Splunk SDK for C# v1.0.1

splunk>

Splunk, Inc.
270 Brannan Street
San Francisco, CA 94107

+1.415.568.4200 (Main)
+1.415.869.3906 (Fax)
www.splunk.com

splunk>

Table of Contents

Overview of the Splunk SDK for C#

Welcome to the Splunk® Software Development Kit (SDK) for C#!

> ***Note:*** *The Splunk SDK for C# v1.0.x is deprecated, and has been replaced by the Splunk SDK for C# 2.0. Unlike the Splunk SDK for C# 1.0.x, the Splunk SDK for C# 2.0 abides by .NET guidelines, as well as FxCop and StyleCop rules. In addition, the API client in Splunk SDK for C# 2.0 is a Portable Class Library (PCL), and supports cross-platform development.*
>
> *Version 2 of the SDK is incompatible with version 1, and applications built with Splunk SDK for C# version 1.0.x will not recompile using Splunk SDK for C# version 2.0. See Migrating from Splunk SDK for C# v1.0.x for more information.*
>
> *The Splunk SDK for C# version 1.0.1 will be supported until September 9, 2016. Until that date, Splunk will resolve any security issues that are discovered, but will not add any new features. Splunk encourages all customers still using Splunk SDK for C# v1.0.x to upgrade to the Splunk SDK for C# version 2.0 as soon as possible, but no later than September 9, 2016.*

This SDK is open source and uses the Apache v2.0 license. If you want to make a code contribution, go to the Open Source page for more information.

This overview tells you more about:

What you can do with the Splunk SDK for C#

This SDK contains library code and examples designed to enable developers to build applications using Splunk. With the Splunk SDK for C# you can write C# applications to programmatically interact with the Splunk 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:

- Search your data, run saved searches, and work with search jobs
- Manage Splunk configurations and objects
- Integrate search results into your applications
- Log directly to Splunk
- Present a custom UI

The Splunk SDK for C# supports development in Microsoft Visual Studio 2012. The minimum supported version of the .NET Framework is version 3.5. Visual Studio downloads are available on the Visual Studio Downloads webpage.

The Splunk SDK for C# components

The Splunk developer platform consists of three primary components: *Splunkd*, the engine; *Splunk Web*, the app framework that sits on top of the engine; and the *Splunk SDKs* that interface with the REST API and extension points.

The Splunk SDK for C# lets you target Splunkd by making calls against the engine's REST API and accessing the various Splunkd extension points such as custom search commands, lookup functions, scripted inputs, and custom REST handlers.

Read more about:

- Namespaces
- The Service class
- Entities and collections
- Managing state between the client and server

Namespaces

To account for permissions to view apps, system files, and other entity resources by users throughout a Splunk installation, Splunk provides access to entity resources based on a *namespace*. This is similar to the app/user context that is used by the Splunk REST API when accessing resources using endpoints.

The namespace is defined by:

- An *owner*, which is the Splunk username, such as "admin". A value of "nobody" means no specific user. The "-" wildcard means all users.
- An *app*, which is the app context for this resource (such as "search"). The "-" wildcard means all apps.
- A *sharing* mode, which indicates how the resource is shared. The sharing mode can be:
    - "user": The resource is private to a specific user, as specified by *owner*.
    - "app": The resource is shared through an app, as specified by *app*. The *owner* is "nobody", meaning no specific user.
    - "global": The resource is globally shared to all apps. The *owner* is "nobody", meaning no specific user.
    - "system": The resource is a system resource (*owner* is "nobody", *app* is "system").

In general, when you specify a namespace you can specify any combination of *owner*, *app*, and *sharing* the SDK library will reconcile the values, overriding them as appropriate. If a namespace is not explicitly specified, the current user is used for *owner* and the default app is used for *app*.

Here are some example combinations of *owner*, *app*, *sharing*:

- List all of the saved searches for a specific user named Kramer: `kramer, -, user`
- Create an index to be used within the Search app: `nobody, search, app`

The Service class

The **Service** class is the primary entry point for the client library. Construct an instance of the **Service** class and provide the login credentials that are required to connect to an available Splunk server. There are different ways to construct the instance and authenticate; here's one way:

```
// Define the context of the Splunk service
ServiceArgs svcArgs = new ServiceArgs();
svcArgs.Host = "localhost";
svcArgs.Port = 8089;

// Create a Service instance and log in
Service service = new Service(svcArgs);
service.Login("admin", "changeme");
```

Once the **Service** instance is created and authenticated, 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 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 prints all Splunk users:

```
foreach (var user in service.GetUsers().Values) {
    System.Console.WriteLine(user.RealName);
}
```

Similarly, the following code prints all the Splunk apps:

```
foreach (var app in service.GetApplications().Values) {
    System.Console.WriteLine(app.Label);
}
```

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.

The class hierarchy for the core SDK library is as follows:

```
Service
Resource
    Entity
    ResourceCollection
    EntityCollection
```

Managing state between the client and server

When you create an object for an entity, the entity's properties are read and copied from the server, creating a local snapshot of those values. Any set operations on the object are only made to the local object. Your changes are not made on the server until you explictly call the object's **Update** method, which uploads all the changes you've made to that object. And, changes made on the server don't affect your local copy unless you call the object's **Refresh** method, which replaces any changes you have made with an updated snapshot from the server.

The SDK does not perform validation on values when you set them, but rather passes these values to the server for validation. Any error messages from the server are then sent back to you through the SDK.

There are two ways to set values for an entity:

- Use the properties that are available for the entity, then call **Update** to upload all the changes you've made to this entity. For example:

  ```
  EventTypeCollection eventTypeCollection = service.GetEventTypes();
  EventType eventType = eventTypeCollection.Create("test", "index=_internal *");
  eventType.Description = "This is a test";
  eventType.Priority = 3;
  eventType.Update();
  ```

- Create a dictionary of arguments, and then supply all of the specified arguments for this entity by calling the **Create** method to create the entity. For example:

```
var eventTypeCollection = service.GetEventTypes();

var args = new Args {
    { "description", "This is a test" },
    { "priority", 3 },
};

var eventType = eventTypeCollection.Create("test", "index=_internal *", args);
```

  A call to **Update**(*args*) also includes any changes you made to properties. For example:

```
var eventTypeCollection = service.GetEventTypes();
var eventType = eventTypeCollection.Create("test", "index=_internal *");
var args = new Args();

eventType.Priority = 3;
eventType.Description = "This is a test";
eventType.Update(args); // Updates both the description and priority
```

Argument dictionaries require a little more work because you must know which arguments are allowed for that entity, carefully specifying the case-sensitive argument name with a value in the expected format. However, using an argument dictionary also lets you set values for any argument that is allowed for a given entity—even if there isn't a corresponding property method for it. In addition, you must use an argument dictionary if you want to initialize an object with values, because object properties aren't available until after an object has been created.

splunk>

Getting started with the Splunk SDK for C#

Getting started with the Splunk® SDK for C# is easy:

1. Get Splunk Enterprise
2. Check prerequisites
3. Install the SDK
4. Build the examples
5. Save your connection info (optional)

Get Splunk Enterprise

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

> ***Note:*** *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 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, Mac 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 read the REST API Overview or browse the Splunk Enterprise REST API Reference.

Check prerequisites

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

Windows 7 Service Pack 1 (SP1) or later

Visual Studio 2012 or later. Visual Studio downloads are available on the Visual Studio Downloads webpage on the Microsoft website.

> ***Notes:*** *Visual Studio is not required, strictly speaking, to code using the Splunk SDK for C#. But if you want to build the SDK source code, you'll need Visual Studio 2012 or later.*

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

Install the SDK

There are three options for installing the Splunk® SDK for C#:

- 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.

