Integrate the Splunk platform using development tools for .NET

You can integrate the Splunk platform with your applications using C#. Use the following tools and resources to help you optimize the solutions you build for Splunk Enterprise.

Splunk Enterprise SDK for C#

The Splunk Enterprise SDK for C# functions as a layer on top of the Splunk REST API and helps you to optimize your productivity while working with Splunk software. You can use the Splunk Enterprise SDK for C# to build .NET applications that can communicate with Splunk Enterprise instances, retrieve and manipulate data, and extend the capabilities of the Splunk platform.

The Splunk Enterprise SDK for C# includes the following resources:

- The Splunk Enterprise .NET library
- C# code examples
- Unit tests
- Documentation, including the C# API Reference

For more information about the Splunk Enterprise SDK for C#, see Splunk Enterprise SDK for C#.

Splunk logging for .NET

Splunk Enterprise SDK for C#

The Splunk Enterprise SDK for C# has been completely rewritten to take advantage of modern .NET technologies, and is asynchronous and portable to different platforms. The SDK includes the Splunk .NET library, C# code samples, and unit tests. It makes it easy for developers to build .NET applications that can communicate with Splunk Enterprise instances, retrieve and manipulate Splunk data, and extend Splunk's capabilities.

The Splunk Enterprise SDK for C# documentation provides information about how to program with the Splunk Enterprise SDK for C#, and includes the Splunk Enterprise SDK for C# API Reference, which provides detailed technical information about each namespace and class that comprises the SDK.

To help you quickly find the information you need to create applications with the Splunk Enterprise SDK for C#, the documentation is divided into several sections:

- **Overview:** Provides information for programmers migrating from the Splunk Enterprise SDK for C# v1.0.x, as well as a basic overview of the SDK's contents, structure, and uses.
- **Get started:** Describes how to obtain, install, and configure the Splunk Enterprise SDK for C#.
- **How do I...?:** Provides introductory information for programming with the Splunk Enterprise SDK for C#—for example, creating a **Service** object, logging into Splunk Enterprise, working with reports (saved searches in Splunk Enterprise 5), working with search jobs, displaying search results, and using modular inputs.
- **Examples:** Provides a set of examples of using the Splunk Enterprise SDK for C# to create applications that access Splunk Enterprise.
- **Troubleshooting:** Helps you troubleshoot common SDK issues.
- **C# API Reference:** Describes every API contained within the Splunk Enterprise SDK for C#. Clicking this link will open a new tab or window with the API reference documentation, which lives on the main Splunk documentation site.

Overview of the Splunk Enterprise SDK for C#

The Splunk Enterprise SDK for C# features APIs that meet C# developer community expectations, are asynchronous, follow .NET guidelines, and abide by FxCop and StyleCop rules. The API client in the Splunk Enterprise SDK for C# is a Portable Class Library (PCL), and supports development targeting the .NET Framework 4.5, Windows Store applications, and Windows Phone 8.1 applications. Using third-party development tools, you can additionally develop for non-Windows platforms such as iOS, Android, Linux, and OS X.

This overview section tells you more about:

- What's new in the Splunk Enterprise SDK for C#
- Migrating from Splunk Enterprise SDK for C# v1.0
- About the Splunk Enterprise SDK for C#
- Asynchronous programming with async/await
- Reactive Extensions (Rx)

Migrating from Splunk Enterprise SDK for C# v1.0.x

The Splunk Enterprise SDK for C# version 2.x is a rewrite of the Splunk Enterprise SDK for C# and introduces completely new APIs.

Important: Applications built with Splunk Enterprise SDK for C# version 1.0.x will not recompile using Splunk Enterprise SDK for C# version 2.x.

Splunk Enterprise SDK for C# version 2.x includes a subset of the capability in version 1.0.x of the SDK, and focuses on the most common scenarios. The major focus areas are search, search jobs, configuration, and modular inputs. For more detailed information about supported functionality, see What you can do with the Splunk Enterprise SDK for C#.

Among the differences between the Splunk Enterprise SDK for C# versions 2.x and 1.0.x are the following:

- Splunk Enterprise SDK for C# v1.0.x is based on the Splunk Enterprise SDK for Java, and features APIs that are written according to Java API naming conventions. Version 2.x of the Splunk Enterprise SDK for C# has brand-new APIs with names and signatures that meet C# developer community expectations.
- Splunk Enterprise SDK for C# v2.x is asynchronous, returns a task, and supports new **async/await** features; while version 1.0.x of the Splunk Enterprise SDK for C# is synchronous.
- Splunk Enterprise SDK for C# v2.x supports the **async** keyword, which requires, at minimum, the .NET Framework 4.0.
- All Splunk Enterprise SDK for C# v2.x APIs follow .NET guidelines and abide by FxCop and StyleCop rules.
- The Splunk API client in Splunk Enterprise SDK for C# v2.x is a Portable Class Library (PCL), and supports cross-platform development. (For more information about supported platforms, see Check prerequisites.)

There are some important differences between the two versions' APIs and how you complete some common tasks:

- The **Service.Connect** method is no longer available. To create a new instance of the **Service** class, use one of its constructors. See the following comparison of code written with the two SDK versions for an example. This code retrieves a list of all Splunk Enterprise apps

```
        8089,
        new Namespace(user: "nobody", app: "search")))
    {
        await service.LogOnAsync("admin", "yourpassword");
        Console.WriteLine("List of Apps:");

...
```

SHOW MORE

- Create and run search reports (also known as saved searches) using asynchronous methods. Instead of using the **Create()**, **Get()**, and **Delete()** methods, use the asynchronous methods **CreateAsync(...)**, **GetOrNullAsync()**, and **RemoveAsync()**. For example, take a look at the following comparative example, which illustrates how to create and dispatch a search report (saved search) using both versions of the SDK:

  Splunk Enterprise SDK for C# 1.0.x

```
var service = Service.Connect(new Dictionary <string, object="">{
                {"hostname", "localhost"},
                {"port", 8089},
                {"scheme", "https"},
                {"username", "admin"},
                {"password", "yourpassword"}
            });

...
```

SHOW MORE

  Splunk Enterprise SDK for C# 2.x

```
using (var service = new Service(
    Scheme.Https, "localhost",
    8089,
    new Namespace(user: "nobody", app: "search")))
```

```
        var job = service.GetJobs().Create(

...
```

SHOW MORE

Splunk Enterprise SDK for C# 2.x

```
using (var service = new Service(
    Scheme.Https, "localhost",
    8089,
    new Namespace(user: "nobody", app: "search")))
{
    await service.LogOnAsync("admin", "yourpassword");
    Job job = await service.Jobs.CreateAsync("search index=_internal | head 10");

...
```

SHOW MORE

- To run a search in real-time and print preview results at regular intervals, you can now use the **Task.Delay** method to delay the thread asynchronously instead of sleeping it. For example:

  Splunk Enterprise SDK for C# 1.0.x

```
var service = Service.Connect(new Dictionary <string, object="">{
                {"hostname", "localhost"},
                {"port", 8089},
                {"scheme", "https"},
                {"username", "admin"},
                {"password", "yourpassword"}
            });

...
```

SHOW MORE

About the Splunk Enterprise SDK for C#

The Splunk API exposes Splunk resources via a REST API using the HTTP protocol. Any tool or language that supports HTTP can use the API to configure and manage a Splunk Enterprise instance and issue search commands.

