Progress

Telerik Fiddler Core

DOCUMENTATION

Version: 5.0.2

Welcome to FiddlerCore

Thank you for choosing Progress® Telerik® FiddlerCore!

What Is FiddlerCore and Why Do I Need It?

FiddlerCore is a .NET class library you can integrate into your .NET applications. It's available in .NET Standard 2.0, .NET Framework 4.0, and .NET Framework 4.5 flavors, which allows using it on Windows, Linux, Mac, and any other platform implementing .NET Standard.

FiddlerCore allows you to capture and modify HTTP and HTTPS traffic just like Fiddler, without any of the Fiddler UI.

Features at a Glance

- HTTP and HTTPS traffic capture and modification.
- Powerful object model for content filtering and modification.
- Store and reload web traffic.
- Support for virtually any client application.
- Support for most devices via mobile proxy settings.

Next Steps

First steps with FiddlerCore

First Steps

This article explains everything you need to know about FiddlerCore before you start.

System Requirements

You can check the requirements for FiddlerCore suite in the System Requirements help article.

Download FiddlerCore

See how to download the library in the following help articles:

- Download Product Files
- Use Telerik NuGet Server

Next Steps

Check the system requirements

System Requirements

In order to develop applications with Telerik FiddlerCore you need to have the following development tools installed:

FiddlerCore for .NET Framework 4.0

- Windows XP or higher
- Microsoft Visual Studio 2013 or higher (optional)
- .NET Framework 4.0

FiddlerCore for .NET Framework 4.5

- Windows 8.1 or higher
- Microsoft Visual Studio 2015 or higher (optional)
- .NET Framework 4.0

FiddlerCore for .NET Standard 2.0

Windows

- Windows 8.1 or higher
- Microsoft Visual Studio 2017 or higher (optional)

Mac OS

- Mac OS Sierra 10.12 or higher
- Visual Studio for Mac (optional)
- XCode 8.3 or higher (for building Xamarin iOS applications)

Linux

- Full list of supported Linux distributions can be found here

Next Steps

- Download Product Files
- Use Telerik NuGet Server

Download Product Files

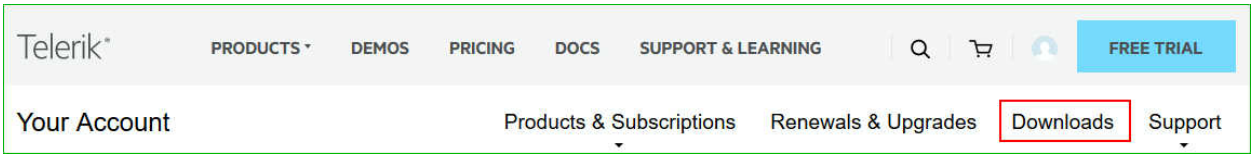
When you have an active trial or developer license, you can download the following files:

- Standalone installation
- Assemblies for manual installation
- NuGet packages
- Old versions

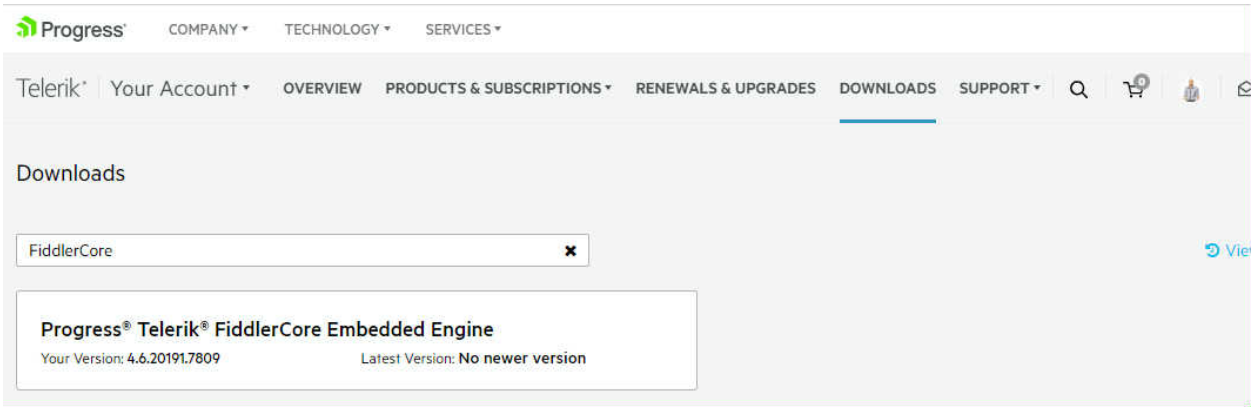
In order to download these you need to take the following steps:

1. Log into your Telerik account.

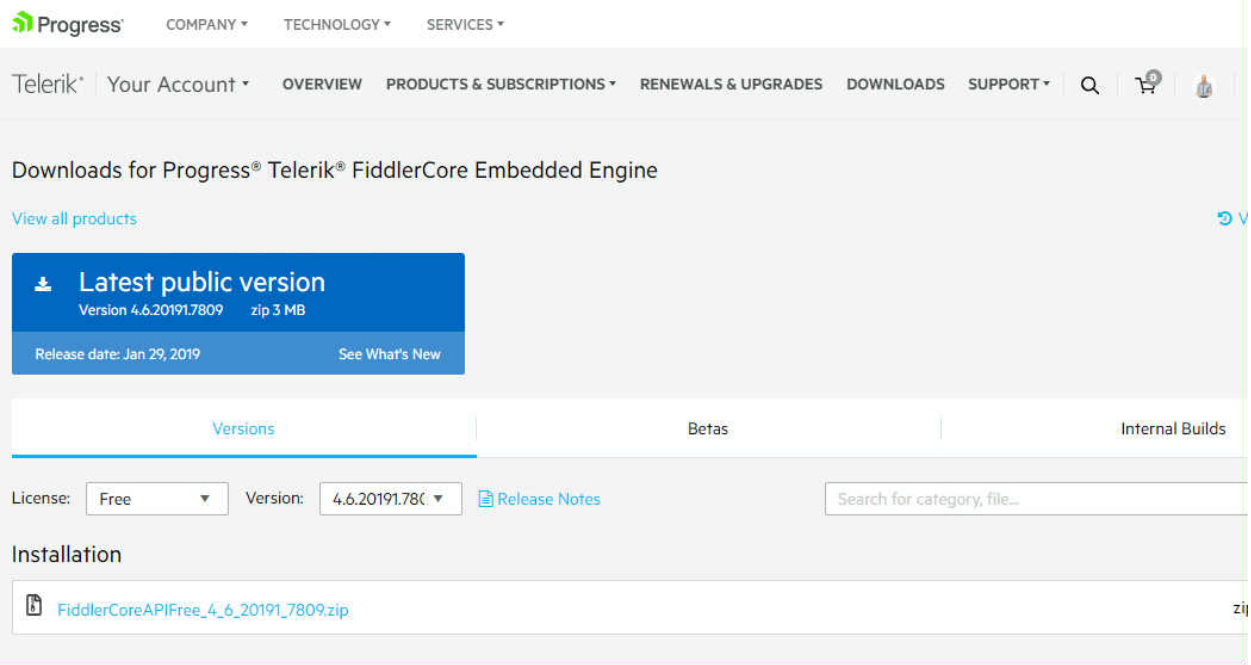
2. Click on the Downloads tab:



3. Select FiddlerCore product title:



4. The next page allows you to download the Automatic Installation msi file, DLLs and NuGet Packages.



Below you could find a list of the available files:

[license] could be Trial or Dev depending on the license you have.