NuGet

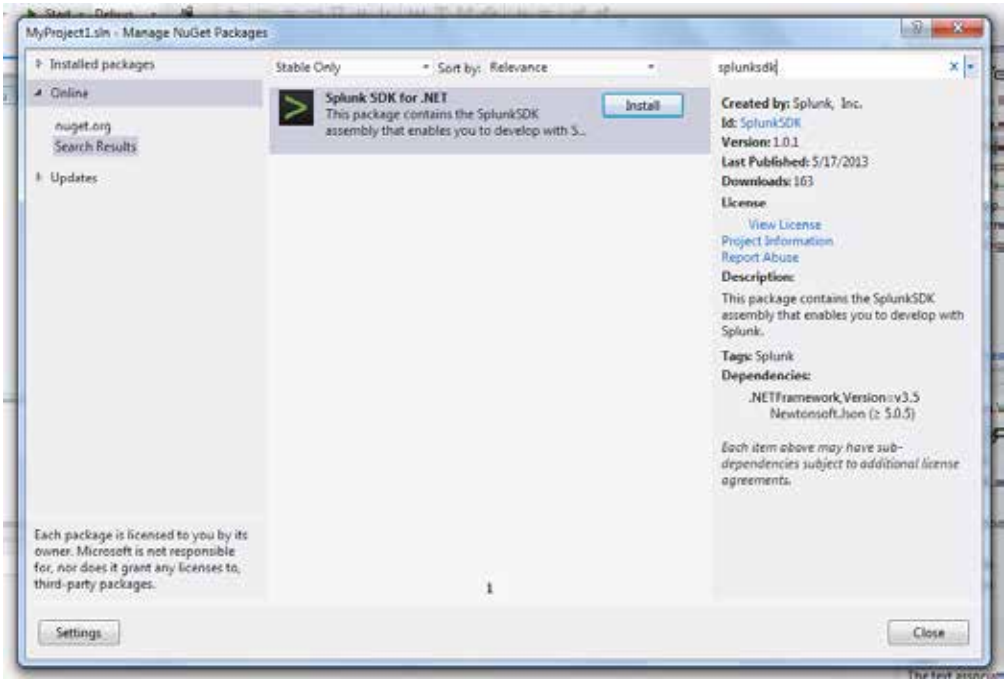
You can install the Splunk SDK for C# using NuGet in Visual Studio in two different ways:

- By searching available NuGet packages
- By using the Package Manager Console

Search available NuGet packages

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

1. Open the project you want to work with Splunk Enterprise.
2. On the **TOOLS** menu, point to **Library Package Manager**, and then click **Manage NuGet Packages for Solution**.
3. In the **Manage NuGet Packages** window, click **Online** from the list on the left, and then enter `SplunkSDK` into the **Search Online** field in the upper-right corner. The **Splunk SDK for .NET** package appears in the list, as shown here:



4. Click the **Install** button for the **Splunk SDK for .NET** package.
5. 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 SDK for C# and its dependencies to your project.

> ***Tip:*** *To browse the Splunk SDK for C# APIs, right-click* ***SplunkSDK*** *in the Solution Explorer, and then click* ***View in Object Browser****. Click the triangle next to* ***SplunkSDK*** *to view its namespaces. Click the triangle next to each namespace to view its classes.*

Use the Package Manager Console

To install the Splunk SDK for C# by using the Package Manager Console, do the following:

1. Open the project you want to work with Splunk Enterprise.
2. On the **TOOLS** menu, point to **Library Package Manager**, and then click **Package Manager Console**.
3. In the Package Manager Console at the **PM>** prompt, type the following: `Install-Package SplunkSDK`

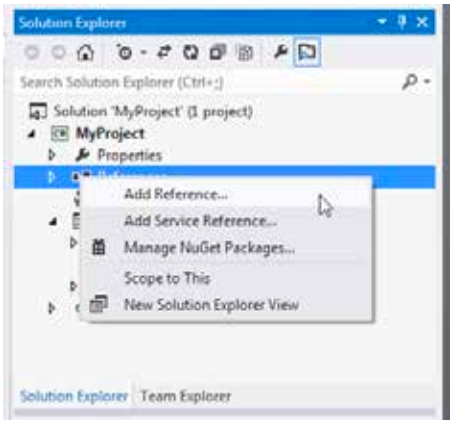
The Package Manager adds the Splunk SDK for C# and its dependencies to your project.

> ***Tip:*** *To browse the Splunk SDK for C# APIs, right-click* ***SplunkSDK*** *in the Solution Explorer, and then click* ***View in Object Browser****. Click the triangle next to* ***SplunkSDK*** *to view its namespaces. Click the triangle next to each namespace to view its classes.*

Download a ZIP file

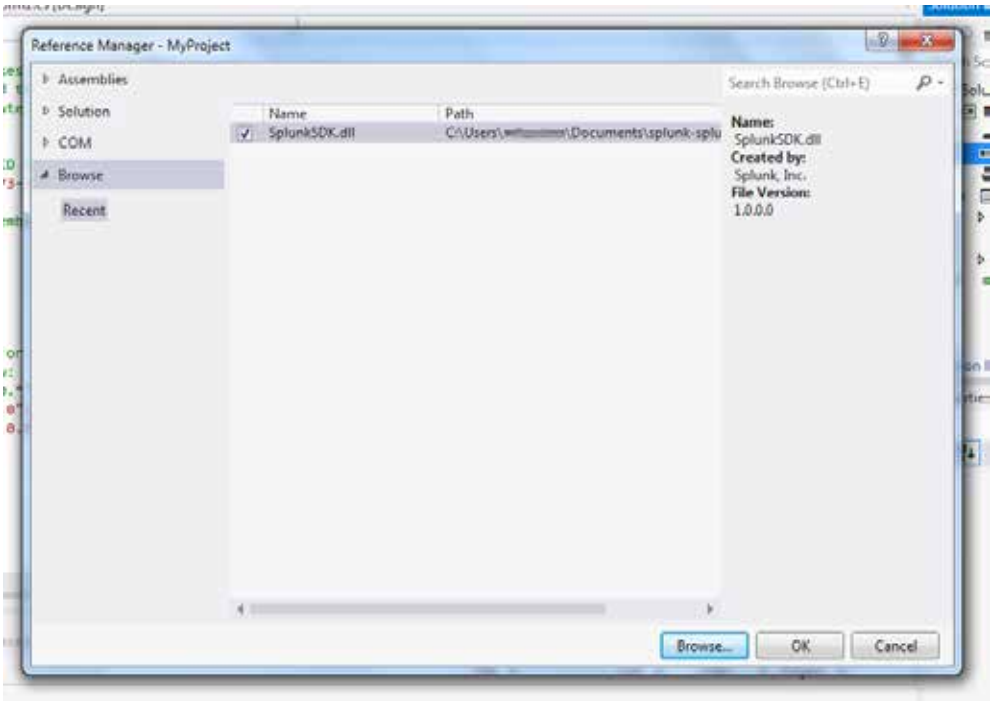
You can also install the Splunk SDK for C# manually—just download and import it.

1. Download the SDK as a ZIP file, and then extract the contents of the file. If you want, you can verify the integrity of the downloaded ZIP file:
   - Download MD5.
   - Download SHA-512.
2. In Visual Studio, open the project you want to work with Splunk Enterprise.
3. In the Solution Explorer, right-click **References**, and then click **Add Reference**.