The Splunk Enterprise SDK for C# is a Splunk-developed collection of C# APIs that uses the Splunk REST API to configure, manage, and issue search commands to your Splunk Enterprise instance. Using the Splunk Enterprise SDK for C#, you can develop your own Splunk application or integrate Splunk functionality into your existing app.

What you can do with the Splunk Enterprise SDK for C#

This SDK contains library code and examples designed to enable developers to build applications using Splunk Enterprise. With the Splunk Enterprise SDK for C# you can write C# applications to programmatically interact with the Splunk Enterprise engine. The SDK is built on top of the REST API, providing a wrapper over the REST API endpoints. So with fewer lines of code, you can write applications that can interact with Splunk Enterprise in the following ways:

- Login
- Access control (users and passwords)
- Searches (normal, blocking, real-time, one-shot, and export)
- Jobs
- Reports (*saved searches* in Splunk Enterprise 5)
- Configuration and Config Properties

```
    await service.LoginAsync("admin", "yourpassword");
```

Once the **Service** instance is created and you're logged in, you can use it to navigate, enumerate, and operate on a wide variety of Splunk resources.

Entities and collections

The Splunk REST API consists of over 160 endpoints that provide access to almost every feature of Splunk. The majority of the Splunk Enterprise SDK for C# API follows a convention of exposing resources as collections of entities, where an entity is a resource that has properties, actions, and metadata that describes the entity. The entity/collection pattern provides a consistent approach to interacting with resources and collections of resources.

For example, the following code lists all apps installed on the Splunk Enterprise server:

```
foreach (var app in service.Applications)
{
    Console.WriteLine(app.Name);
    // write a separator between the name and the description of an app
    Console.WriteLine(Enumerable.Repeat<char>('-', app.Name.Length).ToArray());
    Console.WriteLine(app.Description);
}
```

Collections use a common mechanism to create and remove entities. Entities use a common mechanism to retrieve and update property values, and access entity metadata. Once you're familiar with this pattern, you'll have a reasonable understanding of how the SDK and underlying REST API work.

The SDK contains the base classes **Entity** and **EntityCollection**, both of which derive from the common base class Resource. Note that **Service** is not a **Resource**, but is a container that provides access to all features associated with a Splunk instance.

```
Service
Resource
    Entity
    EntityCollection
```

Asynchronous programming with the Splunk Enterprise SDK for C#

You've probably already noticed the use of the **await** expression in our examples. The APIs in the Splunk Enterprise SDK for C# that involve HTTP calls are fully asynchronous. You can identify async methods by the "Async" suffix—for example, **Service.GetCapabilitiesAsync**.

You designate suspension points using the **await** expression. When you designate a suspension point with **await**, you're telling the compiler that the async method can't continue past that point until the awaited task is complete. The async method is removed from the scheduling queue until the task it is awaiting completes. For more information about asynchronous programming in C#, see Asynchronous Programming with Async and Await on the Microsoft website.

Using Reactive Extensions (Rx) with the Splunk Enterprise SDK for C#

With the Splunk Enterprise SDK for C#, you can use Reactive Extensions for .NET (Rx .NET) to "observe" Splunk's query results and receive notifications from that collection. Rx was built for pushing multiple data streams that you can subscribe to using an observer interface. The Splunk Enterprise SDK for C# defines an **Observable<T>** class to provide basic push-based notifications to observers. This implementation of **Observable<T>** is available without first installing Rx. However, you can implement either the SDK's **Observable<T>** or an observer defined in Rx.

Compare these two search examples that based on the **normal-search** example in the SDK. The first is a "pull" model that uses a **foreach** loop to iterate through the search results, while the second is a "push" model that uses a subscription to observe the collection of search result records:

```
//// Search : Pull model (foreach loop => IEnumerable)

Job job = await service.Jobs.CreateAsync("search index=_internal | head 10");
SearchResultStream stream;

using (stream = await job.GetSearchResultsAsync())
{
    try
    {
        foreach (SearchResult result in stream)
        {
            Console.WriteLine(string.Format("{0:D8}: {1}", stream.ReadCount, result));
        }

        Console.WriteLine("End of search results");
    }
    catch (Exception e)
    {
        Console.WriteLine(string.Format("SearchResults error: {0}", e.Message));
    }
}

//// Search : Push model (by way of subscription to search result records => IObservable)
```

Get started with the Splunk Enterprise SDK for C#

Get started with the Splunk Enterprise SDK for C#:

- Get Splunk Enterprise
- Check prerequisites
- Install the SDK
- Build the SDK and examples

Get Splunk Enterprise for the Splunk Enterprise SDK for C#

You need Splunk® Enterprise 5 or later to use the Splunk Enterprise SDK for C# version. If you haven't already, download Splunk Enterprise: https://www.splunk.com/download.

You can run Splunk Enterprise anywhere on your network and on any platform, as long as you can access your Splunk Enterprise instance from the computer on which you've installed the Splunk Enterprise SDK for C#. You do not need to run Splunk Enterprise on the same computer you'll be using to develop with the SDK, and your Splunk Enterprise instance can be running on any platform —Windows, Linux, OS X, and so on.

From here on out, we're assuming you know a little about using Splunk Enterprise already, have some data indexed, and maybe saved a search or two. But if you're not there yet and need some more Splunk Enterprise education, we have you covered:

- If you want a deeper description of Splunk Enterprise's features, see the Splunk Enterprise documentation.
- Try the Tutorial in the Splunk Enterprise documentation for a step-by-step walkthrough of using Splunk Enterprise with some sample data.
- The Splunk SDKs are built as a layer over the Splunk Enterprise REST API. While you don't need to know the REST API to use this SDK, you might find it useful to browse the Splunk Enterprise REST API Reference.

Requirements for Splunk Enterprise SDK for C#

Beyond Splunk Enterprise, you'll need a few more things before you can install and use the Splunk Enterprise SDK for C#.

Be aware that these requirements only apply to the computer on which you'll be installing the Splunk Enterprise SDK for C#. Your Splunk Enterprise instance can be separate, and can be running on any Splunk-supported platform.

Install a framework

At the very least, you will need one of two frameworks:

- Microsoft .NET Framework 4.5 or later, running on Windows Vista SP2 or later. This enables native development for Windows.
- Mono, which is installed when you install Xamarin Studio.