[version] is replaced with the version the file corresponds to.

Installation

- FiddlerCoreEmbeddedEngine[version]_[license].zip - contains the required .

Next Steps

- FiddlerCore Configuration

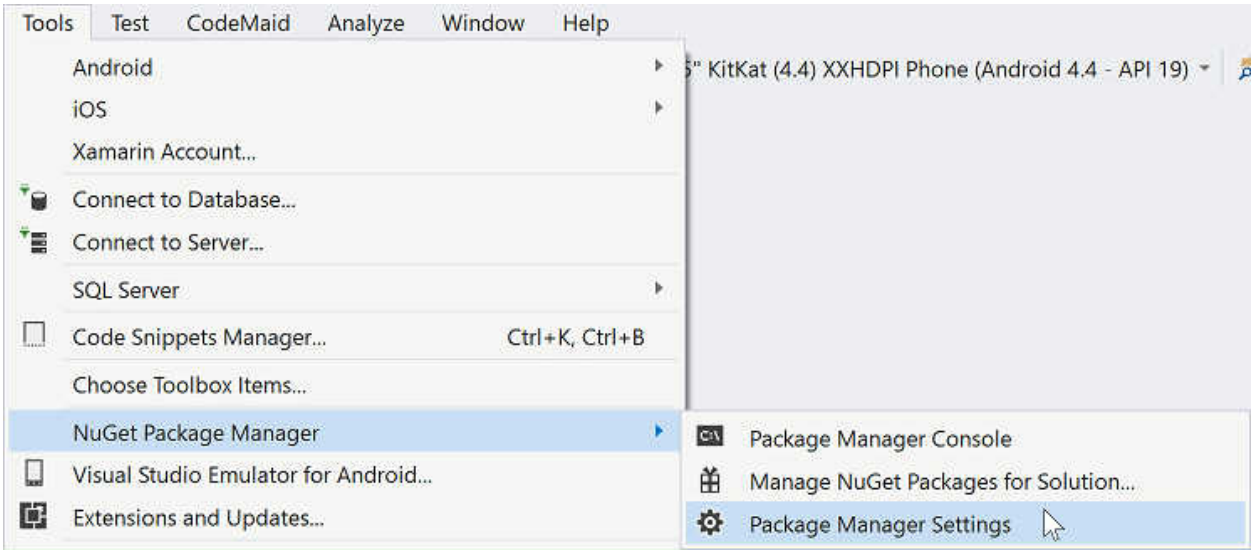
Telerik NuGet Server

The following steps demonstrate how users can take advantage of Telerik NuGet server in order to include our suite in their solution and/or update to the latest available version.

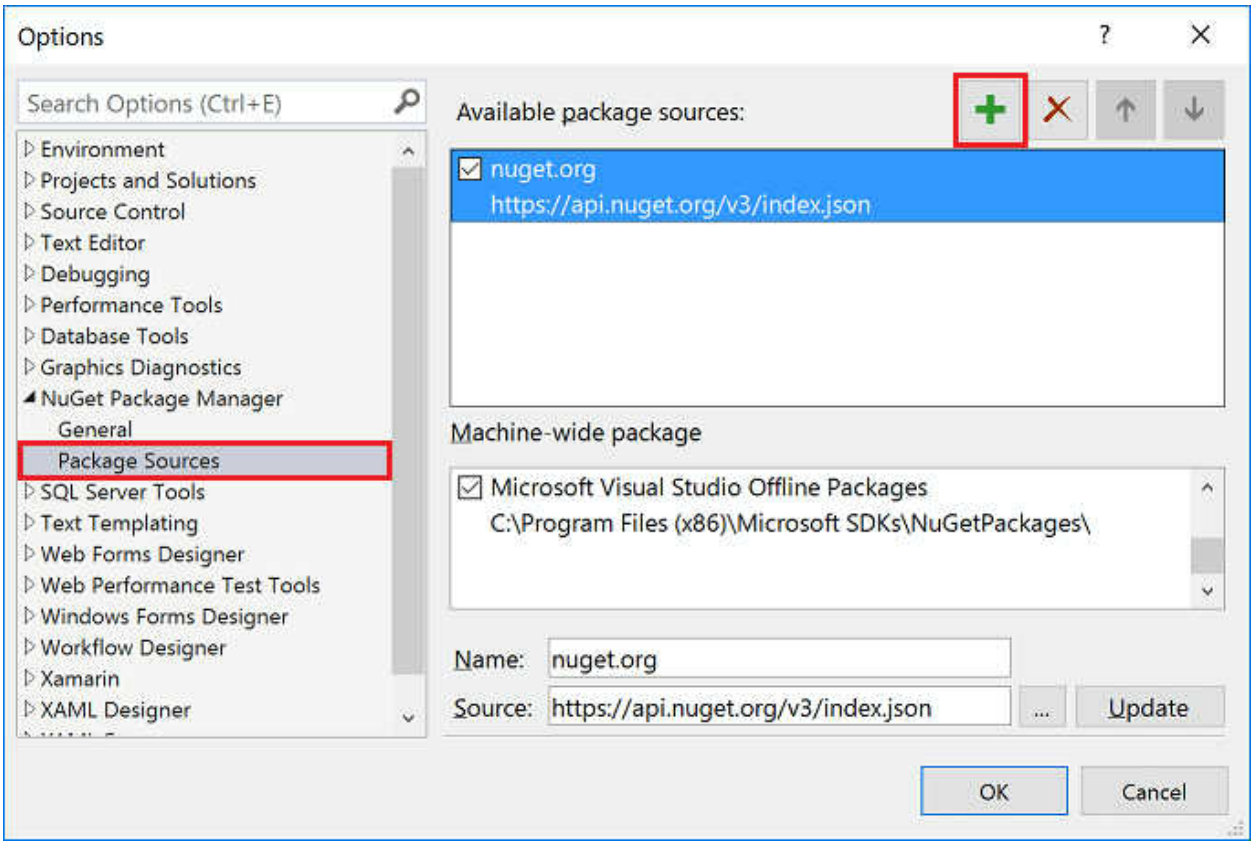
The credentials needed to access Telerik Nuget server are the same you use to log into your Telerik account.

Visual Studio

The first step is to add the Telerik server to the NuGet package sources. This can be done in the Package Manager Settings from the Tools menu.

