, it will slow down HTTPS connection establishment. When set to true, Fiddler will use the Server Name Indication TLS extension from the client's HTTPS handshake to set the CN field of the certificate used in the HTTPS handshake with the client. When set to false, the CN field will be set using the request's Host header. This preference can be handy if you have a client which sends a Host header that contains a literal IP address rather than the hostname of the target server. This may occur, for instance, if you've used iptables to send traffic to Fiddler from an Android device that does not accept a proxy setting. Name Default Explanation fiddler.network.https.NoDecryptionHosts Name Default Explanation fiddler.network.https.BlindTunnelIfCertUnobtainable true Name Default Explanation fiddler.network.https.CertMaker.CleanupServerCertCache false A semi-colon delimited list of hostnames for which HTTPS decryption should not take place. This preference is exposed by the textbox on the Fiddler Options > HTTPS tab. When set to true, Fiddler will fail the HTTPS connection if the certificate cannot be obtained. 282 | Appendices When HTTPS decryption is enabled, the default certificate maker will add certificates to the user's certificate store. When this preference is set to true, all such certificates will be removed when Fiddler shuts down. This preference is used by FiddlerCore; instead call CertMaker.removeFiddlerGeneratedCerts(false); before your application exits. Name Default Explanation fiddler.CertMaker.Assembly %Program Files%\Fiddler2\CertMaker.dll Name Default Explanation fiddler.CertMaker.Root.ExtraParams Name Default Explanation fiddler.CertMaker.EE.ExtraParams Name Default Explanation fiddler.CertMaker.OfferMachineTrust When this preference is set, Fiddler will attempt to load specified assembly when a certificate is needed to secure a HTTPS connection. If the specified assembly cannot be found or does not contain a public class that implements the ICertificateProvider2 interface, Fiddler's default certificate provider will be used instead. Specifies any extra parameters that should be passed to makecert.exe when generating endentity certificates for HTTPS servers. false except on Windows 8. Set this preference to true to configure Fiddler to offer to use an elevated process to add the root certificate to the per-Machine Trusted Root certificate store. By default, Fiddler prompts you to add a Root Certificate to the per-User Trusted Root certificate store. Name Default Explanation fiddler.network.https.cachecaientcert true Name Default Explanation fiddler.config.path.defaultclientcert %USERPROFILE%\My Documents\Fiddler2\ClientCertificate.cer Contains the path to the certificate used to secure a HTTPS connection. When this preference is set to true, Fiddler will cache, for the lifetime of the Fiddler process, the default client certificate upon its first use. Caching ensures that you will not be prompted with a PIN request if your certificate requires one. However, caching also means that you cannot use a different certificate without restarting Fiddler or using script to null the property FiddlerApplication.oDefaultClientCertificate. If a HTTPS server demands a client certificate, Fiddler will look for a client certificate file in this location. 283 | Appendices Name Default Explanation fiddler.network.https.clientcertificate.ephemeral.prompt-for-missing true Name Default Explanation fiddler.network.https.RequestClientCertificate false When this preference is set to true, Fiddler will prompt the user if a client certificate is requested but not provided when interacting with a HTTPS server. When set to false, Fiddler will request a client certificate every time a client attempts to establish a HTTPS connection. Set this preference to test client software to see how it handles client certificate demands. Fiddler UI Preferences Preferences under the Fiddler.UI branch track the state of options and settings that are shown within the Fiddler user interface, including the toolbar. Name Default Explanation fiddler.ui.toolbar.visible true Name Default Explanation fiddler.ui.toolbar.ShowLabels true Name Default Explanation fiddler.ui.StayOnTop false Name Default Explanation fiddler.ui.font.size 8.25 Name Default Explanation fiddler.ui.layout.inspectors.HideList This preference controls whether the toolbar is shown on the system. Controls the default font size for text in Fiddler. It can