Install an IDE

Using an integrated development environment (IDE) is not required, but is recommended. Splunk has tested the following IDEs:

- Microsoft Visual Studio 2012 and later on Windows Vista SP2 or later

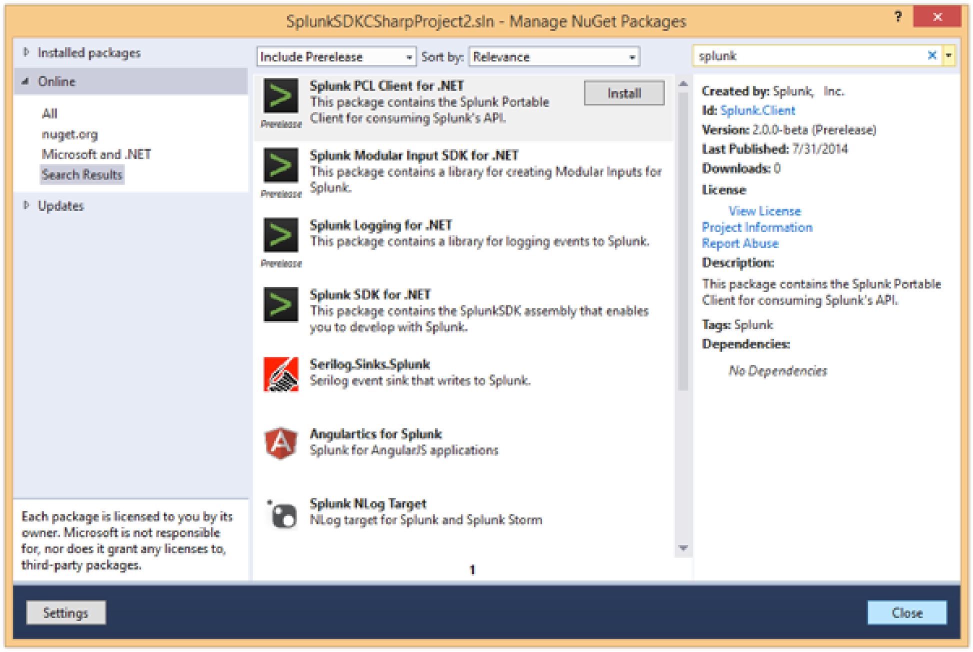
To take advantage of support for Reactive Extensions for .NET in the Splunk Enterprise SDK for C#, download and install them using the instructions on the Get Reactive Extensions for .NET page on MSDN.

Install the Splunk Enterprise SDK for C#

There are several options for installing the Splunk Enterprise SDK for C#:

- Visual Studio:
  - Install via NuGet in Visual Studio by searching available packages.
  - Install via NuGet in Visual Studio using the Package Manager Console.
  - Download a ZIP file and add it to your Visual Studio project.
  - Download a ZIP file from our GitHub repo and add it to your Visual Studio project.
- Xamarin:
  - Install via NuGet in Xamarin Studio by searching available packages.
  - Download a ZIP file and add it to your Xamarin Studio project.
  - Download a ZIP file from our GitHub repo and add it to your Visual Studio project.

Click **Online** from the list on the left, and then enter **splunk** into the **Search Online** field in the upper-right corner. The Splunk PCL Client for .NET package appears in the list, as shown here:



SplunkSDKCSharpProject2.sln - Manage NuGet Packages

Installed packages
Online
All
nuget.org
Microsoft and .NET
Search Results
Updates

Include Prerelease | Sort by: Relevance | splunk

Splunk PCL Client for .NET (Prerelease) — Install
This package contains the Splunk Portable Client for consuming Splunk's API.

Splunk Modular Input SDK for .NET (Prerelease)
This package contains a library for creating Modular Inputs for Splunk.

Splunk Logging for .NET (Prerelease)
This package contains a library for logging events to Splunk.

Splunk SDK for .NET
This package contains the SplunkSDK assembly that enables you to develop with Splunk.

Serilog.Sinks.Splunk
Serilog event sink that writes to Splunk.

Angulartics for Splunk
Splunk for AngularJS applications

Splunk NLog Target
NLog target for Splunk and Splunk Storm

1

Created by: Splunk, Inc.
Id: Splunk.Client
Version: 2.0.0-beta (Prerelease)
Last Published: 7/31/2014
Downloads: 0
License
View License
Project Information
Report Abuse
Description:
This package contains the Splunk Portable Client for consuming Splunk's API.
Tags: Splunk
Dependencies:
No Dependencies

Each package is licensed to you by its owner. Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.

Settings | Close

Important: Ensure that the Id of the package you choose is **Splunk.Client**. If the **Id** field in the column on the right says **SplunkSDK**, you have chosen the wrong package. Choose the package with the correct **Id**.

1. Click the **Install** button for the Splunk PCL Client for .NET package.

2. In the Select Projects window, select the checkboxes next to the projects in which you want to install the package, and then click **OK**.

The Package Manager adds the Splunk Enterprise SDK for C# and its dependencies to your project.

Important: If you also want modular input support, repeat this process for the **Splunk Modular Input SDK for .NET**.

Tip: To browse the Splunk Enterprise SDK for C# APIs, right-click **Splunk.Client** or **Splunk.ModularInputs** in the Solution Explorer, and then click **View in Object Browser**. Click the triangle next to **Splunk.Client** or **Splunk.ModularInputs** to view its namespaces.

and then click **View in Object Browser**. Click the triangle next to **Splunk.Client** or **Splunk.ModularInputs** to view its namespaces. Click the triangle next to each namespace to view its classes. For API reference documentation, see Splunk Enterprise SDK for C# Reference.

Download a ZIP file and add it to your Visual Studio project

You can also install the Splunk Enterprise SDK for C# manually: download it, build it, and import it.

Download the SDK:

- Go to Downloads and download the SDK from GitHub as a ZIP file.

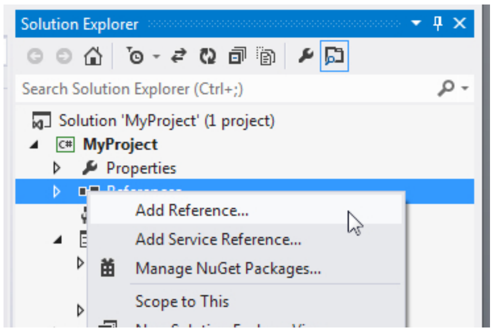
Build the SDK:

1. In Visual Studio, on the **File** menu, click **Open**.
2. Navigate to the location on your hard disk where you extracted the contents of the ZIP file, and open the **splunk-sdk-csharp-pcl.sln** solution file.
3. On the **Build** menu, click **Build Solution**.

The solution builds. You're now ready to import the binary file(s) you want into your project.

Import the binaries:

1. In Visual Studio, open the project you want to work with Splunk Enterprise.
2. In the Solution Explorer, right-click **References**, and then click **Add Reference**.