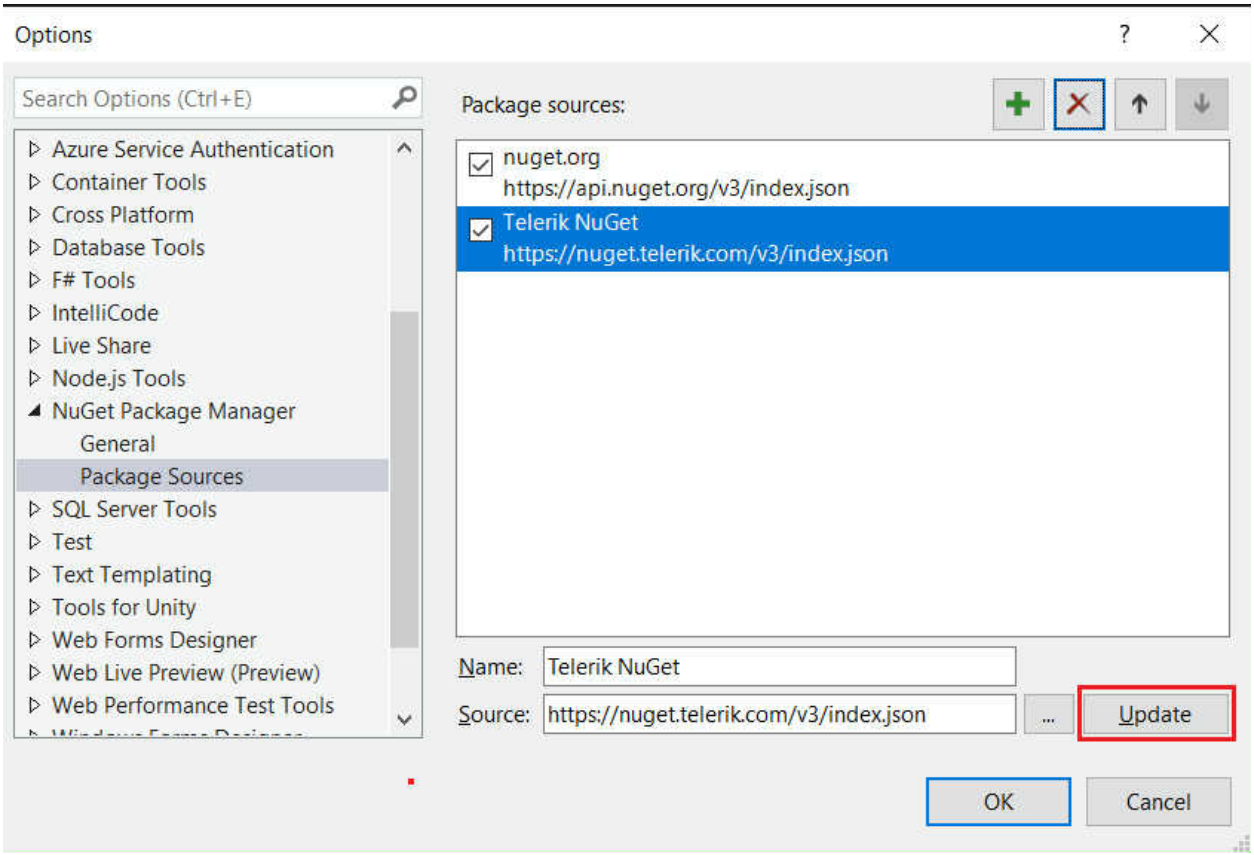


In the the Package Sources section users can add new sources.

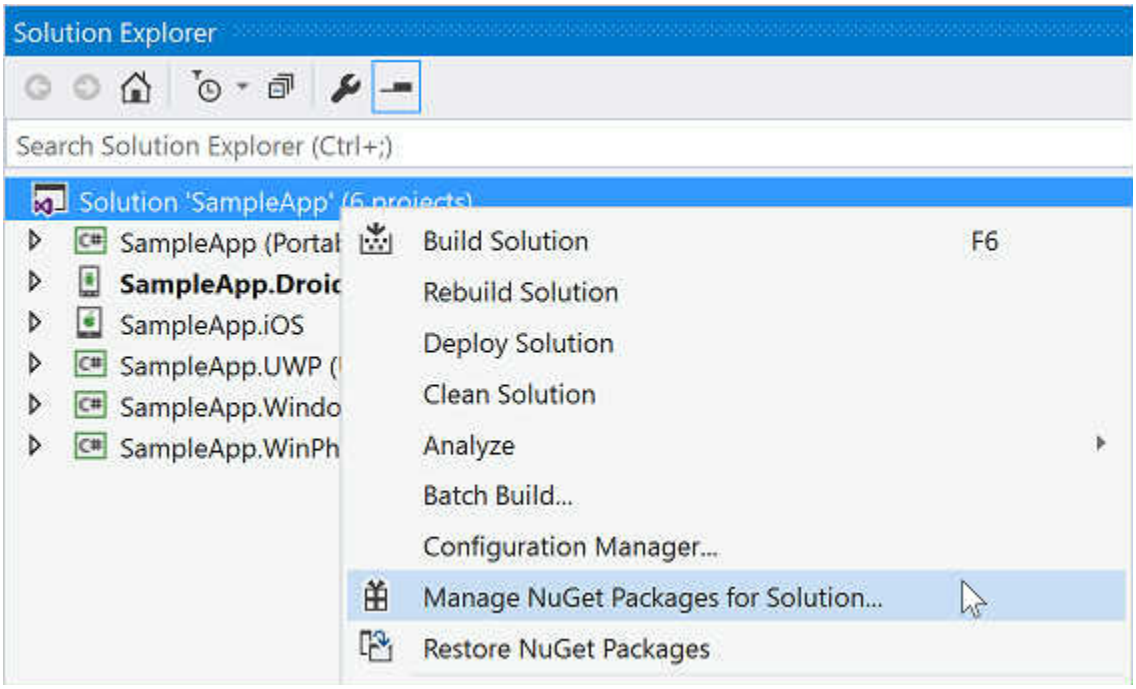


In the Source field users should fill in the address of the Telerik server (URL: https://nuget.telerik.com/v3/index.json) and click the Update button.

The previous version of the Telerik NuGet server (https://nuget.telerik.com/nuget) will be deprecated. Update your settings to point to the new v3 API (URL: https://nuget.telerik.com/v3/index.json), which is faster, lighter, and reduces the number of requests from NuGet clients.