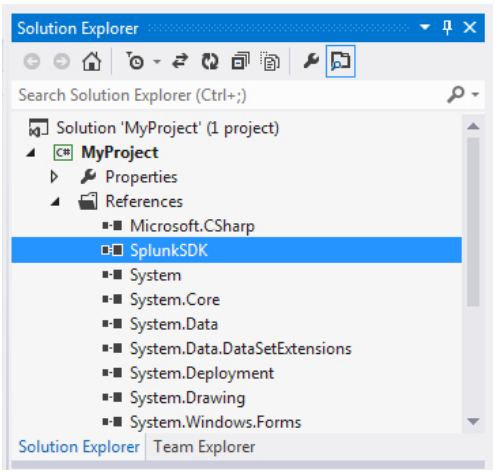


4. In the lower-right corner of the Reference Manager window, click the **Browse** button.
5. Navigate to the extracted Splunk SDK for C# folder, open it, and then double-click **SplunkSDK.dll**.

6. Verify that "SplunkSDK.dll" has been added to the **Browse** section of the Reference Manager window, and that its checkbox has been selected, and then click **OK**.



SplunkSDK now appears in the list of references in the Solution Explorer. You've installed the Splunk SDK for C#!



> ***Tip:*** *To browse the Splunk SDK for C# APIs, right-click* ***SplunkSDK*** *in the Solution Explorer, and then click* ***View in Object Browser****. Click the triangle next to* ***SplunkSDK*** *to view its namespaces. Click the triangle next to each namespace to view its classes.*

Build the examples

To view premade examples of the Splunk® SDK for C# in action, first install and build them:

1. Navigate to the extracted Splunk SDK for C# folder, open it, and then double-click the **splunk-sdk-csharp** folder.
2. Double-click **SplunkSDK.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. For more information about the unit tests, see Unit Tests.

Notes:

- Upon building the SDK, you might see error messages that mention Microsoft SharePoint. These apply to the **sharepoint_web_part** example, and can be safely ignored if you do not plan to try that example. If you want to try the **sharepoint_web_part** example, be aware that you must have Microsoft SharePoint Foundation 2010, SharePoint Server 2010, or later installed on the same computer on which you're building the SDK. In addition, the **sharepoint_web_part** example cannot be run at the command line; it is intended exclusively for the creation of a SharePoint Web Part. For more information, see the README file in the **sharepoint_web_part** directory.
- 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. It's up to you.

Save your connection info (optional)

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 text file and name it **.splunkrc**.

   Windows might display an error when you try to name the file because ".splunkrc" looks like a nameless file with just an extension. You can use the Command Prompt window to create this file by going to the C:\Users\currentusername directory and entering the following at the prompt:

   ```
   Notepad.exe .splunkrc
   ```

A dialog box appears, asking whether you want to create a new file. Click **Yes**, and then continue creating the file.

2. In the new file, paste in the following. Update any lines that contain information that differs from your Splunk Enterprise instance.

```
# Splunk host (default: localhost)
host=localhost
# Splunk admin port (default: 8089)
port=8089
# Splunk username
username=admin
# Splunk password
password=changeme
# Access scheme (default: https)
scheme=https
```

3. Save the file in the current user's home directory—for instance:

```
C:\Users\currentusername\.splunkrc
```

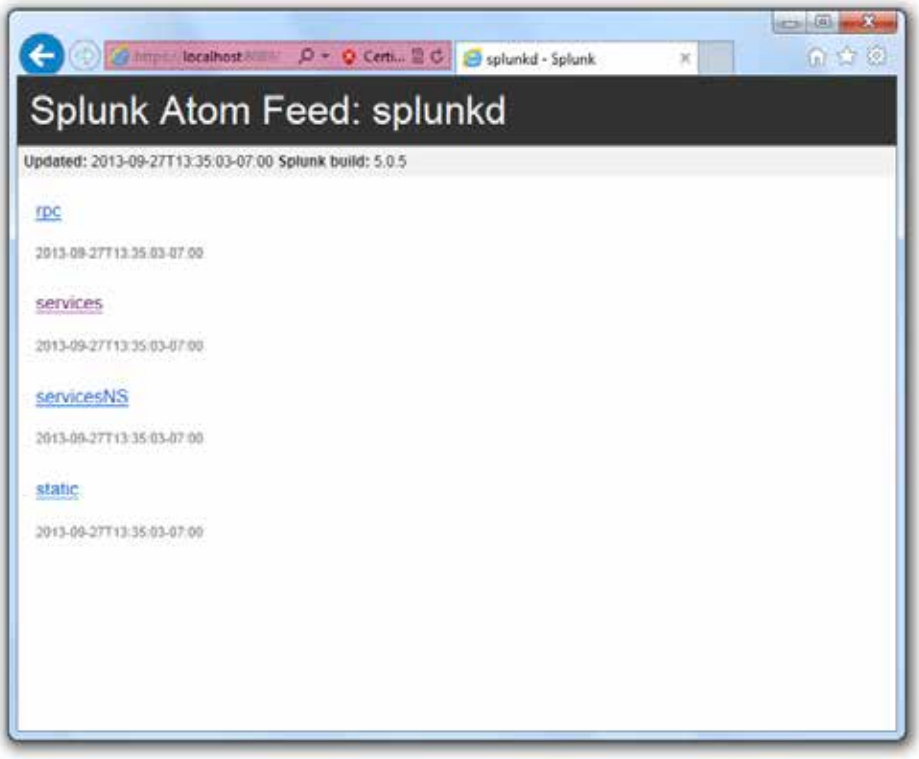
The examples and unit tests are ready to run with no additional connection arguments required!

Troubleshooting

If you're still not able to successfully connect to your Splunk Enterprise instance, or you're not seeing the correct information (for instance, search reports (saved searches in Splunk Enterprise 5) are wrong or aren't appearing), there are several possibilities for what's wrong:

- *Double-check the host and port in **.splunkrc***. Is the hostname correct? Is the port number correct? By default, the port for splunkd is 8089, and splunkd is what the Splunk SDK for C# is connecting to.
- *Your permissions are incorrect or insufficient.* If your permissions are incorrect or insufficient, you might be preventing the Splunk SDK for C# from seeing the search reports you're expecting. Try the following:
  1. Using the values for **scheme**, **host**, and **port** that you entered into the **.splunkrc** file, construct a URL as follows in the address bar of a new browser window: *scheme://host:port*
  2. Press Enter. If you see a window warning you about a problem with a security certificate, choose the option to continue. Assuming the URL is correct, you are presented with a window that looks like the

following:



3. Click the link for "services".
4. When prompted, enter the same credentials that you entered into the **.splunkrc** file.

   If, after clicking **OK**, you see an expanded list of endpoints, then the credentials you entered are incorrect. Re-enter your credentials, or check with your Splunk Enterprise administrator.

5. In the list of services, click the link for "saved".
6. Click "searches". If the "savedsearch" page that appears is empty, that means there are either no global search reports within your Splunk Enterprise instance, or no search reports in the default app of the current user.

If you don't see the search reports you expected, you'll need to change their permissions when logged in as someone with sufficient permissions, or create new ones.

Examples

The Splunk® SDK for C# provides a number of examples that show how to interact with the core Splunk SDK for C# library. 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\examples** directory. When you build the SDK, the examples are built as well.

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

Example	Description
authenticate	Authenticates to the server and prints the received token.
list_apps	Lists the apps installed on the server.
modular_input	Creates modular inputs programmatically.
search	Runs a normal search using a specified search query.
search_oneshot	Runs a oneshot search using a specified search query.
search_realtime	Runs a real-time search, gets a number of snapshots, prints them, and then exits.
sharepoint_web_part	Shows you how to create a Splunk Web Part for Microsoft SharePoint.
submit	Submits events into Splunk.