Enterprise SDK for C#!

Tip: To browse the Splunk Enterprise SDK for C# APIs, right-click a namespace in the Solution Explorer, and then click **View in Object Browser**. Click the triangle next to the namespace to view its contents. Click the triangle next to each namespace to view its classes. For API reference documentation, see Splunk Enterprise SDK for C# Reference.

Download a ZIP file from our Github repo and add it to your Visual Studio project

You can install the latest version of the Splunk Enterprise SDK for C# from our Github repo. Go to https://github.com/splunk/splunk-sdk-csharp-pcl and click the **Download ZIP** button to download the SDK as a ZIP file. Then, follow the instructions starting with "Build the SDK" in "Download a ZIP file and add it to your Visual Studio project".

Xamarin

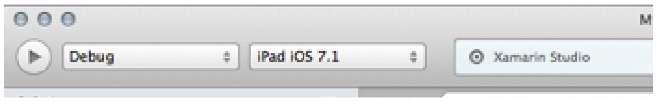
You can install the Splunk Enterprise SDK for C# in Xamarin Studio in several different ways:

- Install via NuGet in Xamarin Studio by searching available packages.
- Download a ZIP file and add it to your Xamarin Studio project.
- Download a ZIP file from our GitHub repo and add it to your Visual Studio project.

Search available NuGet packages

To install the Splunk Enterprise SDK for C# by searching available NuGet packages:

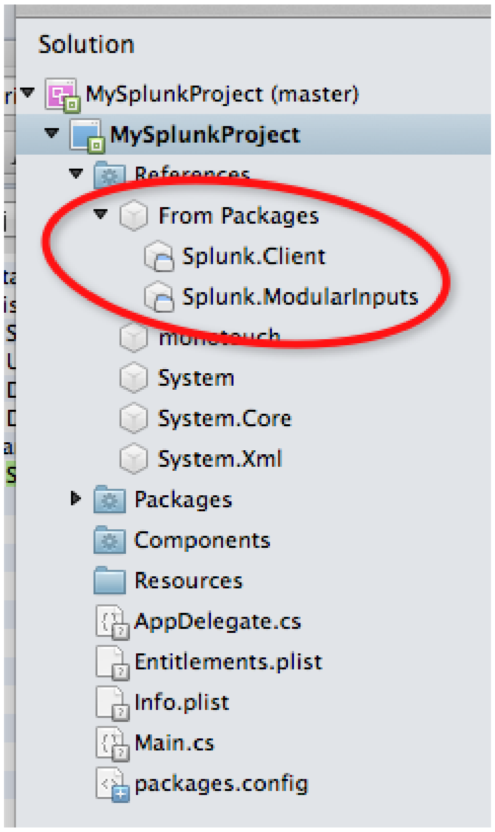
1. Open the project you want to work with Splunk Enterprise.
2. Right-click the name of your project in the **Solution** pane, point to **Add**, and then click **Add Packages**.



- The **Splunk PCL Client for .NET** is the main SDK.
- The **Splunk Modular Input SDK for .NET** enables you to add the ability to create custom Splunk Enterprise modular inputs using C#.
- The **Splunk Logging Libraries for .NET** contains libraries you can include in your C# projects that will log app activity back to Splunk Enterprise via UDP or TCP.

Disregard the **Splunk Enterprise SDK for .NET** package, as it is the old version of the Splunk Enterprise SDK for C#.

The packages you chose to install are now listed in your project references under **From Packages**. You can also view information about what happened during installation by clicking the **View** menu, pointing to **Pads**, and then clicking **Package Console**.

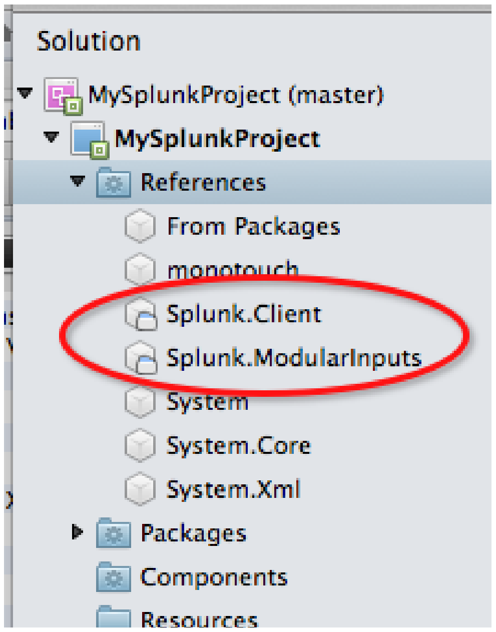


2. In the Solution pane, right-click **References**, and then click **Edit References**.
3. In the Edit References dialog box, click the **.Net Assembly** tab.
4. Navigate to the location on your hard disk where you extracted the contents of the ZIP file, and find the file that corresponds to the library you want:
    - **/src/Splunk.Client/bin/Debug/Splunk.Client.dll**
    - **/src/Splunk.ModularInputs/bin/Debug/Splunk.ModularInputs.dll**

    Because the modular inputs component relies on the main component, you'll find both **Splunk.Client.dll** and **Splunk.ModularInputs.dll** at the second path.

5. Select the files, click **Add**, and then click **OK**.

The packages you chose to install are now listed in your project references.



Build the Splunk Enterprise SDK for C# and examples

You can view premade examples of the Splunk Enterprise SDK for C# in action. Go to Downloads and download the SDK from GitHub as a ZIP file, then extract the contents of the file.

Then, install and build the SDK examples.

Build the examples using Visual Studio

If you are using Visual Studio:

1. Navigate to the extracted Splunk Enterprise SDK for C# directory, open it, and then double-click the **splunk-sdk-csharp-pcl** directory.
2. Double-click splunk-sdk-csharp-pcl.sln to open the SDK solution in Visual Studio.
3. On the **Build** menu, click **Build Solution**.

This will build the SDK, the examples, and the unit tests. For more information about the examples, see Examples.

You can make things easier by saving your host, port, scheme, and login information in the **.splunkrc** file so you don't have to enter them as command-line arguments every time you run an example. For more information, see Save your connection info.

How to use the Splunk Enterprise SDK for C#

This section of the Splunk Enterprise SDK for C# documentation describes how to accomplish some of the most common Splunk tasks.

- Connect to Splunk Enterprise
- Log on to and off of Splunk Enterprise
- Control access
- Perform searches
- Create, run, and edit jobs
- Create, run, and edit reports
- Manage applications
- Create and manage modular inputs

How to connect to Splunk Enterprise using the Splunk Enterprise SDK for C#

Before you log on to Splunk Enterprise, you need to connect to it.