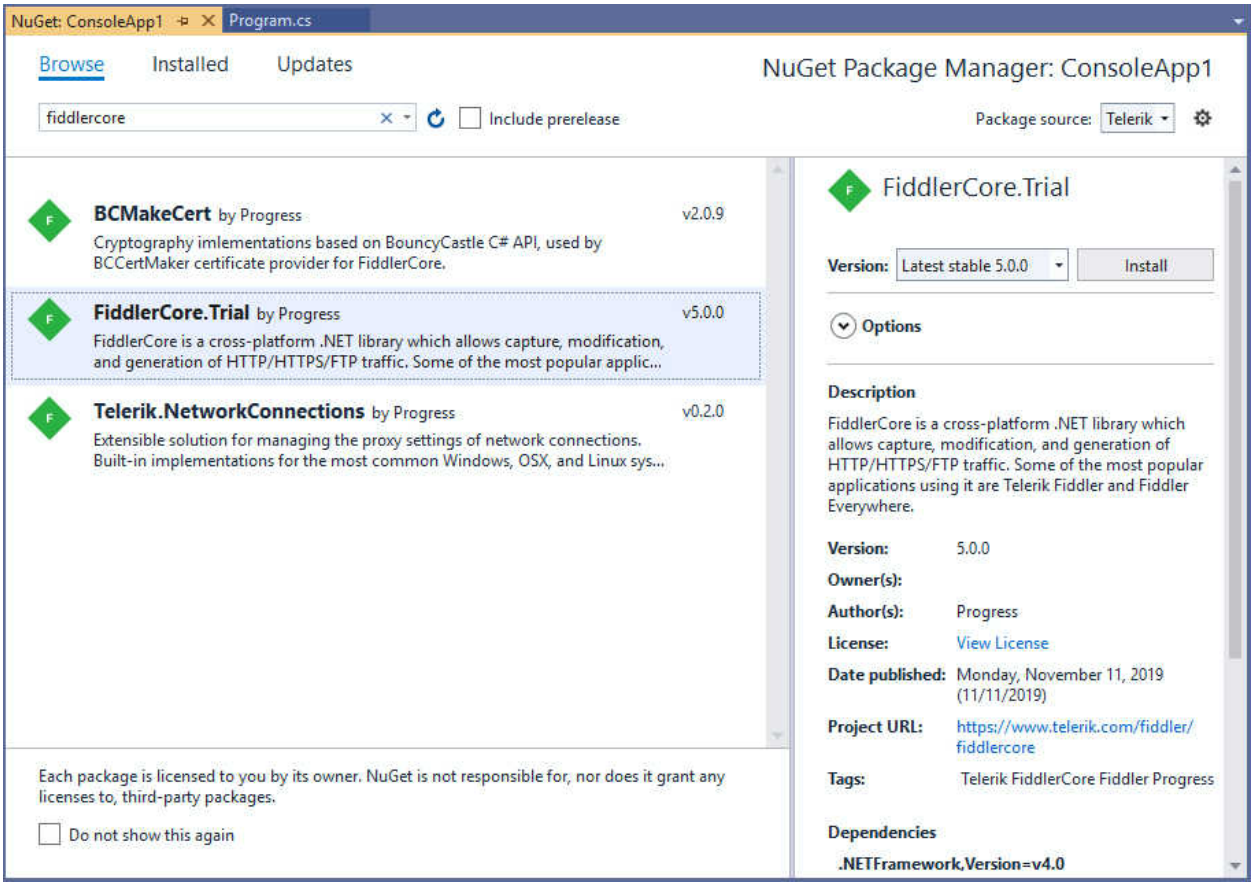


The Telerik server is now ready to use. Users can go to their solution and open the solution package manager.



Users have to find the FiddlerCore package and install it to their projects following these steps:

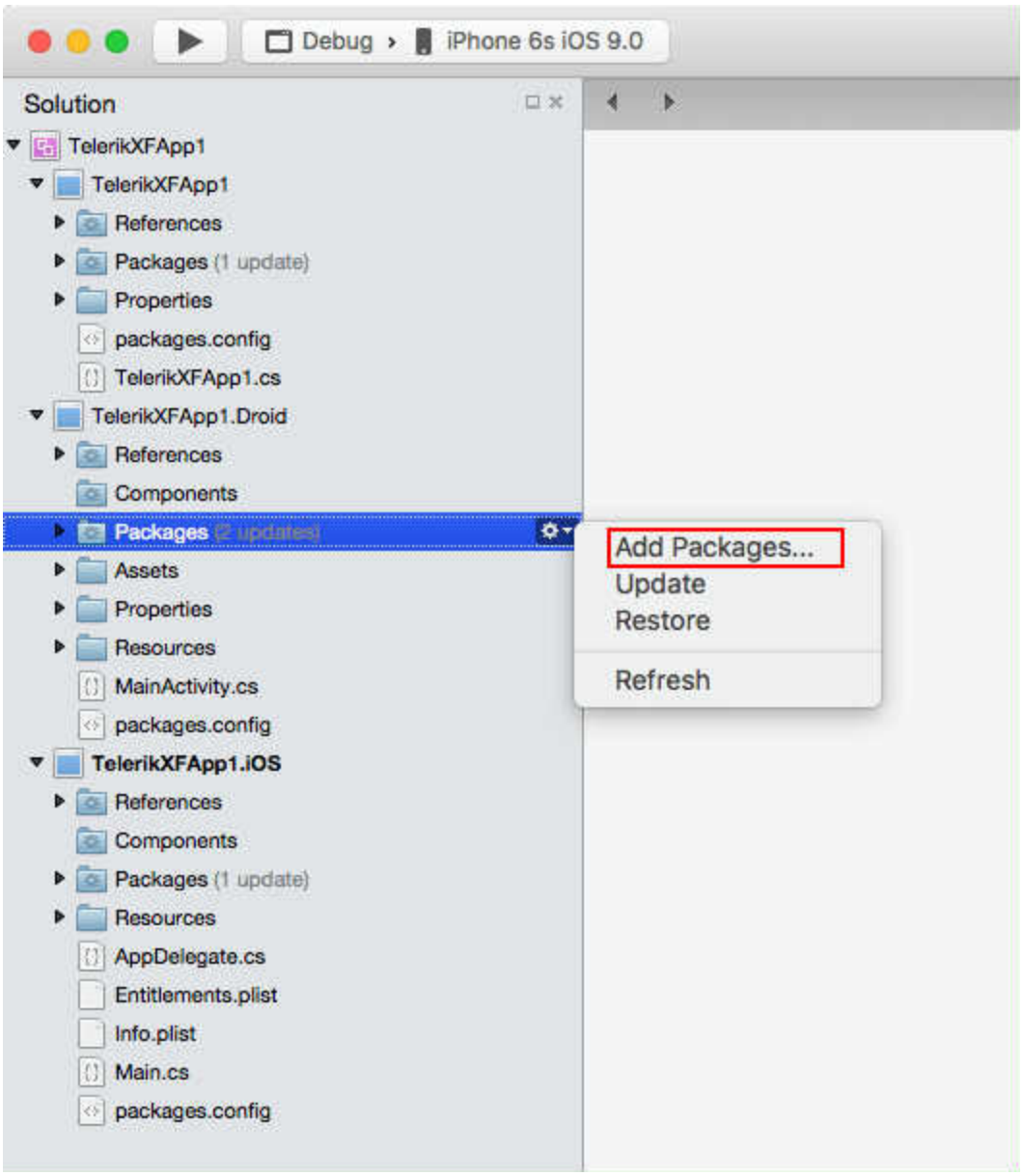
1. Select the Telerik server as a package source and enter their credentials when prompted.
2. Search for the FiddlerCore package.
3. Select the package when found.
4. Select which projects will have the package installed.
5. Choose the desired version and click Install.



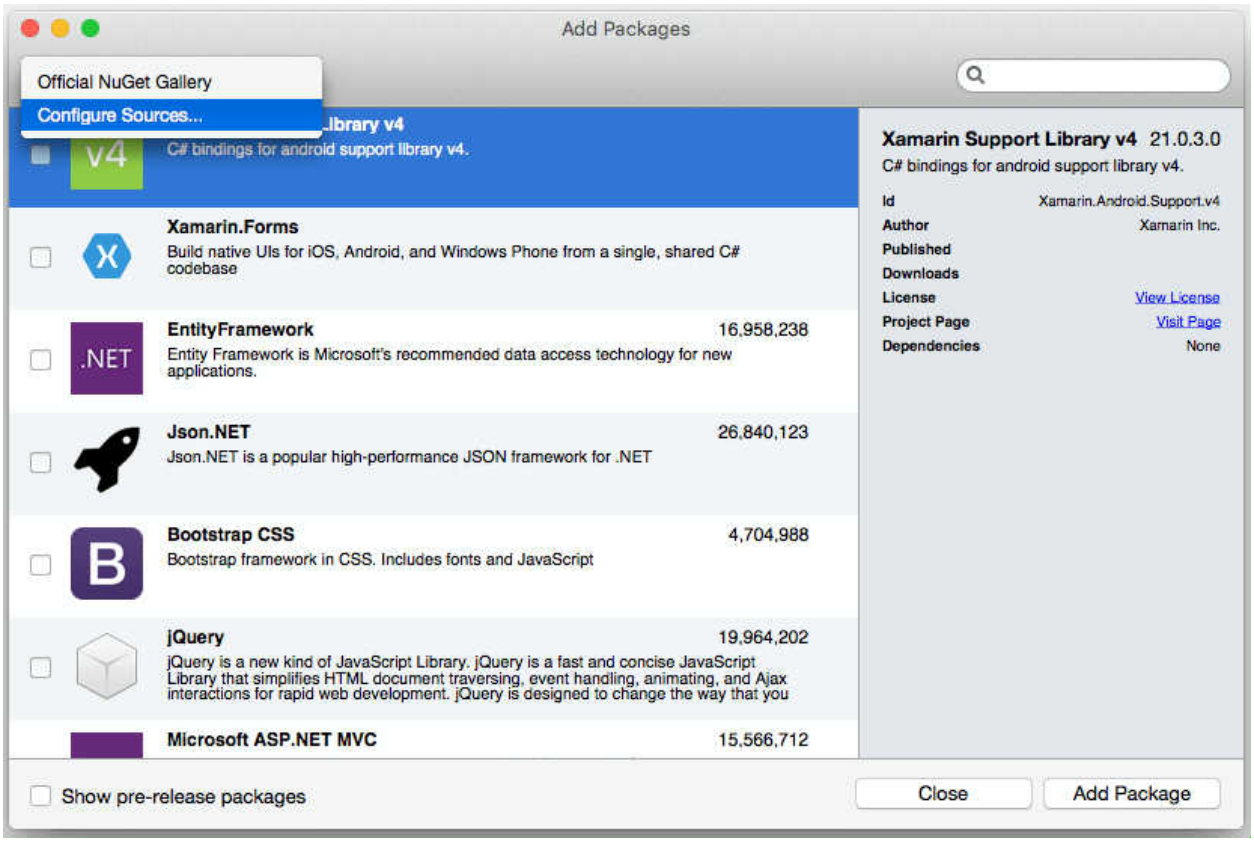
Now the user's solution has all required Telerik assemblies.