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. Specific instructions are included on each sample's topic page.

authenticate

The **authenticate** example authenticates to the Splunk Enterprise instance and prints the received token. To run the **authenticate** example from the command line:

1. At the command prompt, go to the **Debug** directory of the built example, for instance:

   ```
   \splunk-sdk-csharp\examples\authenticate\bin\Debug
   ```

2. Run the example by typing:

   ```
   authenticate.exe
   ```

   The token is retrieved and printed to the console.

> ***Note:*** *These instructions assume that you have already configured a* ***.splunkrc*** *file with your connection information, or that you're connecting to Splunk Enterprise with default connection values. (To see the default connection values, see Step 2 in Save your connection info.) If not, you will need to specify any non-default* ***host***, ***port***, ***username***, ***password***, *and* ***scheme*** *values as command line arguments. For example, if your*

> *username is someguy and your password is xyz123 (and this information is not specified in **.splunkrc**), for this example you would type the following at the command prompt:*
>
> `authenticate.exe --username="someguy" --password="xyz123"`

list_apps

The **list_apps** example authenticates to the Splunk Enterprise instance, and then lists all of the apps available to the authenticated user. (Apps to which the user you logged in with doesn't have access are not listed.) To run the **list_appsexample** from the command line:

1. At the command prompt, go to the Debug directory of the built example, for instance:

   `\splunk-sdk-csharp\examples\authenticate\bin\Debug`

2. Run the example by typing:

   `list_apps`

   A list of apps is printed to the console.

> *Note: These instructions assume that you have already configured a **.splunkrc** file with your connection information, or that you're connecting to Splunk Enterprise with default connection values. (To see the default connection values, see Step 2 in Save your connection info.) If not, you will need to specify any non-default **host**, **port**, **username**, **password**, and **scheme** values as command line arguments. For example, if your username is someguy and your password is xyz123 (and this information is not specified in **.splunkrc**), for this example you would type the following at the command prompt:*
>
> `authenticate.exe --username="someguy" --password="xyz123"`

modular_input

The **modular_input** example programmatically creates a modular input in Splunk Enterprise. It is described in detail in How to create modular inputs.

search, search_oneshot, and search_realtime

The **search**, **search_oneshot**, and **search_realtime** examples run a normal search, a oneshot search, and a realtime search, respectively. They require additional arguments when you run them from the command line.

With each of the search examples, you will need to provide search criteria in the following form, where `query_syntax` is a valid search query:

```
examplename.exe --search="query_syntax"
```

For instance, in the **Debug** directory of the built **search_oneshot** example, you can enter the following at the command prompt:

```
search_oneshot.exe --search="search sourcetype="access_combined_wcookie" 10.2.1.44 | head 10"
```

This runs a oneshot search for `sourcetype="access_combined_wcookie" 10.2.1.44` (the IP address 10.2.1.44 and a sourcetype of **access_combined_wcookie**) and prints the first ten results to the console window.

> ***Note:*** *These instructions assume that you have already configured a* ***.splunkrc*** *file with your connection information, or that you're connecting to Splunk Enterprise with default connection values. (To see the default connection values, see Step 2 in Save your connection info.) If not, you will need to specify any non-default* ***host***, ***port***, ***username***, ***password***, *and* ***scheme*** *values as command line arguments. For example, if your username is* someguy *and your password is* xyz123 *(and this information is not specified in* ***.splunkrc****), for this example you would type the following at the command prompt:*
>
> ```
> authenticate.exe --username="someguy" --password="xyz123"
> ```

sharepoint_web_part

The **sharepoint_web_part** example shows you how to create a Microsoft SharePoint Web Part to display data from Splunk® Enterprise. To run this example, ensure you have the following:

- SharePoint Foundation 2010, SharePoint Server 2010, or later installed on the computer on which you'll be running the example
- A correctly configured **.splunkrc** file

This example cannot be run via the command line, since it is intended to be run inside SharePoint.

To install the example Web Part:

1. Deploy the project. In Visual Studio with the SDK project open, on the **BUILD** menu, click **Deploy Solution**.
2. In SharePoint, edit a page and insert the custom Web Part called **IndexSummaryWebPart**.

submit

The **submit** example submits events to the Splunk® Enterprise instance's default index. After you run the **submit** example, you'll find two new events in your dashboard, "Hello World." and "Goodbye world.", with `source=splunk-sdk-tests` and `sourcetype=splunk-sdk-test-event`.

To run the **submit** example from the command line:

1. At the command prompt, go to the **Debug** directory of the built example, for instance:

   ```
   \splunk-sdk-csharp\examples\submit\bin\Debug
   ```

2. Run the example by typing:

   ```
   submit.exe
   ```

To find the events that were just added, go to Splunk Home (*Splunk Web* in Splunk Enterprise 5) and search the default index for **source="splunk-sdk-tests"**.

Unit tests

A great place to look for examples of how to use the Splunk® SDK for C# is in the unit tests. These are the same tests that we used to validate the core SDK library. The unit tests are located in the **\splunk-sdk-csharp\UnitTests** directory. When you build the SDK, the unit tests are built as well.

Before you run the unit tests, you must specify the test settings file to use:

1. On the **TEST** menu, point to **Test Settings**, and then click **Select Test Settings File**.
2. In the **Open Settings File** dialog box, navigate to the root folder of the Splunk SDK for C#.
3. Select the file named "Local.runsettings", and then click **Open**.

To run the unit tests in Visual Studio:

- On the **TEST** menu, point to **Run**, and then choose the option you want.

Alternately, you can use the Test Explorer in Visual Studio:

1. Click the **TEST** menu, point to **Windows**, and then click **Test Explorer**.
2. In the Test Explorer pane, click **Run All** to run all the tests, or choose a different option from the Run menu.

For more information about running unit tests, see the topic Running Unit Tests with Test Explorer on the Microsoft Developer Network (MSDN).

splunk>

How to use the Splunk SDK for C#

This section shows how to do the basics in Splunk®. We're assuming you already followed the instructions in the Getting Started section and were able to run the examples. We're also assuming you know your way around Splunk Web and got your feet wet; you've added some data and saved a search or two. If so, you're ready to start using the SDK to develop Splunk applications.

If you try the examples that are included with the SDK, you'll notice that they are all written from a command-line perspective. That is, they start by parsing the parameters that are provided at the command line and the parameters that are defined in the **.splunkrc** file (the optional file that stores your login credentials for convenience when running the SDK examples, described in Save your connection info).

For simplicity, the code examples in this section avoid error handling, command-line processing, and complex logic, staying focused simply on showing you how to use the SDK APIs.

How to...

Connect to Splunk	Introduces the basic process of connecting to splunkd and logging in.
Work with saved searches	Shows how to list, create, and run saved searches.
Work with searches and jobs	Shows how to list search jobs and run different types of searches.
Create modular inputs	Shows how to programmatically create modular inputs.
Display search results	Shows how to display the results from searches in different formats.

How to connect to Splunk

To start a Splunk® session, the first thing your program must do is connect to Splunk by sending login credentials to the splunkd server. Splunk returns an authentication token, which is then automatically included in subsequent calls for the rest of your session. By default, the token is valid for one hour, but is refreshed every time you make a call to splunkd.

The basic steps are as follows:

1. Import the Splunk library with a `using` statement.
2. Create an instance of the **ServiceArgs** class, specifying any of its properties. (At a minimum, you should specify **Host** and **Port**.)
3. Create a new **Service** object, using the **ServiceArgs** instance as its parameter.
4. Use the **Login** method, supplying the username and password.