Add a using directive

Remember that before adding capabilities to your app, you must first add a **using** directive for the Splunk client library to your **using** block:

```
using Splunk.Client;
```

Create a Service object

Then, create an instance of the **Service** class, which is the primary entry point for the client library. For example, this code snippet creates an instance of the **Service** class using default scheme, host, port, and namespace values:

```
// Create a Service instance
var service = new Service(new Uri("https://localhost:8089"));

// Log in
await service.LogOnAsync("admin", "yourpassword");
```

How to log on and off Splunk Enterprise using Splunk Enterprise SDK for C#

Use the **LogOnAsync** method of the **Service** class to log on to Splunk Enterprise. For example:

```
await service.LogOnAsync(username, password);
```

This code snippet assumes that you've retrieved a valid username and password from the user, and assigned each to the *username* and *password* variables.

Similarly, use the **LogOffAsync** method of the **Service** class to log off of Splunk Enterprise:

```
await service.LogOffAsync();
```

For a complete example that you can run locally, see the **authenticate** example in the **examples** directory.

Reference

- **Service.LogOnAsync(string *username*, string *password*)**
- **Service.LogOffAsync()**

How to control access using the Splunk Enterprise SDK for C#

You can control access to Splunk Enterprise (manage users and passwords) from your app using the Splunk Enterprise SDK for C#. Accessing user passwords requires admin access.

The access APIs

The classes for working with access are listed here. These APIs wrap the **storage/passwords** endpoint of the Splunk Enterprise REST API.

- **StoragePassword**: A Splunk Enterprise storage password.
- **StoragePasswordCollection**: A collection of Splunk Enterprise storage passwords.
- **StoragePasswordCollection.Filter**: Provides arguments for retrieving a **StoragePasswordCollection**.

How to run searches and jobs using the Splunk Enterprise SDK for C#

ON THIS PAGE

With the Splunk Enterprise SDK for C#, you can add Splunk Enterprise search capabilities to your program. The Splunk Enterprise SDK for C# supports multiple search types, some of which create search jobs:

- Searches that produce search jobs and stream the results:
  - Normal search: A normal search runs asynchronously. It returns a search job immediately, but you retrieve the results when the search has finished. Poll the job to determine its status. You can also preview the results if "preview" is enabled.
  - Blocking search: A blocking search runs synchronously. It does not return a search job until the search has finished, so there is no need to poll for status. Blocking searches don't work with real-time data.
  - Real-time search: A real-time search runs asynchronously and immediately. It returns a search job immediately and searches live events as they stream into Splunk for indexing. The events that are returned match your criteria within a specified time range window.

- Searches that only stream the results:
  - One-shot search: A one-shot search runs synchronously and immediately. Instead of returning a search job, this mode returns the results of the search once completed.
  - Export search: An export search runs asynchronously and immediately, does not return a search job, and starts streaming results right away. This search is useful when you're dealing with large amounts of data, and is ideal for real-time data. Export searches always have preview results enabled.

```
...
```

SHOW MORE

The Splunk Enterprise SDK for C# defines the **Observable<*T*>** class to provide push-based notifications to observers. This example uses a subscription to observe the collection of search result records, and prints each record to the console.

```
job = await service.Jobs.CreateAsync("search index=_internal | head 10");
using (stream = await job.GetSearchResultsAsync())
{
    stream.Subscribe(new Observer<SearchResult>(
        onNext: (result) =>
        {
            Console.WriteLine(string.Format("{0:D8}: {1}", stream.ReadCount, result));

...
```

SHOW MORE

Reference

- **Job** class
  - **Job.GetSearchResultsAsync()** method

- **SearchResult** class
- **SearchResultStream** class
- **JobCollection** class
  - **JobCollection.CreateAsync()** method

- **Observable<*T*>**

Blocking searches

Running a blocking search creates a search job and runs the search synchronously. The job is returned after the search has finished and all the

  - **Job.GetSearchResultsAsync()** method

- **JobArgs** class
- **SearchResultStream** class
- **JobCollection** class
  - **JobCollection.CreateAsync()** method

Real-time searches

Real-time searches return live events as they are indexed, and this type of search continues to run as events continue to arrive. So, to view results from a real-time search, you must view the preview results. You can think of the previews as a snapshot of the search results at that moment in time. A few search job parameters are required to run a real-time search, which you can set by using the methods of the JobArgs class:

- Set the execution mode (**ExecutionMode**) to **Normal**.
- Set the search mode (**SearchMode**) to **RealTime**, which also enables previews.
- If you want to specify a sliding window for your search (for example, everything in the last hour), you can set the earliest and latest times to relative time modifiers. For example, set **EarliestTime** to rt-1h and **LatestTime** to rt and the time range is continuously updated based on the current time.

The real-time search continues to run until you either cancel or finalize it. If you cancel the search, the search job is deleted. If you finalize it, the search is stopped and the search job is completed.

The following code sample is taken from the **search-realtime** example in the **examples** directory. Two things are of note here:

- Because the search is real-time, we've built in a mechanism for canceling the search.
- With real-time searches, you view search *preview* results, hence our use of the **Job.GetSearchPreviewAsync()** method.

The following example shows a real-time search of your internal index.

```
static void Main(string[] args)
```

- **JobCollection** class
  - **JobCollection.CreateAsync()** method
- **SearchMode** enumeration
- **SearchResult** class
- **SearchResultStream** class
  - **SearchResultStream.FieldNames** property
  - **SearchResultStream.IsFinal** property

One-shot searches

Unlike other searches, the *one-shot search* does not create a search job, so you can't access it using the job-related classes. Instead, use the **Service.SearchOneShotAsync()** method. Though this method is asynchronous, be aware that a one-shot search is a synchronous operation on the Splunk Enterprise server. That is, it will only return results once the search has finished.

To set properties for the search (for example, to specify a time range to search), you'll need to create an argument map (using a **JobArgs** object) with the parameter key-value pairs. Some common parameters are:

- **EarliestTime**: Specifies the earliest time in the time range to search. The time string can be a UTC time (with fractional seconds), a relative time specifier (to now), or a formatted time string.
- **LatestTime**: Specifies the latest time in the time range to search. The time string can be a UTC time (with fractional seconds), a relative time specifier (to now), or a formatted time string.
- **rf**: Specifies one or more fields to add to the search.

For a full list of possible properties, see the parameters for the search/jobs endpoint in the Splunk Enterprise *REST API Reference Manual*, although most of these parameters don't apply to a one-shot search.

This example runs a one-shot search within a specified time range and displays the results in XML.

```
using (SearchResultStream searchResultStream =
```

Export searches

An *export search* runs immediately, does not return a search job, and starts streaming results right away. This search is useful for exporting large amounts of data from Splunk Enterprise. Export searches have two modes: reporting and non-reporting.

- A *reporting* search aggregates events from the index and streams real-time previews followed by a final result set when the search completes.
- A *non-reporting* search passes events, potentially with modifications to each one, directly from the index and sends back final results in chunks as it gets them from the index.