Visual Studio for Mac

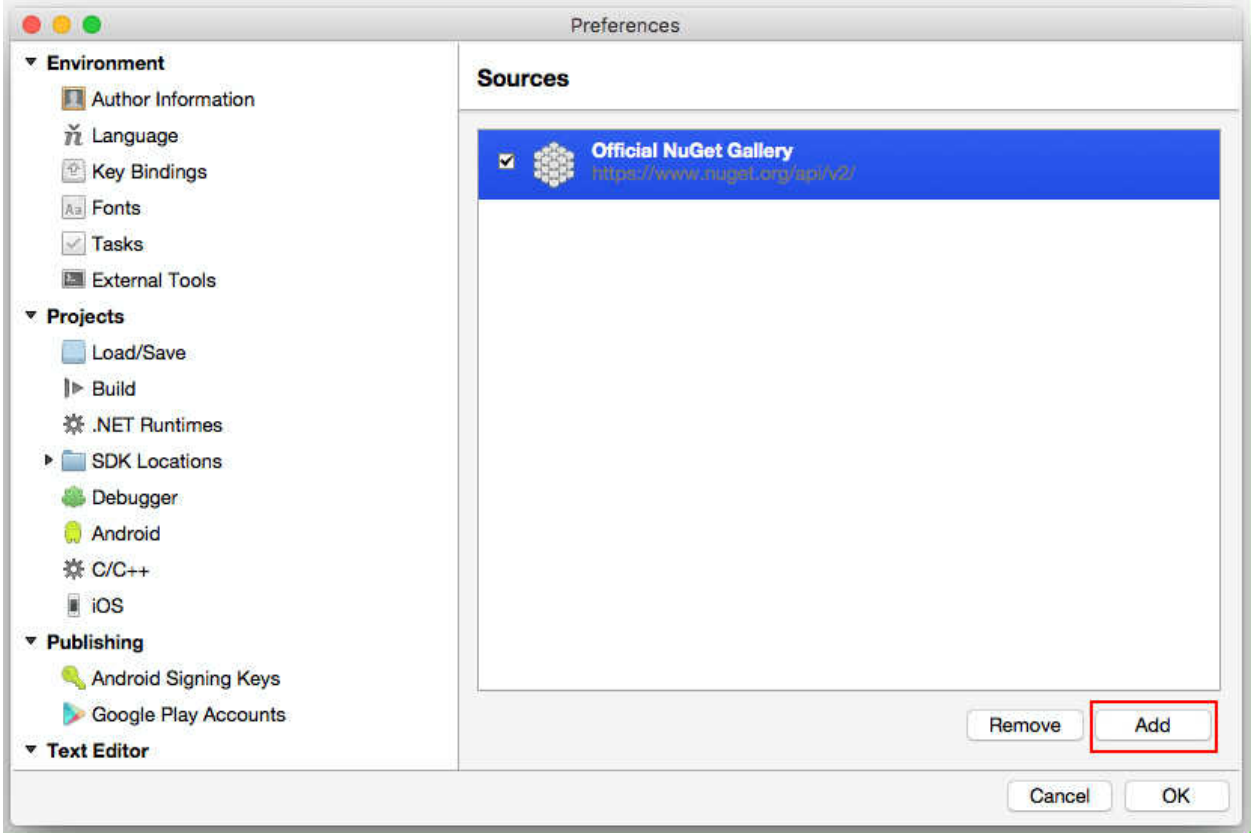
Users first have to add the Telerik NuGet server in their packages sources list. This can be done by clicking on the settings icon of any “Packages” folder (any project will do the job) and choosing “Add Packages…”.



This will open another dialog. Users need to choose “Configure Sources…” option from the dropdown in the upper right corner.

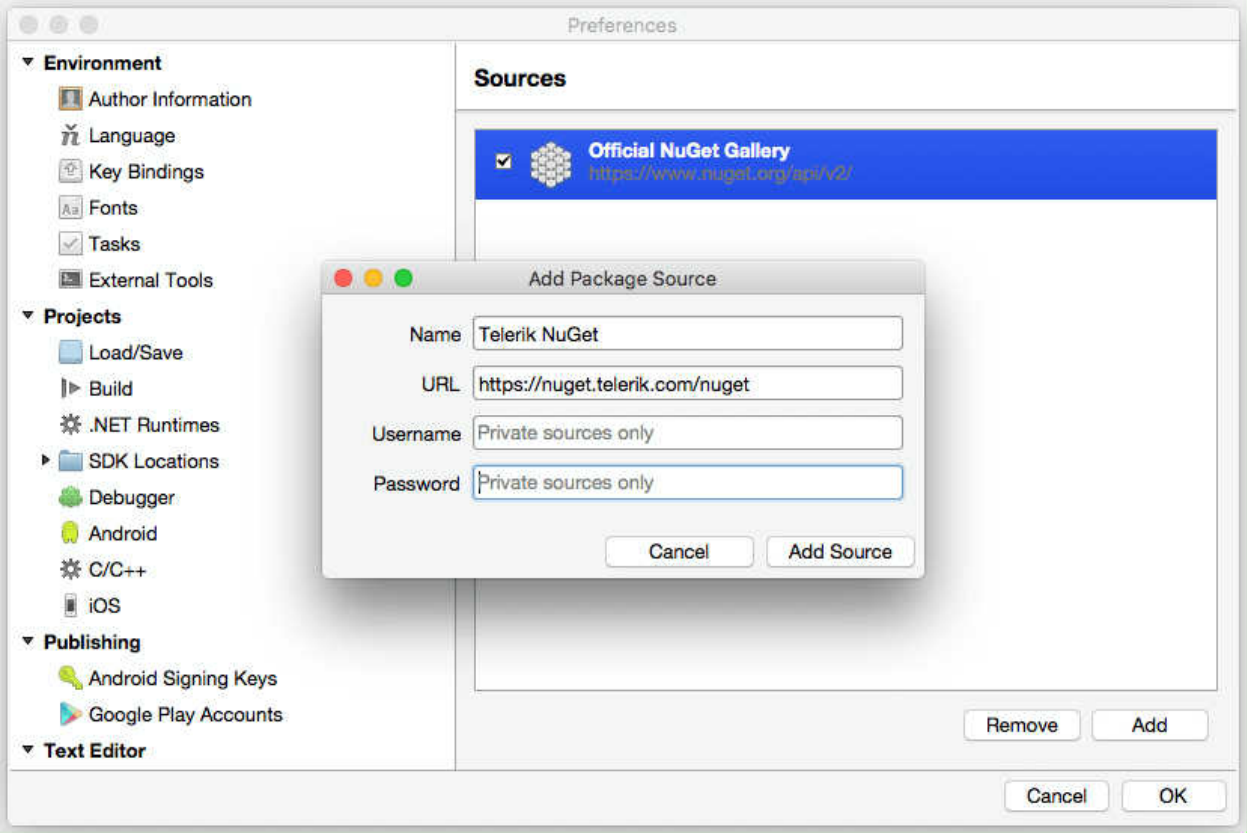


On the next dialog users will see all the available sources. Choose “Add” to add the new server.