> ***Important:*** *At this point, you should provide a mechanism to supply the login credentials for your Splunk server. In the example shown here, the login credentials are hard coded for convenience. Similarly, in the Splunk SDK for C# examples, the login credentials are stored locally or in a separate file. For security reasons, neither practice is recommended for your production app. Use whatever authentication mechanism you prefer (for instance, a login form) to supply the login credentials.*

Here's the example code to start a Splunk session. The hostname, port number, and credentials for the Splunk server are hard coded, so replace them with your own. This example also prints the received token to the console to verify you connected successfully.

Before you run this, start the Splunk server if you haven't already.

```
using Splunk;

/// <summary>
/// An example program to authenticate to the server
/// and print the received token.
/// </summary>
public class Program
{
    public static void Main()
    {
    // Create new ServiceArgs object to store
    // connection info
    var connectArgs = new ServiceArgs
    {
        Host = "localhost",
        Port = 8089
        // See ServiceArgs documentation for full
        // list of properties
    };

    // Create new Service object
    Service service = new Service(connectArgs);

    // Use the Login method to connect
    service.Login("admin", "changeme");

    // Print received token to verify login
    System.Console.WriteLine("Token: ");
    System.Console.WriteLine(" " + service.Token);
    }
}
```

For more sample code, see the **examples** directory of the SDK.

How to work with saved searches using the Splunk SDK for C#

The most fundamental feature in Splunk® is searching your data. But before diving into the details of how to use the SDK to search, let's clarify the terms:

- A *search query* is a set of commands and functions you use to retrieve events from an index or a real-time stream, for example:

```
search * | head 10
```

- A *saved search* is a search query that has been saved to be used again and can be set up to run on a regular schedule. The results from the search are not saved with the query.
- A *search job* is an instance of a completed or still-running search operation, along with the results. A search ID is returned when you create a job, allowing you to access the results of the search when they become available. Search results are returned in XML (the default), JSON, JSON_ROWS, JSON_COLS, Atom, or CSV format.

This topic focuses on working with saved searches. For more about working with search jobs, see How to work with searches and jobs.

The saved search APIs

The classes for working with saved searches are:

- The **SavedSearchCollection** class for retrieving the collection of saved searches.
- The **Service** class for an individual saved search.

Access these classes through an instance of the **Service** class. Retrieve a collection, and from there you can access individual items in the collection and create new ones. For example, here's a simplified program for getting a collection of saved searches and creating a new one:

```
// Connect to Splunk
Service service = Service.Connect(connectArgs);

// Retrieves the collection of saved searches
SavedSearchCollection savedSearches = service.GetSavedSearches();

// Creates a saved search
SavedSearch savedSearch = savedSearches.Create(name, query);

// Another way to create a saved search
// SavedSearch savedSearch = service.GetSavedSearches().Create(name, query);
```

Code examples

This section provides examples of how to use the search APIs, assuming you first connect to a Splunk instance:

- To list saved searches
- To view the history of a saved search
- To create a saved search
- To view and modify the properties of a saved search
- To run a saved search
- To run a saved search with run-time arguments
- To delete a saved search

The following parameters are available for saved searches:

- Collection parameters
- Saved search parameters

To list saved searches

This example shows how to retrieve and list the saved searches in the saved search collection. If you don't explicitly specify a namespace, the current one is used.

```
// List all saved searches for the current namespace
SavedSearchCollection savedSearches = service.GetSavedSearches();
System.Console.WriteLine(savedSearches.Size + " saved searches are available to the
current user:\n");
foreach (SavedSearch entity in savedSearches.Values) {
    System.Console.WriteLine("     " + entity.Name);
}
```

To retrieve a collection for a specific namespace–for example, to list the saved searches available to a specific username–provide the namespace as arguments to the **Service.GetSavedSearches** method.

This example retrieves the collection of users and lists the saved searches for the last username in the collection (just to show a user other than "admin"):

```
// Get the collection of users and save the name of the last user
UserCollection users = service.GetUsers();
String lastUser = null;
foreach (User user in users.Values)
{
    lastUser = user.Name;
}

// Specify a namespace using the name of the last user
ServiceArgs ns = new ServiceArgs();
ns.App = "search";
ns.Owner = lastUser;
SavedSearchCollection savedSearches2 = service.GetSavedSearches(ns);

System.Console.WriteLine(savedSearches2.Size + " saved searches are available to '" +
lastUser + "':\n");
foreach (SavedSearch search in savedSearches2.Values)
{
    System.Console.WriteLine("     " + search.Name);
}
```

To view the history of a saved search

The history of a saved search contains the past and current instances (jobs) of the search. This example shows the history for all the saved searches in the current collection:

```
// Retrieve the collection of saved searches
SavedSearchCollection savedSearches = service.GetSavedSearches();

// Iterate through the collection of saved searches and display the history for each
one
foreach (SavedSearch entity in savedSearches.Values)
{
    Job[] sHistory = entity.History();
    System.Console.WriteLine("\n" + sHistory.Length + " jobs for the '" + entity.Name
+ "' saved search");
    for (int i = 0; i < sHistory.Length; ++i)
    {
    System.Console.WriteLine("     " + sHistory[i].EventCount + " events for Search ID
" + sHistory[i].Sid + "\n");
    }
}
```

To create a saved search

When you create a saved search, at a minimum you need to provide a search query and a name for the search. Then you have a couple of options for how to set properties for the saved search:

- **Use the object properties.** The object properties are the easiest way to set and modify the object's parameters, but they aren't available until after the saved search has been created. See the next section for more about modifying a saved search.
- **Create an argument dictionary of key-value pairs.** Creating an argument dictionary is the only way to set properties at the same time you create a saved search, but it requires a little more work to look up properties and provide values in the correct format. For a list of possible properties, see Saved search parameters.

This example shows how to create a simple saved search:

```
// Create a saved search by specifying a name and search query
// Note: Do not include the 'search' keyword for a saved search
String myQuery = "* | head 10";
String mySearchName = "Test Search";
SavedSearch savedSearch = service.GetSavedSearches().Create(mySearchName, myQuery);
System.Console.WriteLine("The search '" + savedSearch.Name +
    "' (" + savedSearch.Search + ") was saved");
```

To view and modify the properties of a saved search

This example shows how to view the properties of the new saved search:

```
// Retrieve the search that was just created
SavedSearch savedSearch = service.GetSavedSearches().Get("Test Search");

// Display some properties of the new search
System.Console.WriteLine("Properties for '" + savedSearch.Name + "':\n\n" +
    "Description:     " + savedSearch.Description + "\n" +
    "Scheduled:       " + savedSearch.IsScheduled + "\n" +
    "Next scheduled time: " + savedSearch.NextScheduledTime + "\n"
);
```

To set properties, use the **SavedSearch** properties.

This example shows how to set the description and schedule the saved search in cron format:

```
// Set the properties and schedule
savedSearch.Description = "This is a test search";
savedSearch.IsScheduled = true;
savedSearch.CronSchedule = "15 4 * * 6";

// Display the updated properties
System.Console.WriteLine("New properties for '" + savedSearch.Name + "':\n\n" +
    "Description:     " + savedSearch.Description + "\n" +
    "Scheduled:       " + savedSearch.IsScheduled + "\n" +
    "Next scheduled time: " + savedSearch.NextScheduledTime + "\n"
    );
```

To run a saved search

Running a saved search creates a search job that is scheduled to run right away. Use the **SavedSearch.Dispatch** method to run a saved search, which returns a **Job** object that corresponds to the search job. The Job object gives you access to information about the search job, such as the search ID, the status of the search, and the search results once the search job has finished.