For reporting mode export searches, use the **ExportSearchPreviewsAsync()** method of the **Service** object. For non-reporting mode export searches, use the **ExportSearchResultsAsync()** method.

To access preview results, first create a **SearchPreviewStream** object, which is a streaming XML reader that reads **SearchPreview** objects. Then use the **Service.ExportSearchPreviewsAsync()** method to export an observable sequence of search result previews to the reader. In this example, the fields of each preview search event are listed (the **SearchPreview.Results** property), along with whether each event is final or partial (the **SearchPreview.IsFinal** property):

```
// create streaming reader
SearchPreviewStream searchPreviewStream;
// export the search result previews
using (searchPreviewStream = service.ExportSearchPreviewsAsync("search index=_internal | head 100").Result)
{
    int previewNumber = 0;

...
```

SHOW MORE

To access the final results of an export search, simply use the final results versions of the same APIs, as noted here:

EXPORT PREVIEW SEARCH RESULTS API	EXPORT SEARCH RESULTS API	DESCRIPTION
Service.ExportSearchPreviews Async() method	**Service.ExportSearchResults Async()** method	Asynchronously exports an observable sequence of search result previews or search results

splunk>

Contact Community Slack Privacy Terms of Service

How to work with reports (saved searches) using Splunk Enterprise SDK for C#

This topic focuses on working with reports (also known as *saved searches*) programmatically using the Splunk Enterprise SDK for C#. For more information about working with search jobs, see How to run searches and jobs.

The report APIs

The classes for working with reports are:

- The **SavedSearch** class provides an object representation of a report in Splunk Enterprise.
- The **SavedSearchAttributes** class provides the arguments for creating a **SavedSearch** object.
- The **SavedSearchCollection** class represents a collection of reports.
- The **SavedSearchDispatchArgs** class provides the arguments for retrieving transformed search results.
- The **SavedSearchTemplateArgs** class provides custom arguments to a **SavedSearch** object.

Access these classes through an instance of the **Service** class. Access a collection of reports using the **Service.SavedSearches** property. From there you can access individual items in the collection and create new ones. For example, here's a simplified program (from **saved-searches**) for creating a new report:

```
// connect to Splunk Enterprise
var service = new Service(connectArgs);
```

How to manage applications using Splunk Enterprise SDK for C#

The Splunk Enterprise SDK for C# allows you to manage applications on your Splunk Enterprise instance.

The application APIs

The APIs for working with applications are:

- The **Application** class provides an object representation of an app in Splunk Enterprise.
- The **ApplicationArchiveInfo** class provides information about an application archive produced by Splunk Enterprise.
- The **ApplicationAttributes** class provides arguments for setting the attributes of an application.
- The **ApplicationCollection** class provides an object representation of a collection of Splunk Enterprise apps.
- The **ApplicationSetupInfo** class represents the setup information for an application.
- The **ApplicationUpdateInfo** class provides an object representation for the update information available for an application.
- The **Applications** property of the **Service** object gets the applications.

For example, the following code snippet (from the **list_apps** example) uses the **ApplicationCollection.GetAllAsync()** method to retrieve a collection of all the installed applications that the logged-in user has access to.

```
Console.WriteLine("List of Apps:");
// get a collection of all applications
await service.Applications.GetAllAsync();
foreach (var app in service.Applications)
```

How to create modular inputs in Splunk Enterprise SDK for C#

ON THIS PAGE

Support for modular inputs enables you to add new types of custom data inputs to a Splunk platform deployment that are treated as native data inputs. Your users interactively create and update instances of modular inputs using Splunk Web, just as they do for native inputs. If you're not already familiar with modular inputs, see Overview of modular inputs.

Some examples of modular inputs include:

- Real-time Windows performance monitoring: A perfmon input that runs a separate process for every input defined.
- Splunk Add-on for Microsoft PowerShell: A modular input that runs PowerShell 3.0 scripts to collect data.
- Examples of modular input scripts and configurations: Examples include a modular input that streams JSON data from a Twitter source to the Splunk platform, and a modular input that streams data from the Amazon S3 data storage service to the Splunk platform.

The Splunk Enterprise SDK for C# includes built-in support for creating and using modular inputs with C#.

This topic shows you how to create modular inputs using the Splunk Enterprise SDK for C#, and how to integrate them into your Splunk app.

Why use modular inputs?

- Log easily back to Splunk Enterprise from within your modular input.
- Easily access Splunk Enterprise capabilities via the SDK from within your modular input.
- Package your modular input within a Splunk Processing Language (SPL) query to deploy it to other Splunk Enterprise instances.

Modular input SDK example

The Splunk Enterprise SDK for C# includes a modular input example in the **examples** directory: **random-numbers**. To run these examples, you'll need to install them. (They are not installed with the SDK installation process.)

1. Set the **$SPLUNK_HOME** environment variable to the root directory of your Splunk Enterprise instance.
2. Build the project using your IDE, and then copy the **random-numbers** directory from:

   ```
   ...\splunk-sdk-csharp-pcl\examples\random-numbers\bin\Debug\app
   ```

   into

   ```
   $SPLUNK_HOME\etc\apps\
   ```

3. Restart Splunk Enterprise. From Splunk Home, click the **Settings** menu. Under **System**, click **Server Controls**. Click **Restart Splunk**.

After you have installed the example modular input, it appears alongside other data inputs. To try out the modular input:

1. From Splunk Home, click the **Settings** menu.
2. Under **Data**, click **Data inputs**, and find the new "Random numbers" modular input.
3. Click **Random numbers**, and then click **New**.
4. Give the input some values: Enter a name, a minimum value, and a maximum value.
5. Click **Save**.

Splunk Enterprise starts, it runs the script to determine the modular input's behavior and configuration.

- *Validate the script's configuration (optional).* Whenever a user creates or edits an input, Splunk Enterprise can call the script to validate the configuration.
- *Stream data.* The script streams event data that can be indexed by Splunk Enterprise. Splunk Enterprise invokes the script and waits for it to stream events.

Splunk Enterprise defines a contract that all implementations of modular inputs must obey. The Splunk Enterprise SDK for C# provides a class that implements most of the details of that contract, leaving only the details of your particular modular input to be filled in.

A modular input written using the Splunk Enterprise SDK for C# should inherit from the **Splunk.ModularInputs.ModularInput** class. The preceding three steps are accomplished as follows using the Splunk Enterprise SDK for C#:

- Return the introspection scheme*:* Override the **Scheme** property.
- Validate the script's configuration (optional):_ Override the **Validate** method.
- Stream data:_ Override the **StreamEventsAsync** method.

In addition, you must define a minimal **Main** method.