In the Add Package Source dialog users should fill in the information of the Telerik server (URL: https://nuget.telerik.com/v3/index.json) as well as their private Telerik credentials. Authentication procedure is required in order to allow downloading the packs.

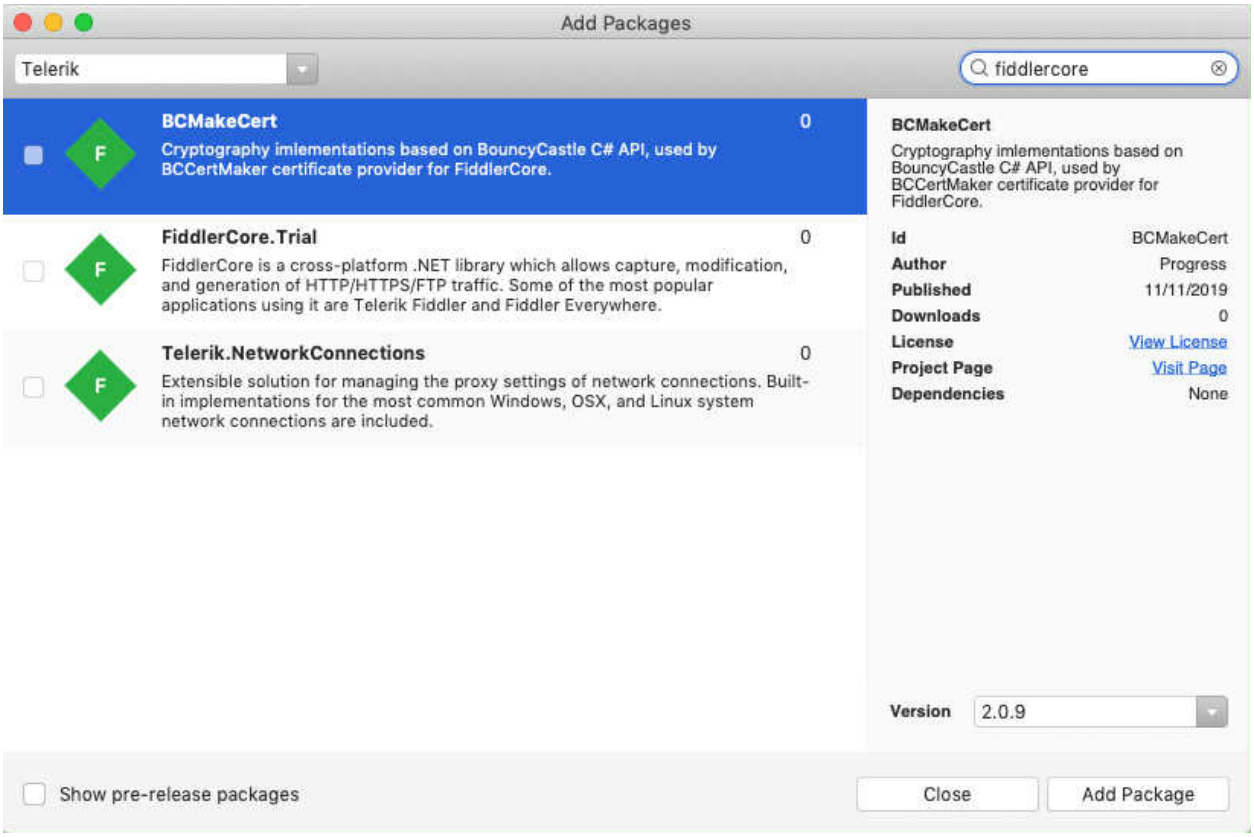
The previous version of the Telerik NuGet server (https://nuget.telerik.com/nuget) will be deprecated. Update your settings to point to the new v3 API (URL: https://nuget.telerik.com/v3/index.json), which is faster, lighter, and reduces the number of requests from NuGet clients.



After the Telerik NuGet server is added users will be able to see the packages they are allowed to download in the Add Packages dialog. This will allow them to check the FiddlerCore pack and add it into their projects.

Add FiddlerCore pack

Once the server is added users will be able to add to their projects any of the Telerik NuGet packages available for their license. One click on the settings icon of the “Packages” folder of any project will open the Add Packages dialog where the available Telerik packs will be listed.



Troubleshooting

'401 Logon failed' error

If you're receiving this error when connecting to Telerik Nuget Server, you could try to update your NuGet credentials through the Windows Credential Manager. Please follow the steps below:

1. Close all open Visual Studio instances (this is so that all NuGet package manager tasks are stopped);
2. Open the "Credential Manager" app on your PC;
3. Scroll through all the entries until you find any that are for nuget.telerik.com;
4. Once you find that entry, expand it and select "edit";
5. Make sure the username and password are the same ones you use for your Telerik account (use the Email in the place of username) and click "Save".

Now you can reopen Visual Studio and access the Telerik NuGet server.

Next Steps

- FiddlerCore Configuration

Configuration

On high-level, working with FiddlerCore is done through the `FiddlerApplication` class and contains 3 steps:

1. Attach to some event handlers, which will be used by FiddlerCore to report information to the application, for example `FiddlerApplication.BeforeRequest` or `FiddlerApplication.AfterSessionComplete`.
2. Start FiddlerCore using `FiddlerApplication.Startup(FiddlerCoreStartupSettings)` method.
3. Shutdown FiddlerCore using `FiddlerApplication.Shutdown()` method.

The simplest usage is demonstrated in the following code snippet:

c#

```
using System;
using Fiddler;

class Program
{
    static void Main(string[] args)
    {
       // Attach to events of interest:
       FiddlerApplication.AfterSessionComplete += session =>
Console.WriteLine(session.fullUrl);

       // Build startup settings:
       var settings = new FiddlerCoreStartupSettingsBuilder()
       .RegisterAsSystemProxy()
       .Build();

       // Start:
       FiddlerApplication.Startup(settings);

       Console.ReadLine();

       // Shutdown:
       FiddlerApplication.Shutdown();
    }
}
```

Startup configuration

To configure FiddlerCore, you can use `FiddlerCoreStartupSettingsBuilder` class, which encapsulates the logic for creating `FiddlerCoreStartupSettting` instance, which in turn can be used as an argument for `FiddlerApplication.Startup(FiddlerCoreStartupSettings)` method.