The **Dispatch** method takes these optional parameters:

- **dispatch.now**: A time string that is used to dispatch the search as though the specified time were the current time.
- **dispatch.***: Overwrites the value of the search field specified in *.
- **trigger_actions**: A Boolean that indicates whether to trigger alert actions.
- **force_dispatch**: A Boolean that indicates whether to start a new search if another instance of this search is already running.

You can set these properties using the generic **Args** class and passing these parameters as key-value pairs.

This example runs the search that was created in the previous example and shows how to poll the status to determine when the search has completed:

```
// Retrieve the new saved search
SavedSearch savedSearch = service.GetSavedSearches().Get("Test Search");
```

```
// Run a saved search and poll for completion
System.Console.WriteLine("Run the '" + savedSearch.Name + "' search ("
    + savedSearch.Search + ")\n");
Job jobSavedSearch = null;

// Run the saved search
try
{
    jobSavedSearch = savedSearch.Dispatch();
}
catch (ExecutionEngineException e1)
{
    System.Console.WriteLine(e1.StackTrace);
}

System.Console.WriteLine("Waiting for the job to finish...\n");

// Wait for the job to finish
while (!jobSavedSearch.IsDone)
{
    try
    {
    Thread.Sleep(500);
    }
    catch (ExecutionEngineException e)
    {
    // TODO Auto-generated catch block
    System.Console.WriteLine(e.StackTrace);
    }
}
```

Once the search has finished, retrieve the search results from the **Job** object. For more, see How to display search results.

To run a saved search with run-time arguments

If you want to save a search containing a search keyword or field value that will not be known until you actually run the saved search, you can use a variable in the saved search query. Then when you run the saved search, specify the value of that variable in an argument dictionary (create it using the **SavedSearchDispatchArgs** class) and pass that dictionary to the **SavedSearch.Dispatch(*args*)** method.

You can also specify additional search attributes at run time (such as time modifiers) as dispatch arguments. Some of these attributes can be set using properties of the **SavedSearchDispatchArgs** class. If the attribute you want to set doesn't have a property, just set the attribute as a key-value pair in the dispatch argument dictionary.

The example below shows how to set run-time variables, starting with a simple query for a saved search:

```
search index=_internal sourcetype=splunkd_access
```

But let's say you won't know which source type to search on until run time. In that case, you can add a variable "mysourcetype" in the format `$args.variablename$` (the variable name must be prefixed with "args."). The search query becomes:

```
search index=_internal sourcetype=$args.mysourcetype$
```

After saving the search, the example creates an argument dictionary using a mix of **SavedSearchDispatchArgs** properties and key-value pairs. Then, the saved search is run using the **SavedSearch.Dispatch(*args*)** method.

```
String myQuery = "index=_internal sourcetype=$args.mysourcetype$";
String mySearchName = "Test Search Args";
SavedSearch savedSearch = service.GetSavedSearches().Create(mySearchName, myQuery);

// Set the arguments for dispatching the saved search
SavedSearchDispatchArgs dispatchArgs = new SavedSearchDispatchArgs();

// These attributes have properties
dispatchArgs.DispatchEarliestTime = "-20m@m";
dispatchArgs.DispatchLatestTime = "now";

// This attribute is set using a key-value pair
dispatchArgs.Add("span", "5min");

// The value of the field variable "mysourcetype" is also set as a key-value pair
dispatchArgs.Add("args.mysourcetype", "splunkd");

// Run the saved search with the dispatch arguments
Job job = null;
try
{
    job = savedSearch.Dispatch(dispatchArgs);
}
    catch (ThreadInterruptedException e1)
{
    // TODO Auto-generated catch block
    System.Console.WriteLine(e1.StackTrace);
}

System.Console.WriteLine("Waiting for the job to finish...");

// Wait for the job to finish
while (!job.IsDone)
{
    try
    {
    Thread.Sleep(500);
    }
    catch (ThreadInterruptedException e)
    {
    // TODO Auto-generated catch block
    System.Console.WriteLine(e.StackTrace);
    }
    System.Console.WriteLine("Done! The job finished with " + job.EventCount + "
events.");
}
```

To delete a saved search

You can delete a saved search using the **Entity.Remove** method. Any jobs for the saved search are not deleted.

This example shows how to delete a saved search using the **Entity.Remove** method:

```
// Retrieve a saved search
SavedSearch savedSearch2 = service.GetSavedSearches().Get("Test Search");

// Delete the saved search
savedSearch2.Remove();
```

Collection parameters

The parameters below are available when retrieving a collection of saved searches.

By default, all entities are returned when you retrieve a collection. Using the parameters below, you can specify the number of entities to return, how to sort them, and so on. Set these parameters by creating a generic **Args** dictionary:

Parameter	Description
`count`	A number that indicates the maximum number of entries to return. A value of 0 means all entries are returned.
`earliest_time`	A string that contains all the scheduled times starting from this time (not just the next run time).
`latest_time`	A string that contains all the scheduled times until this time.
`offset`	A number that specifies the index of the first item to return. For oneshot inputs, this value refers to the current position in the source file, indicating how much of the file has been read.
`search`	A string that specifies a search expression to filter the response with, matching field values against the search expression. For example, "search=foo" matches any object that has "foo" as a substring in a field, and "search=field_name%3Dfield_value" restricts the match to a single field.
`sort_dir`	An enum value that specifies how to sort entries. Valid values are "asc" (ascending order) and "desc" (descending order).
`sort_key`	A string that specifies the field to sort by.
`sort_mode`	An enum value that specifies how to sort entries. Valid values are "auto", "alpha" (alphabetically), "alpha_case" (alphabetically, case sensitive), or "num" (numerically).

Saved search parameters

The properties that are available for saved searches correspond to the parameters for the saved/searches endpoint in the REST API.

This table summarizes the properties you can set for a saved search. While you can use class properties to modify existing properties, you can only set them at the same time that you create a saved search by providing a dictionary of property key-value pairs.

This example shows how to do that by setting a "description" property while creating a saved search:

```
// Create an argument dictionary with properties for a new saved search
Args savedSearchArgs = new Args();
savedSearchArgs.Add("description", "This is my test search");
SavedSearch savedSearch = service.GetSavedSearches().Create("My Test Search",
    "search * | head 5", savedSearchArgs);
```

Parameter	Description
`name`	Required. A string that contains the name of the saved search.
`search`	Required. A string that contains the search query.

action.*	A string with wildcard arguments to specify specific action arguments.
action.email	A Boolean that indicates the state of the email alert action. Read only.
action.email.auth_password	A string that specifies the password to use when authenticating with the SMTP server. Normally this value is set while editing the email settings, but you can set a clear text password here that is encrypted when Splunk is restarted.
action.email.auth_username	A string that specifies the username to use when authenticating with the SMTP server. If this is empty string, authentication is not attempted.
action.email.bcc	A string that specifies the BCC email address to use if "action.email" is enabled.
action.email.cc	A string that specifies the CC email address to use if "action.email" is enabled.
action.email.command	A string that contains the search command (or pipeline) for running the action.
action.email.format	An enum value that indicates the format of text and attachments in the email ("plain", "html", "raw", or "csv"). Use "plain" for plain text.
action.email.from	A string that specifies the email sender's address.
action.email.hostname	A string that specifies the hostname used in the web link (URL) that is sent in email alerts. Valid forms are "hostname" and "protocol://hostname:port".
action.email.inline	A Boolean that indicates whether the search results are contained in the body of the email.
action.email.mailserver	A string that specifies the address of the MTA server to be used to send the emails.
action.email.maxresults	The maximum number of search results to send when "action.email" is enabled.
action.email.maxtime	A number indicating the maximum amount of time an email action takes before the action is canceled. The valid format is number followed by a time unit ("s", "m", "h", or "d"), for example "5d".
action.email.pdfview	A string that specifies the name of the view to deliver if "action.email.sendpdf" is enabled.
action.email.preprocess_results	A string that specifies how to pre-process results before emailing them.
action.email.reportCIDFontList	Members of an enumeration in a space-separated list specifying the set (and load order) of CID fonts for handling