Here's the framework of a new modular input script created in C#:

```
namespace random_numbers
{
    using Splunk.ModularInputs;
    public class RandomNumbers : ModularInput
    {
        public static int Main(string[] args)
        {
            return Run<RandomNumbers>(args);

...
```

SHOW MORE

Splunk Enterprise calls the program a number of times. The first time is when Splunk Enterprise starts; it calls the program to get a scheme for the modular input by invoking the **get** method of the **Scheme** class. If the script is using external validation, Splunk Enterprise calls the program

```
        Arguments = new List<Argument>
        {
            new Argument
```

...

SHOW MORE

Validate the configuration

You can specify a string to the **Validation** property of an **Argument** in a **Scheme**, and Splunk Enterprise will perform internal validation based on what you specify. For the format of the built-in validation functions you can use, see Validation functions. If you need more functionality than built-in validation offers, you have two more places you can perform validation.

First, set the **UseExternalValidation** property of the modular input's **Scheme** object to true. Then you can set delegates to do validation for each parameter, and define a method to do validation on the whole set of parameters.

To add a delegate to a particular parameter, create an instance of **ValidationHandler** and set it as the value of the argument's **ValidationDelegate** property.

ValidationHandler is defined as follows:

```
public delegate bool ValidationHandler(Parameter parameter, out string errorMessage);
```

To define validation on the whole collection of parameters, override the **Validate** method. The **Validate** method takes a **Validation** object—which contains some server metadata and the proposed parameter values—and an out parameter **errorMessage** that returns an error message if there is an error. If **Validate** returns true, then the parameters are taken to be valid and the error message is ignored. If **Validate** returns false, then the error message is returned to Splunk Enterprise.

Following is an example implementation of **Validate** from the **random-numbers** example:

```
public override bool Validate(Validation validation, out string errorMessage)
{
    double min = validation.Parameters["min"].ToDouble();
    double max = validation.Parameters["max"].ToDouble();
    if (min >= max) {
        errorMessage = "min must be less than max.";
```

you would expect—string, int, long, float, double, and bool for single valued parameters; and arrays of the same for multivalued parameters. An invalid cast will result in a runtime error.

Then stream events back to the server using the **EventWriter** object's **QueueEventForWriting** method. Events are specified as instances of a structured class **Event**.

Following is an example implementation of the **StreamEventsAsync** method:

```
public override async Task StreamEventsAsync(
    InputDefinition inputDefinition, EventWriter eventWriter)
{
    double min = inputDefinition.Parameters["min"].ToDouble();
    double max = inputDefinition.Parameters["max"].ToDouble();
    while (true)
    {

...
```

SHOW MORE

To add the modular input to Splunk Enterprise

With your modular input script completed, you're ready to integrate it into Splunk Enterprise. First, build the project, then package the script, and then install the modular input.

Build the project

Build your project, taking care of any errors that may appear.

Once you've built the project successfully, an executable with the name of your project will appear within your project's **bin** directory. Keep track of this file, as well as the .DLL files here; you'll need them later.

Package the script

To add a modular input that you've created in C# to Splunk Enterprise, you'll need to add it as a Splunk Enterprise app. First, create the files you'll need to package the input as an app.

```
[launcher]
author=Me
description=My great modular input
version = 1.0
```

inputs.conf.spec

When creating this file, in addition to replacing *modinput_name* with the name of your modular input's project, do the following:

- After the asterisk (*), type a description for your modular input.
- Add any arguments to your modular input as shown. You must list every argument that you defined in the Scheme method.

```
[modinput_name://<name>]
*My great modular input.
arg1 = <value>
arg2 = <value>
count = <value>
```

Directories

Next, create a directory that corresponds to the name of your modular input script—for instance, "*modinput_name*"—in a location such as your **Documents** directory. (It can be anywhere; you'll copy the directory over to your Splunk Enterprise directory at the end of this process.)

1. Within this directory, create the following directory structure:

```
modinput_name/
    bin/
    default/
    README/
```

2. Copy your modular input executable (*modinput_name.exe*), the two .DLL files alongside the executable, and the files you created in the previous sections so that your directory structure looks like this:

```
        $SPLUNK_HOME/etc/apps
```

3. Restart Splunk Enterprise: From Splunk Home, click the **Settings** menu. Under **System**, click **Server Controls**. Click **Restart Splunk**.

After you have installed your modular input, it appears alongside other data inputs. To try out the modular input:

1. From Splunk Home, click the **Settings** menu.
2. Under **Data**, click **Data inputs**, and find the name of your modular input.
3. Click the name of your modular input, and then click **New**.
4. Give the input some values: Fill in the settings that you specified when you created the modular input script.
5. Click **Save**.
6. Go to the **Search & Reporting** app, and search using the input you just created. For example, to search using an instance of the *modinput_name* modular input called *example*, set **source** as shown here:

```
source="modinput_name://example"
```

You've now configured an instance of your modular input as a Splunk Enterprise input.

How to debug modular inputs in Splunk Enterprise SDK for C#

This topic describes how to debug modular inputs using the Splunk Enterprise SDK for C#, plus how to effectively use the diagnostic information generated by the SDK.

Modular input debugging

A common developer request has been for a mechanism to debug modular inputs created with the Splunk Enterprise SDK for C#. With the Splunk Enterprise SDK for C# version 2.1.0 and later and Microsoft Visual Studio, now you can, using *remote debugging*. The basic process is:

1. Enable: Enable remote debugging.
2. Deploy: Compile and deploy the updated binary to a Splunk Enterprise instance, and then enable the new input in Splunk Enterprise.
3. Attach: Add a break point to the code, and then remotely attach to a running modular input process.
4. Debug: Step through and over, use the watch window, and so on.

Though this should also work with Xamarin Studio, it has not been tested, and is not supported at this time.

Attach

Add a break point to the code:

- If you specified the **StreamEvents** debugger attach point, add a break point within the **StreamEventsAsync** method.
- If you specified the **ValidateArguments** debugger attach point, add a break point within the **Validate** method.

You can only attach for the **StreamEventsAsync** and **Validate** methods.
The **Validate** method has about 30 seconds before Splunk Enterprise kills the process, so work fast if you want to debug validation.

Now, in Visual Studio, open the **Attach to Process** window and attach to the running modular input process.

If you can't see the modular input process (which is identified by the name of the modular input, followed by **.exe**), select the **Show processes from all users** checkbox.

Debug

As soon as you attach to the process, Visual Studio breaks on the break point you specified. At this point, you can step through and over, use the watch window, and so on.

Modular input diagnostics

You can use the Splunk Enterprise SDK for C# to capture diagnostic information about failures in modular inputs.

In the Splunk Enterprise SDK for C# version 2.1.0 and later, the following data is captured:

- All trapped and untrapped exceptions, including the stack trace. This applies to **Validate**, **Scheme**, and **StreamEventsAsync**.
- Wherever possible, the input instance name is captured. For instance, if you have several GitHub commits inputs, you'll know exactly which

Splunk Enterprise SDK for C# examples

The Splunk Enterprise SDK for C# provides a number of examples that show how to interact with the Splunk Enterprise SDK for C# PCL. The topics in this section contain descriptions and instructions for all the examples that are included in the SDK.