`FiddlerCoreStartupSettingsBuilder` provides fluent API, for example a convenient usage is:

c#

```
FiddlerCoreStartupSettings startupSettings =
    new FiddlerCoreStartupSettingsBuilder()
       .ListenOnPort(fiddlerCoreListenPort)
       .RegisterAsSystemProxy()
       .Build();

FiddlerApplication.Startup(startupSettings);
```

The following configuration methods of `FiddlerCoreStartupSettingsBuilder` are available:

Common settings:

- `ListenOnPort(int)`: Specifies the port on which FiddlerCore will listen. If 0 is used, random port is assigned.
- `AllowRemoteClients()`: Allows FiddlerCore to accept requests from outside of the current machine, e.g. remote computers and devices.

Be cautious when allowing remote clients to connect to FiddlerCore. If an attacker is able to proxy its traffic through this FiddlerCore instance, it could circumvent IPSec traffic rules and intranet firewalls.

System proxy settings:

There are a lot of possible systems and types of connections which might have to be modified in order to set proper proxy settings, and the following methods handle only the most common scenarios. For more advanced proxy configuration, see Register as System Proxy article.

- `RegisterAsSystemProxy()`: Modifies the local LAN connection's proxy settings to point to the port on which FiddlerCore is listening on localhost.
- `MonitorAllConnections()`: Modifies all system connections' proxy settings to point to the port on which FiddlerCore is listening on localhost.
- `CaptureFTP()`: Modifies the system's proxy FTP-related settings to point to the port on which FiddlerCore is listening on localhost.
- `HookUsingPACFile()`: Modifies current proxy settings to be configured using PAC file. On the other side, FiddlerCore serves a PAC file which is used to modify the connection. The default PAC file which is served can be configured by changing the "fiddler.proxy.pacfile.text" preference, which includes the body of the PAC file's `FindProxyForURL(url, host)` function, for example:

c#

```
FiddlerApplication.Prefs.SetStringPref("fiddler.proxy.pacfile.text", "return 'PROXY
127.0.0.1:8888'");
```

FiddlerCore proxy settings:

- `ChainToUpstreamGateway()`: Sets the current LAN connection's proxy settings as an upstream gateway proxy. For example, if the application is running in a corporate environment behind a corporate proxy, the corporate proxy will be used as an upstream gateway proxy for FiddlerCore.

  Chaining when upstream gateway proxy settings are configured to use PAC file is supported only on Windows.

- `SetUpstreamProxySettingsTo(ProxySettings)`: Sets the FiddlerCore upstream proxy settings as specified in the passed instance for ProxySettings class.

Other settings:

- `DecryptSSL()`: Enables decryption of HTTPS traffic. You should have a CertificateProvider loaded with trusted certificate. For more details see Use Custom Root Certificate article.
- `OptimizeThreadPool()`: Optimizes the thread pool to better handle multiple simultaneous threads. This is often convenient, as each `Session` is handled in a different thread. Under the hood, this methods uses `ThreadPool.SetMinThreads` with a value larger that the default one.

Handling events

`FiddlerApplication` exposes numerous events, described in more details in the Capture HTTP/S Traffic article.

Shutdown

`FidlerCore can be shut down using the following method of`FiddlerApplication`

- `Shutdown()`: Shuts down FiddlerCore, and reverts the proxy settings to the original ones, in case they were modified on startup.

  If there is traffic in progress when calling `FiddlerApplication.Shutdown()`, some of the background threads handling the sessions may throw `ObjectDisposedException` or `NullReferenceException`.

Next Steps

- Register as System Proxy
- Use Custom Root Certificate

Register as System Proxy

This article explains how to register FiddlerCore as system proxy. This is a common scenario, in which part or all of the system traffic is redirected to FiddlerCore, so it can capture and/or modify it – similar to what Fiddler do.

Basic approach

The simplest way to register FiddlerCore as a system proxy is by passing `FiddlerCoreStartupSettings` on startup:

c#

```
FiddlerCoreStartupSettings startupSettings =
    new FiddlerCoreStartupSettingsBuilder()
       .RegisterAsSystemProxy()
       .Build();

FiddlerApplication.Startup(startupSettings);
```

There are more basic methods affecting the system proxy settings in the FiddlerCoreStartupSettings. You can read more in Configuration/Proxy settings article.

Advanced approach

Instead of using the basic configuration methods, you can manually modify the proxy settings. The logic for modifying the system connections' proxy settings is separated in the Telerik.NetworkConnections assembly.

It contains the following key members:

- `INetworkConnectionsDetector`: Base interface representing network connection detector. It contains a single `Detect()` method, which should return a set of `NetworkConnection` instances of a particular type.
- `NetworkConnection`: Base abstract class which allows manipulation and monitoring of proxy settings for a specific network connection. The most important members are:
  - `GetCurrentProxySettings()`: Returns the current `ProxySettings` for the connection.
  - `SetProxySettings(ProxySettings)`: Sets specified proxy settings for the connection.
  - `ProxySettingsChanged`: Event raised when proxy settings for the connection are changed.

To manually manipulate network connections' proxy settings, you can use any of the built-in detectors, obtain an instance of a NetworkConnection class, and invoke it's `SetProxySettings` method, for example:

c#

```
// Detect the network connections:
var networkConnections = new WinINetNetworkConnectionsDetector().Detect();

// Create appropriate proxy settings (in this case bypassing particular hosts):
var proxySettings = new ProxySettings(true, "https://www.telerik.com");

// Modify some of the network connections:
networkConnections.First().SetProxySettings(proxySettings);

// Start:
FiddlerApplication.Startup(new FiddlerCoreStartupSettingsBuilder().Build());
```

The following default implementations for `INetworkConnectionsDetector` are provided:

- `WinINetNetworkConnectionsDetector`: detector for Windows-specific Windows Internet (WinINET) networking component network connection.
- `RasNetworkConnectionsDetector`: detector for Windows-specific RAS network connections.
- `MacNetworkConnectionsDetector`: Detector for Mac-specific network connections.
- `LinuxNetworkConnectionsDetector`: Detector for Linux-specific network connections.

The built-in connection detectors are OS-specific and will throw exception if invoked on not supported platforms.

Next Steps

- Capture HTTP/S Traffic

Capture HTTP/S Traffic with FiddlerCore

This article explains how to capture HTTP/S traffic with FiddlerCore

Once FiddlerCore is configured, it starts to listen for a traffic on the background. When it captures any session, it notifies you by raising the following events:

The following event handlers are invoked on session-handling background threads.

If you need to update a UI thread (e.g. in WPF or Windows Forms application), you may need to delegate the update logic back to the UI thread, for example by using Dispatcher.BeginInvoke (WPF) or Control.Invoke (Windows Forms).

Also, if you use collections that are not thread-safe, like `List<T>`, you may need to employ additional synchronization mechanisms.

FiddlerApplication.BeforeRequest

You should use this event to act when client request is received. For example, you can modify the session flags to use specific ssl protocols.

c#

```
FiddlerApplication.BeforeRequest += session =>
    {
        session["x-OverrideSslProtocols"] = "ssl3;tls1.0;tls1.1;tls1.2";
    };
```

FiddlerApplication.BeforeResponse

You should use this event to act when a server response is received. For example, you can decode the body.

c#

```
FiddlerApplication.BeforeResponse += session =>
    {
        session.utilDecodeResponse();
    };
```

FiddlerApplication.AfterSessionComplete

You should use this event to act when a session is completed. For example, notify the user.

c#

```
FiddlerApplication.AfterSessionComplete += session =>
    {
       Console.WriteLine($"Finished session: {session.fullUrl}");
    }
```

These are only the most commonly used handlers. For the full list of events check FiddlerApplication's API reference.

Next Steps

- Import/export sessions

Use Custom Root Certificate

This article explains how to generate and trust your own root certificate.