	Simplified Chinese (gb), Traditional Chinese (cns), Japanese (jp), and Korean (kor) in Integrated PDF Rendering.
action.email.reportIncludeSplunkLogo	A Boolean that indicates whether to include the Splunk logo with the report.
action.email.reportPaperOrientation	An enum value that indicates the paper orientation ("portrait" or "landscape").
action.email.reportPaperSize	An enum value that indicates the paper size for PDFs ("letter", "legal", "ledger", "a2", "a3", "a4", or "a5").
action.email.reportServerEnabled	A Boolean that indicates whether the PDF server is enabled.
action.email.reportServerURL	A string that contains the URL of the PDF report server, if one is set up and available on the network.
action.email.sendpdf	A Boolean that indicates whether to create and send the results as a PDF.
action.email.sendresults	A Boolean that indicates whether to attach search results to the email.
action.email.subject	A string that specifies the subject line of the email.
action.email.to	A string that contains a comma- or semicolon-delimited list of recipient email addresses. Required if this search is scheduled and "action.email" is enabled.
action.email.track_alert	A Boolean that indicates whether running this email action results in a trackable alert.
action.email.ttl	The number of seconds indicating the minimum time-to-live (ttl) of search artifacts if this email action is triggered. If the value is a number followed by "p", it is the number of scheduled search periods.
action.email.use_ssl	A Boolean that indicates whether to use secure socket layer (SSL) when communicating with the SMTP server.
action.email.use_tls	A Boolean that indicates whether to use transport layer security (TLS) when communicating with the SMTP server.
action.email.width_sort_columns	A Boolean that indicates whether columns should be sorted from least wide to most wide, left to right. This value is only used when "action.email.format"="plain", indicating plain text.
action.populate_lookup	A Boolean that indicates the state of the populate-lookup alert action. Read only.
action.populate_lookup.command	A string that specifies the search command (or pipeline) to run the populate-lookup alert action.
action.populate_lookup.dest	A string that specifies the name of the lookup table or lookup path to populate.

action.populate_lookup.hostname	A string that specifies the host name used in the web link (URL) that is sent in populate-lookup alerts. Valid forms are "hostname" and "protocol://hostname:port".
action.populate_lookup.maxresults	The maximum number of search results to send in populate-lookup alerts.
action.populate_lookup.maxtime	The number indicating the maximum amount of time an alert action takes before the action is canceled. The valid format is number followed by a time unit ("s", "m", "h", or "d").
action.populate_lookup.track_alert	A Boolean that indicates whether running this populate-lookup action results in a trackable alert.
action.populate_lookup.ttl	The number of seconds indicating the minimum time-to-live (ttl) of search artifacts if this populate-lookup action is triggered. If the value is a number followed by "p", it is the number of scheduled search periods.
action.rss	A Boolean that indicates the state of the RSS alert action. Read only.
action.rss.command	A string that contains the search command (or pipeline) that runs the RSS alert action.
action.rss.hostname	A string that contains the host name used in the web link (URL) that is sent in RSS alerts. Valid forms are "hostname" and "protocol://hostname:port".
action.rss.maxresults	The maximum number of search results to send in RSS alerts.
action.rss.maxtime	The maximum amount of time an RSS alert action takes before the action is canceled. The valid format is number followed by a time unit ("s", "m", "h", or "d").
action.rss.track_alert	A Boolean that indicates whether running this RSS action results in a trackable alert.
action.rss.ttl	The number of seconds indicating the minimum time-to-live (ttl) of search artifacts if this RSS action is triggered. If the value is a number followed by "p", it is the number of scheduled search periods.
action.script	A Boolean that indicates the state of the script alert action. Read only.
action.script.command	A string that contains the search command (or pipeline) that runs the script action.
action.script.filename	A string that specifies the file name of the script to call, which is required if "action.script" is enabled.
action.script.hostname	A string that specifies the hostname used in the web link (URL) that is sent in script alerts. Valid forms are "hostname" and "protocol://hostname:port".

action.script.maxresults	The maximum number of search results to send in script alerts.
action.script.maxtime	The maximum amount of time a script action takes before the action is canceled. The valid format is number followed by a time unit ("s", "m", "h", or "d").
action.script.track_alert	A Boolean that indicates whether running this script action results in a trackable alert.
action.script.ttl	The number of seconds indicating the minimum time-to-live (ttl) of search artifacts if this script action is triggered. If the value is a number followed by "p", it is the number of scheduled search periods.
action.summary_index	A Boolean that indicates the state of the summary index alert action. Read only.
action.summary_index._name	A string that specifies the name of the summary index where the results of the scheduled search are saved.
action.summary_index.command	A string that contains the search command (or pipeline) that runs the summary-index action.
action.summary_index.hostname	A string that specifies the hostname used in the web link (URL) that is sent in summary-index alerts. Valid forms are "hostname" and "protocol://hostname:port".
action.summary_index.inline	A Boolean that indicates whether to run the summary indexing action as part of the scheduled search.
action.summary_index.maxresults	The maximum number of search results to send in summary-index alerts.
action.summary_index.maxtime	A number indicating the maximum amount of time a summary-index action takes before the action is canceled. The valid format is number followed by a time unit ("s", "m", "h", or "d"), for example "5d".
action.summary_index.track_alert	A Boolean that indicates whether running this summary-index action results in a trackable alert.
action.summary_index.ttl	The number of seconds indicating the minimum time-to-live (ttl) of search artifacts if this summary-index action is triggered. If the value is a number followed by "p", it is the number of scheduled search periods.
actions	A string that contains a comma-delimited list of actions to enable, for example "rss,email".
alert.digest_mode	A Boolean that indicates whether Splunk applies the alert actions to the entire result set or digest ("true"), or to each individual search result ("false").

alert.expires	The amount of time to show the alert in the dashboard. The valid format is numberfollowed by a time unit ("s", "m", "h", or "d").
alert.severity	A number that indicates the alert severity level (1=DEBUG, 2=INFO, 3=WARN, 4=ERROR, 5=SEVERE, 6=FATAL).
alert.suppress	A Boolean that indicates whether alert suppression is enabled for this scheduled search.
alert.suppress.fields	A string that contains a comma-delimited list of fields to use for alert suppression.
alert.suppress.period	A value that indicates the alert suppression period, which is only valid when "Alert.Suppress" is enabled. The valid format is number followed by a time unit ("s", "m", "h", or "d").
alert.track	An enum value that indicates how to track the actions triggered by this saved search. Valid values are: "true" (enabled), "false" (disabled), and "auto" (tracking is based on the setting of each action).
alert_comparator	A string that contains the alert comparator. Valid values are: "greater than", "less than", "equal to", "rises by", "drops by", "rises by perc", and "drops by perc".
alert_condition	A string that contains a conditional search that is evaluated against the results of the saved search.
alert_threshold	A value to compare to before triggering the alert action. Valid values are: integer or integer%. If this value is expressed as a percentage, it indicates the value to use when "alert_comparator" is set to "rises by perc" or "drops by perc".
alert_type	A string that indicates what to base the alert on. Valid values are: "always", "custom", "number of events", "number of hosts", and "number of sources". This value is overridden by "alert_condition" if specified.
args.*	A string containing wildcard arguments for any saved search template argument, such as "args.username"="foobar" when the search is search $username$.
auto_summarize	A Boolean that indicates whether the scheduler ensures that the data for this search is automatically summarized.
auto_summarize.command	A string that contains a search template that constructs the auto summarization for this search.
auto_summarize.cron_schedule	A string that contains the cron schedule for probing and generating the summaries for this saved search.
auto_summarize.dispatch.earliest_time	A string that specifies the earliest time for summarizing this saved search. The time can be relative or absolute; if absolute, use the "dispatch.time_format" parameter to format the value.