be adjusted using the Tools > Fiddler Options > Appearance > Font Size dropdown. Stores the last active View when Fiddler is shutdown. That tab is subsequently reactivated the next time Fiddler is started. Stores the list of Inspectors which should be hidden. You can hide an Inspector by right- 284 | Appendices clicking on it and choosing Hide Inspector from the menu. Unhiding an Inspector by deleting this preference requires that you restart Fiddler. Name Default Explanation fiddler.ui.rules.HideImages false Name Default Explanation fiddler.ui.rules.HideConnects false Name Default Explanation fiddler.ui.inspectors.request.AlwaysUse When set to true, Fiddler will reset all filters each time the debugger is launched. This option is helpful if you're in the habit of applying filters, using them for a while, closing Fiddler, restarting it days later, forgetting you've applied filters, and then spending frustrated hours trying to figure out why Fiddler isn't showing you traffic. When set to a non-zero value, Fiddler will automatically trim the Web Sessions list to the specified number of Sessions. This preference backs the Keep dropdown on the toolbar. Stores the Rules > Hide Images menu option. Stores the Rules > Hide CONNECTs menu option. Stores the Rules > Remove All Encodings menu option and the Decode toolbar toggle. Stores the Rules > Apply GZIP Encoding menu option. Stores the Stream toolbar toggle. Stores the Rules > Require Proxy Authentication menu option. Use this preference to always choose a particular Request Inspector when activating a Session. Set this preference to the Inspector tab's title text, e.g. Raw. 285 | Appendices Name Default Explanation fiddler.ui.inspectors.response.AlwaysUse Name Default Explanation fiddler.reissue.AutoRedirCount 0 Name Default Explanation fiddler.differ.UltraDiff true Name Default Explanation fiddler.differ.Params "{0} {1}" Use this preference to always choose a particular Response Inspector when activating a Session. Set the preference to the Inspector tab's title text, e.g. WebView. When set to true, Fiddler will attempt to respond to authentication challenges when you reissue a request from the Web Sessions list. Specifies the maximum number of redirections Fiddler will perform if a HTTP/3xx redirect response is returned when a Session is reissued using the context menu or toolbar. When the redirection limit is exceeded, Fiddler will ignore the Location header specified on the final redirect. The UltraDiff option is used when comparing two Sessions. When set to true, Fiddler reorders the headers of both Sessions such that headers that exactly match in both Sessions are listed first, then headers with different values are listed, then headers that are entirely different are listed last. When set to false, this reordering does not occur. Specifies the command line arguments to supply to the file comparison tool when the Compare command is invoked. {0} is replaced with the first filename and {1} is replaced with the second filename. Specifies the command line arguments to supply to the file comparison tool when the Compare command is invoked while holding the SHIFT key. The default value adds the -p parameter to instruct WinDiff to use its "break on punctuation" feature. fiddler.ui.ephemeral.rules.BreakOnRequest false Name Default Explanation fiddler.ui.ephemeral.rules.BreakOnResponse false Name Default Explanation Stores the Help > Troubleshoot Filters... menu option. When set to true, Sessions that would normally be hidden are instead shown in the Web Sessions list using a strikethrough font. 286 | Appendices Name Default Explanation fiddler.ui.LastView.UpdateInterval 80 Name Default Explanation fiddler.session.prependIDToSuggestedFilename false Name Default Explanation fiddler.find.ephemeral.LastSearch When set to a positive integer, you will search the Web Sessions list. When set to true, hitting CTRL+X to clear the Web Sessions list will leave behind any pinned Sessions. For instance, when saving Session #4's response to disk, the default filename of download.exe becomes 4 download.exe when this preference is set to true. Stores the last search term supplied to Fiddler's Edit > Find Sessions prompt. An extension might choose to use this value to pre-fill its own inline search box for convenience. FiddlerScript Preferences Name Default Explanation fiddler.script.AutoRef true 287 | Appendices This preference is set to true, Fiddler will