By default, when you use the `FiddlerCoreStartupSettingsBuilder.DecryptSSL()` setting, FiddlerCore will create new certificate every time the application runs. This example shows how to change this behavior.

Set the Default Certificate Provider

The following code explains how to set the default certificate provider for FiddlerCore:

c#

```
BCCertMaker.BCCertMaker certProvider = new BCCertMaker.BCCertMaker();
    CertMaker.oCertProvider = certProvider;
```

Create Root Certificate

The following code explains how to create your own root certificate.

c#

```
string rootCertificatePath = @"Path\To\Your\Root\Certificate\RootCertificate.p12";
    string rootCertificatePassword = "S0m3T0pS3cr3tP4ssw0rd";
    if (!File.Exists(rootCertificatePath))
    {
       certProvider.CreateRootCertificate();
       certProvider.WriteRootCertificateAndPrivateKeyToPkcs12File(rootCertificatePath,
rootCertificatePassword);
    }
```

Read Root Certificate from File

The following code explains how to create your own root certificate.

c#

```
string rootCertificatePath = @"Path\To\Your\Root\Certificate\RootCertificate.p12";
    string rootCertificatePassword = "S0m3T0pS3cr3tP4ssw0rd";
    if (File.Exists(rootCertificatePath))
    {

certProvider.ReadRootCertificateAndPrivateKeyFromPkcs12File(rootCertificatePath,
rootCertificatePassword);
    }
```

Trust Root Certificate

The following code explains how to trust your root certificate.

c#

```
if (!CertMaker.rootCertIsTrusted())
    {
       CertMaker.trustRootCert();
    }
```

Next Steps

- Import/export sessions

Import and Export Sessions with FiddlerCore

This article explains how to import and export sessions with FiddlerCore.

Import Sessions

You can import sessions with FiddlerCore by using the following code:

c#

```
Session[] loaded = Utilities.ReadSessionArchive(sazFilename, false, "", (file, part)
=>
    {
       Console.WriteLine($"Enter the password for { part } (or just hit Enter to
cancel):");
       string sResult = Console.ReadLine();
       Console.WriteLine();

       return sResult;
    });
```

Export Sessions

You can export sessions with FiddlerCore by using the following code:

c#

```
bool success = Utilities.WriteSessionArchive(filename, sessions.ToArray(), password,
false);
```

Custom SAZ Provider

There are cases when you may want to use custom provider to save FiddlerCore sessions. You do this by setting the following property:

c#

```
FiddlerApplication.oSAZProvider = new CustomSazProvider();
```

Session Flags

A field in each Session object contains flags that control the processing of the session. The flags can be accessed by oSession.oFlags["flagname"] or by using the default indexer on the Session object: oSession["flagname"].

Using SessionFlags

- Flag names are not case-sensitive.
- Flag values are always strings.
- If you examine oFlags["non-existent-flag"], the result will be null.
- The oFlags collection is the "indexer" for the Session object, so oSession.oFlags["flagname"] can be written as:
  - oSession["flagname"] or
  - oSession["SESSION", "flagname"]
- You can remove a flag from the list by:
  - Calling: oFlags.Remove("flagname") or
  - Setting oSession["flagname"] = null
- The value of most flags is not important; simply adding the flag is enough. So oSession["ui-hide"]="no" does the same thing as oSession["ui-hide"] = "true" (hides the session).
- While you can call oFlags.Add("flagname"), this will throw an exception if the flag already exists. It's better to just set the value: oFlags["flagname"] = "value";
- You can create new flags that attach metadata to a given session. To avoid naming conflicts, it's recommended that you choose distinctive flagnames. For example: addon.acme.loggingFlag.

Host Flags

- x-overrideHost - Provide the Host:Port combination, which should be used for DNS resolution purposes. Note that this mechanism does not change the HOST header on the request and thus is not useful if there's an upstream gateway.
- x-hostIP - Indicates the IP address of the server used for this request. Read-only flag.
- x-overrideGateway - Provide the Host:Port combination of a gateway that should be used to proxy this request, or DIRECT to send the request directly to the origin server.

Client Flags

- x-ProcessInfo - Information (module name and ProcessID) on source of local requests.
- x-clientIP - Indicates the client IP that sent this request. It is most useful when multiple computers on a network are pointed to a single Fiddler instance. Read-only flag.
- x-clientport - Indicates the port on the client that sent this request. Read-only flag.
- x-ConnectResponseRemoveConnectionClose - Use during the FiddlerApplication.RequestHeadersAvailable or FiddlerApplication.BeforeRequest event, so that when FiddlerCore responds to the client's CONNECT request with "200 Connection established", it will not add the "Connection: close" header. This header is known to be

controversial for the CONNECT response, and it causes problems with some clients.

Socket Reuse Flags

- x-serversocket - String containing data about the reuse status of the server socket. Read-only flag.
- x-securepipe - String containing data about the reuse status of a secure server socket. Read-only flag.

Decryption and Authentication Flags

- x-no-decrypt - If set on a CONNECT tunnel, the traffic in the tunnel will not be decrypted.
- https-Client-Certificate - Filename of client certificate (e.g. .CER) that should be attached to this secure request.
- x-OverrideCertCN - String specifying the hostname that should appear in the CN field of this CONNECT tunnel's Fiddler-generated certificate.
- x-SuppressProxySupportHeader - Prevent FiddlerCore from adding a "Proxy-Support: Session-Based-Authentication" header to HTTP/401 or HTTP/407 responses that request Negotiate or NTLM authentication.

Performance Flags

- request-trickle delay - Milliseconds to delay each outbound kilobyte of request data.
- response-trickle-delay - Milliseconds to delay each inbound kilobyte of response data.

Drop Sessions Flags

- log-drop-request-body - Drop the request body from the session list after a request is sent to the server. Helpful in reducing memory usage.
- log-drop-response-body - Drop the request body from the session list after a response is sent to the client. Helpful in reducing memory usage.

Copyright

License Agreement

End User License Agreement for Progress Telerik FiddlerCore can be found on the following pages:

- https://www.telerik.com/purchase/license-agreement/fiddlercore for the internal version of FiddlerCore, including Trial License.

Personal Data Collection

How personal data is used by Telerik Products

Telerik FiddlerCore uses your account details, email address and a product usage data to better understand customer experience allowing us to focus development efforts on the products and features that matter most to our customers. Additionally, this information may be used by Marketing and / or Sales to ensure you receive relevant content updates, product news and materials to ensure your success with our products. Please see our Privacy Policy for more details.

This information is for our own purposes and do not sell or otherwise provide this information to any third-parties for a commercial purpose.

How I can see what data about me is stored?

You can see the information stored for your account by sending request to us via the following form GDPR Data Subject Access Rights Request

How I can delete the data stored about me?

You can request deletion of the information stored for your account by sending request to us via the following form GDPR Data Subject Access Rights Request

Release History

v5.0.2 FiddlerCore

Improvments

- Add digital signature to the FiddlerCore NuGet package and the FiddlerCore assemblies.

v5.0.1 FiddlerCore

Features

- Add x-ConnectResponseRemoveConnectionClose session flag (documented in the Client flags section).

v5.0.0 FiddlerCore

Fixed bugs

- SSL handshake fails for some servers with TLS1.2

Improvments

- Removal of makecert.exe from FiddlerCore distributions
- Improved the NetworkConnections API
- Included PDBs for the NetworkConnections assemblies in the distributions
- Hook using PAC script only
- Updated FiddlerCore EULA
- Updated FiddlerCore demo project