Examples are located in the `\splunk-sdk-csharp-pcl\examples` directory. When you build the SDK, the examples are built as well.

This version of the Splunk Enterprise SDK for C# contains the following examples:

EXAMPLE	DESCRIPTION
authenticate	Authenticates to the server and print the received token.
environment_variable_monitor	Creates an example modular input that monitors changes to a system environment variable and logs them to Splunk Enterprise.
github-commits	Modular input example that streams commit events in a given GitHub repository.
list_apps	Lists apps installed on the server.
mock-context	Mock context for unit testing.
mock-interface	Mock interface for unit testing.
mock-object	Mock object for unit testing.
normal-search	Starts a normal search and polls for completion to find out when the search has finished.
random-numbers	Creates an example modular input that generates random number for logging by Splunk Enterprise.
saved-searches	Checks whether a report (saved search in Splunk Enterprise 5) already exists; if not, creates and runs a new report.
search-export	Starts an export search and displays preview results.
search-realtime	Creates a new real-time search and displays preview results, waiting for the user to cancel the search

Developer Guide

Reference

Tutorials

Downloads

Examples

Search

Save your connection info for the Splunk Enterprise SDK for C#

You can save your Splunk® Enterprise host, port, and scheme (HTTP or HTTPS) information--and even your login credentials--for examples and unit tests. This might make it easier to run them, since you will not need to specify this information every time you run a sample.

To connect to Splunk Enterprise, many of the SDK examples and unit tests take command-prompt arguments that specify values for the host, port, scheme, and login credentials. For convenience during development, you can store these arguments as key-value pairs in a text file named **.splunkrc**. Then, when you don't specify these arguments at the command prompt, the SDK examples and unit tests use the values from the **.splunkrc** file.

Important: Storing user login credentials in the **.splunkrc** file is only for convenience during development; this file isn't part of the Splunk platform and shouldn't be used for storing user credentials for production. And, if you're at all concerned about the security of your credentials, just enter them at the command prompt rather than saving them in the .splunkrc file.

To use a **.splunkrc** file:

1. Create a new file named **.splunkrc** in the current user's home directory.

   On *nix or OS X:

   - Save the file as:

     ```
     ~/.splunkrc
     ```

   On Windows:

   - Save the file as:

```
# Splunk username
username=admin
# Splunk password
password=yourpassword
# Access scheme (default: https)
scheme=https
```

The examples and unit tests are ready to run with no additional connection arguments required!

Keep in mind that if you haven't configured a **.splunkrc** file, you may need to include **host**, **port**, **username**, **password**, and **scheme** arguments as key-value pairs when you run an example from the command line. You don't need to include any keys that take default values.

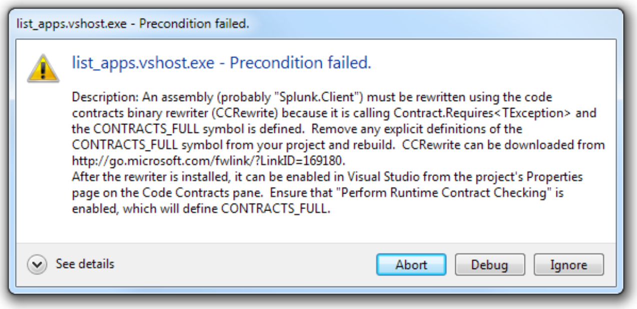
Troubleshooting Splunk Enterprise SDK for C#

The following troubleshooting topics are available:

- Troubleshooting "Precondition failed" error message
- Troubleshooting connectivity to the Splunk Enterprise API

Troubleshooting "Precondition failed" error message

If, after attempting to run an SDK example, you encounter the following error message, you need to install a separate component.



list_apps.vshost.exe - Precondition failed.

list_apps.vshost.exe - Precondition failed.

Description: An assembly (probably "Splunk.Client") must be rewritten using the code contracts binary rewriter (CCRewrite) because it is calling Contract.Requires<TException> and the CONTRACTS_FULL symbol is defined. Remove any explicit definitions of the CONTRACTS_FULL symbol from your project and rebuild. CCRewrite can be downloaded from http://go.microsoft.com/fwlink/?LinkID=169180.
After the rewriter is installed, it can be enabled in Visual Studio from the project's Properties page on the Code Contracts pane. Ensure that "Perform Runtime Contract Checking" is enabled, which will define CONTRACTS_FULL.

To run the examples, you'll need to install the code contracts binary rewriter. Follow the instructions contained in the error message: Exit Visual Studio and install CCRewrite from https://go.microsoft.com/fwlink/?LinkID=169180. Then, rebuild the solution and run the example again.

Troubleshooting connectivity to Splunk Enterprise for the Splunk Enterprise SDK for C#

ON THIS PAGE

There are a number of ways to troubleshoot connectivity to Splunk Enterprise when using a Splunk SDK or logging library. This topic covers a few methods that may help you figure out what is causing problems.

There are two families of problems that are most common:

- Issues with general connectivity to the API
- Issues involving certificate validation or the Splunk Enterprise security protocol configuration

General connectivity issues

If you're unable to successfully connect to the Splunk Enterprise API, first try using the command-line tool to log into the Splunk API via cUrl. For example, try the following command, replacing the placeholders with actual values from your setup:

```
curl -k <server:port>/services/auth/login -d username=<username> -d password=<password>
```

Developer Guide

Reference

Tutorials

Downloads

Examples

Search

```
<msg type="WARN">Remote login has been disabled for 'admin' with the default password. Either set the password, or
override by changing the 'allowRemoteLogin' setting in your server.conf file.</msg>
```

Check the `allowRemoteLogin` setting in your Splunk Enterprise instance's **server.conf** file. You have it set to `requireSetPassword` (the default) and have never changed the admin's password from the default.

- You receive an error such as the following:

```
curl: (7) Failed connect to 10.80.9.131:8088; No error
```

Either the URI is wrong, the port number is not correct, or the port is not opened on the firewall.

- If you receive an empty reply, you're using the wrong scheme. For example, you're using HTTP when you should be using HTTPS.

If you get a valid response using cUrl, but the SDK is still failing, then either the credentials or the URI passed in the code that uses the SDK could be wrong. Check your app's configuration.

Security configuration issues

The second type of issues relates to either certificate validation failing, or to the security protocol configuration in Splunk Enterprise.

Certificate validation

Another kind of error you might see relates to certificate validation. By default, most platforms automatically throw an exception when the HTTPS certificate is not valid. Splunk does not return a valid cert by default, which causes this failure.

This can be disabled within your application's code. Using the Splunk Enterprise SDK for C#, you can turn off certificate validation using code such as the following:

```
ServicePointManager.ServerCertificateValidationCallback += (sender, certificate, chain, sslPolicyErrors) =>
```