automatically determine which assemblies to reference based on the import statements at the top of the FiddlerScript file. When set to false, library references must be manually provided in the Fiddler Options > Extensions tab. Name Default Explanation fiddler.script.CompileToFilename Specify a fully-qualified path that contains assemblies your FiddlerScript will use. For instance, set the value to C:\src\library.dll and when your FiddlerScript is next compiled, Fiddler will look for referenced assemblies in the library DLLs folder. Set this preference to a fully-qualified .DLL filename and upon next compilation, your FiddlerScript will generate an assembly of the provided name. Use this option to create an assembly that you can load into Reflector or another code-inspection tool to inspect the compiled output. Warning: When you use this preference, your FiddlerScript will not execute until the preference is removed and you recompile your script. FiddlerScript Preferences Name Default Explanation fiddler.textwizard.OutputEncoding UTF-8 Specify a character set from which %-encoded characters will be decoded by the TextWizard. The string value provided must be an Encoding name recognized by the .NET Framework; valid values can be found at . Specify a character set to %-encoded by the TextWizard. The string value provided must be an Encoding name recognized by the .NET Framework; valid values can be found at . Request Composer Preferences Name Default Explanation fiddler.composer.FollowRedirects true When checked the Composer will automatically follow a redirect when the server returns one for a request sent by the Composer. Set this preference using the Composer's Options tab. Controls the number of redirects that the Composer will follow before giving up. This preference sets the X-Builder-MaxRedir flag on the new Session. 288 | Appendices Name Default Explanation fiddler.composer.AutoAuth true When set to true, the Composer will automatically attempt to respond to Authentication challenges sent by servers. This preference sets the x-AutoAuth flag on the new Session. Controls whether the Composer will automatically attempt to respond to Authentication challenges sent by servers. Set this preference to the x-AutoAuth flag on the new Session. When set to (default), the current logon user's credentials will be used. To use different credentials, set the preference to a string in the format username:password. Name Default Explanation fiddler.composer.InspectSession true Controls whether the Composer will automatically activated when you send a request using the Composer. Set this preference on the Composer's Options tab. Controls whether the Composer will automatically activated when you send a request using the Composer. Set this preference on the Composer's Options tab. Some paths may be configured using the user-interface, but some can only be controlled using preferences. Name Default Explanation fiddler.config.path.captures %USERPROFILE%\My Documents\Fiddler2\Captures Name Default Explanation fiddler.config.path.requests %USERPROFILE%\My Documents\Fiddler2\Captures\Requests Name Default Explanation fiddler.config.path.responses %USERPROFILE%\My Documents\Fiddler2\Captures\Responses Path under which captured requests and responses will be stored. Path to the file comparison tool used when comparing two Sessions. Path to the client certificate which will be used to respond to any certificate demands during HTTPS handshaking. Path under which captured requests and responses will be stored. Path under which captured responses will be stored. 289 | Appendices Name Default Explanation fiddler.config.path.templateresponses %ProgramFiles%\Fiddler2\ResponseTemplates\ Name Default Explanation fiddler.config.path.TextEditor notepad.exe Path where which Template Responses used by the AutoResponder are stored. Path to the editor to launch when viewing textual content. Miscellaneous Name Default Explanation fiddler.websocket.ShowFreshness true Name Default Explanation fiddler.SAZ.AES.Use256Bit false Name Default Explanation fiddler.ui.inspectors.response.raw.TruncateText 262144 Name Default Explanation fiddler.ui.inspectors.response.raw.TruncateBinaryAt 128 Name Default Explanation fiddler.inspectors.request.raw.TruncateTextAt When the AutoTruncate feature is enabled, the Raw Inspector will display at most the specified number of bytes of a Request body that is believed to be textual in nature. When set to true, you will be prompted for permission to check for a new version each time you launch a version of Fiddler build. Stores the Cache-Freshness true Name Default Explanation Name Default Explanation fiddler.websocket.ParseMessages true Name Default Explanation ParseFreshness When set to true, you will be prompted for permission to check for a new version each time you launch a version of Fiddler build. Displays the bytes of the WebSocket messages in the Log tab. When set to true, any exceptions raised when Fiddler calls into extensions will be shown in a notification window. When set to false, exceptions raised when calling into extensions will be silently discarded. When set to true, all extension-related messages will be logged on Fiddler's Log tab and any exceptions raised when Fiddler calls into extensions will be shown in a notification window. The set this preference to true when developing and debugging Fiddler extensions. When set to true, Fiddler will store password-protected SAZ files using 256-bit encryption instead of the default of 128-bit encryption. The higher level of encryption is much slower and is probably unnecessary if your adversary is not a government intelligence agency. 290 | Appendices Name Default Explanation fiddler.script.delaycreate true When set to true, Fiddler will create the CustomRules.js file only when you attempt to edit the script using Rules > Customize Rules. This ensures that all the script you always have the latest script even after the initial SampleRules.js file only when you attempt to edit the script using Rules > Customize Rules. This ensures that all the script you always have the latest script even after that control their behavior. The following is a list of commonly used Preferences that modules. Name Inspector fiddler.inspectors.response.raw.TruncateBinaryAt 128 Name Default Explanation fiddler.inspectors.request.raw.TruncateTextAt 262144 When the AutoTruncate feature is enabled, the Raw Inspector will display at most the specified number of bytes of a Request body that is believed to be textual in nature. When the AutoTruncate feature is enabled, the Raw Inspector will display at most the specified number of bytes of a Request body that is believed to be binary in nature. When the AutoTruncate feature is enabled, the Raw Inspector will display at most the specified number of bytes of a Response body that is believed to be textual in nature. JavaScript Formatter fiddler preferences.JSFormat.AutoFormat Name fiddler Default Explanation When set to true, the JavaScript Formatting extension will attempt to automatically autoformat all script files before they are returned to the client application. When set to false, you must manually format any JavaScript files using the Make JavaScript Pretty context menu item in the Web Sessions list. Certificate Maker The Certificate Maker add-on I offers many flags that enable it to generate certificates which are compatible with a wide variety of platform, including iOS and Android devices. In its default configuration, iOS-compatible certificates are generated. 291 | Appendices Name Default Explanation fiddler.CertMaker.BC.Debug false Name Default Explanation fiddler.CertMaker.BC.ReusePrivateKeys true Name Default Explanation fiddler.CertMaker.BC.AddCRL false Name Default Explanation fiddler.CertMaker.EE.CriticalAKID true Name Default Explanation fiddler.CertMaker.BC.Root.SetAKID false Name Default Explanation fiddler.CertMaker.BC.Root.CriticalBasicConstraints true Name Default Explanation fiddler.CertMaker.EE.CriticalBasicConstraints true Name Default Explanation fiddler.CertMaker.BC.EE.CriticalEKU true Name Default Explanation fiddler.CertMaker.BC.Root.CriticalBasicConstraints When set to true, the Certificate Maker plugin will emit logging information to the Log tab. When set to true, the same private key for each certificate it generates. This is obviously insecure, but harmless when debugging locally. When set to true, the Certificate Maker plugin will emit Private keys to the Log tab. When set to true, the Certificate Maker plugin will add a Certificate Revocation List URL to the certificates it generates. The CRL added is hardcoded to point to a nonexistent URL that you can intercept in Fiddler or FiddlerCore. When set to true, the end-entity certificates generated by the Certificate Maker plugin will have the BasicConstraints extension marked as Critical. When set to true, the end-entity certificates generated by the Certificate Maker plugin will have the ExtendedKeyUsage extension marked as Critical. When set to true, the Certificate Maker plugin will have an AuthorityKeyID to the root certificate which points at itself. When set to true, the root certificate generated by the Certificate Maker plugin will have the KeyUsage extension marked as Critical. When set to true, the Certificate Maker plugin will mark the root certificate's AuthorityKeyID as Critical. 293 | Appendices INDEX A Android HTTPS Decryption, 103 AppContainers, 83 Apple HTTPS Decryption, 103 iOS, 87 Mac OSX, 84 audio and video previewing, 135 streaming, 94 authentication Auth Inspector, 116 automatic, 109 channel-binding, 110 client certificates, 111 loopback, 82, 110 methods, 108 problems, 110 WinHTTP, 110 AutoResponder action text, 45 drag-and-drop, 47 FARX files, 47 match condition, 44 regular expressions, 46 B breakpoints Breakpoint bar, 66 Filters tab, 38 overview, 65, 127 QuickExec, 66 C caching Inspector, 118 294 | Appendices capturing .NET Framework, 79 browsers, 77 devices, 86 DirectAccess, 91 FTP, 107 Java, 80 loopback, 81 Mac OSX, 84 other PCs, 85 PHP/CURL, 81 reverse proxy, 88 VPNs/Modems, 91 Win8 apps, 82 Windows Phone, 87 Windows RT, 88 WinHTTP, 79 certificates CertMgr.msc, 100 troubleshooting, 263 trusted root, 99 Windows 8, 99 clipboard pasting from, 21 pasting to, 16 Clone Response, 18 code samples extensions overview, 227 implementing OnValidateServerCertificate, 250 scripts to manipulate requests, 189 scripts to manipulate responses, 191 transcode example, 237 trusting certificate, 257 columns, 13 adding from code, 181 COMETPeek, 18 overview, 95 Composer automated breakpoints, 54 options, 51 overview, 51 parsed, 52 saving, 52 sequential requests, 52 uploading files, 53 cookies Inspector, 119 CPU architecture and bitness, 92 E EnableLoopback.exe, 83 encoding decoding programmatically, 232 D Decoding using UI, 133 Decoding with Transformer, 132 headers, 75 HTTP Chunking, 131 HTTP Compression, 131 text encoding overview, 75 TextWizard, 49 encoding overview, 75 FiddlerScript, 172 saving with FiddlerScript, 204 searching Find Sessions window, 54 servers, 73 AnyWHERE, 149 assemblies, 232 best practices, 195 building in .NET, 194 Content Blocker, 143 delay-loading, 195 Developing, 225 example code, 227 FiddlerScript tab, 165 for Performance analysis, 138 for Security analysis, 138 Gallery tab, 141 installing, 141 third-party, 138 Traffic Differ, 144 F FiddlerApplication overview, 250 FiddlerCap, 154 FiddlerCore overview, 246 getting started, 247 overview, 226 startup, 248 ClassView sidebar, 146 cross-process messaging, 173 Editing, 145 engine, 208 example extending, 141 extensions API, 231 F FiddlerGencert.exe, 101 FiddlerHook, 77 FiddlerScript, 185 general functions, 170 JScript.NET, 166 overview, 166 references, 168 editor, 146 filtering Filters tab, 35 from code, 178 host, 36 process, 37 request headers, 39 response, 40 G GeoEdit, 144 151 HOST, 32 I iOS HTTPS Decryption, 103 HTTP_PROXY, 81 import and export, 20 importing other formats, 22 inspectors building, 239 overview, 115 ImageView, 125 JSON, 120 overview, 115 SyntaxView, 127 TextView, 124 WebView, 120 XML, 128 J JavaScript formatting, 140 K keyboard shortcuts, 12 L Log macro commands, 208 writing to, 207 loopback authentication, 82, 110 capturing, 81 capturing, 81 M memory analysis, 149 M Menus Context menu, 15 Edit menu, 20 extending with extensions, 227 extending with FiddlerScript, 176 File menu, 10 Help menu, 24 N NetLog, 8 NNTP, 8 noscript mode, 158 W Web Sessions list colors, 13 columns, 13 icons, 13 overview, 12 WebSockets Log tab, 55 overview, 96 windows About box, 25 Fiddler Options, 70 Find Sessions, 56 Windows 8 Metro capturing, 82 Windows Phone HTTPS Decryption, 103 Windows RT, 88 O Match menu, 24 Replay submenu, 17 Rules menu, 24 Tools menu, 23 View menu, 25 macros and replacement tokens, 209 CustomRules.js file, 165 editing requests, 187 editing responses, 191 creating menus, 176 overview, 165 properties and fields, 254 P port setting, 32 proxy chaining, 89 QuickExec commands, 28 extension integration, 228 general commands, 29 overview, 28 selecting sessions, 28 R reverse proxy, 88 S SAZ Files Connections tab, 72 exchanging, 20 Statistics, 34 TextWizard character encoding, 50 invoking from code, 207 Transcoders, 237 timing events, 236 supporting options, 235 V Viewer mode, 158 overview, 157 Index 227 | Appendices