auto_summarize.dispatch.latest_time	A string that contains the latest time for summarizing this saved search. The time can be relative or absolute; if absolute, use the "dispatch.time_format" parameter to format the value.
auto_summarize.dispatch.ttl	The number of seconds indicating the time to live (in seconds) for the artifacts of the summarization of the scheduled search. If the value is a number followed by "p", it is the number of scheduled search periods.
auto_summarize.max_disabled_buckets	A number that specifies the maximum number of buckets with the suspended summarization before the summarization search is completely stopped, and the summarization of the search is suspended for the "auto_summarize.suspend_period" parameter.
auto_summarize.max_summary_ratio	A number that specifies the maximum ratio of summary size to bucket size, which specifies when to stop summarization and deem it unhelpful for a bucket. The test is only performed if the summary size is larger than the value of "auto_summarize.max_summary_size".
auto_summarize.max_summary_size	A number that specifies the minimum summary size, in bytes, before testing whether the summarization is helpful.
auto_summarize.max_time	A number that specifies the maximum time (in seconds) that the summary search is allowed to run. Note that this is an approximate time because the summary search stops at clean bucket boundaries.
auto_summarize.suspend_period	A string that contains the time indicating when to suspend summarization of this search if the summarization is deemed unhelpful.
auto_summarize.timespan	A string that contains a comma-delimited list of time ranges that each summarized chunk should span. This comprises the list of available granularity levels for which summaries would be available.
cron_schedule	A string that contains the cron-style schedule for running this saved search.
description	A string that contains a description of this saved search.
disabled	A Boolean that indicates whether the saved search is enabled.
dispatch.*	A string that specifies wildcard arguments for any dispatch-related argument.
dispatch.buckets	The maximum number of timeline buckets.
dispatch.earliest_time	A time string that specifies the earliest time for this search. Can be a relative or absolute time. If this value is an absolute time, use "dispatch.time_format" to format the value.

dispatch.latest_time	A time string that specifies the latest time for this saved search. Can be a relative or absolute time. If this value is an absolute time, use "dispatch.time_format" to format the value.
dispatch.lookups	A Boolean that indicates whether lookups for this search are enabled.
dispatch.max_count	The maximum number of results before finalizing the search.
dispatch.max_time	The maximum amount of time (in seconds) before finalizing the search.
dispatch.reduce_freq	The number of seconds indicating how frequently Splunk runs the MapReduce reduce phase on accumulated map values.
dispatch.rt_backfill	A Boolean that indicates whether to back fill the real-time window for this search. This value is only used for a real-time search.
dispatch.spawn_process	A Boolean that indicates whether Splunk spawns a new search process when running this saved search.
dispatch.time_format	A string that defines the time format that Splunk uses to specify the earliest and latest time.
dispatch.ttl	The number indicating the time to live (ttl) for artifacts of the scheduled search (the time before the search job expires and artifacts are still available), if no alerts are triggered. If the value is a number followed by "p", it is the number of scheduled search periods.
displayview	A string that contains the default UI view name (not label) in which to load the results.
is_scheduled	A Boolean that indicates whether this saved search runs on a schedule.
is_visible	A Boolean that indicates whether this saved search is visible in the saved search list.
max_concurrent	The maximum number of concurrent instances of this search the scheduler is allowed to run.
next_scheduled_time	A string that indicates the next scheduled time for this saved search. Read only.
qualifiedSearch	A string that is computed during run time. Read only.
realtime_schedule	A Boolean that specifies how the scheduler computes the next time a scheduled search is run: • When "true": The schedule is based on the current time. The scheduler might skip some scheduled periods to make sure that searches over the most recent time range are run.

	• When "false": The schedule is based on the last search run time (referred to as "continuous scheduling") and the scheduler never skips scheduled periods. However, the scheduler might fall behind depending on its load. Use continuous scheduling whenever you enable the summary index option ("action.summary_index"). The scheduler tries to run searches that have real-time schedules enabled before running searches that have continuous scheduling enabled.
`request.ui_dispatch_app`	A string that contains the name of the app in which Splunk Web dispatches this search.
`request.ui_dispatch_view`	A string that contains the name of the view in which Splunk Web dispatches this search.
`restart_on_searchpeer_add`	A Boolean that indicates whether a real-time search managed by the scheduler is restarted when a search peer becomes available for this saved search. The peer can be one that is newly added or one that has become available after being down.
`run_on_startup`	A Boolean that indicates whether this search is run when Splunk starts. If the search is not run on startup, it runs at the next scheduled time. It is recommended that you set this value to "true" for scheduled searches that populate lookup tables.
`vsid`	A string that contains the view state ID that is associated with the view specified in the "displayview" attribute.

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

Searches run in different modes, determining when and how you can retrieve results:

- **Normal:** A normal search runs asynchronously. It returns a search job immediately. Poll the job to determine its status. You can retrieve the results when the search has finished. You can also preview the results if "preview" is enabled. Normal mode works with real-time searches.
- **Blocking:** 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 mode doesn't work with real-time searches.
- **Oneshot:** A oneshot search is a blocking search that is scheduled to run immediately. Instead of returning a search job, this mode returns the results of the search once completed. Because this is a blocking search, the results are not available until the search has finished.
- **Realtime:** A real-time search runs in normal mode 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.
- **Export:** An export search runs immediately, does not create a job for the search, and starts streaming results immediately. This search is useful for exporting large amounts of data from Splunk.

For those searches that produce search jobs (normal, blocking, and realtime), the search results are saved for a period of time on the server and can be retrieved on request. For those searches that stream the results (oneshot and export), the search results are not retained on the server. If the stream is interrupted for any reason, the results are not recoverable without running the search again.

The job APIs

The classes for working with jobs are:

- The **JobCollection** class for the collection of search jobs.
- The **Job** class for an individual search job.
- The **JobArgs** class with arguments for creating jobs.
- The **JobEventsArgs** class with arguments for retrieving events from a job.
- The **JobExportArgs** class with arguments for creating an export search.
- The **JobResultsArgs** class with arguments for retrieving results from a job.
- The **JobResultsPreviewArgs** class with arguments for retrieving preview results from a job.

Access these classes through an instance of the **Service** class. Retrieve a collection, and from there you can access individual items in the collection and create new ones. For example, here's a simplified program for getting a collection of jobs and creating a new one:

```
// Connect to Splunk
Service service = Service.Connect(connectArgs);

// Retrieves the collection of search jobs
JobCollection jobs = service.GetJobs();

// Creates a search job
Job job = jobs.Create(query);

// Another way to create a search job
// Job job = service.GetJobs().Create(query);
```

The methods for running other searches are available from the **Service** class (rather than the **Job** class) because these searches don't create search jobs:

- The **Service.Oneshot** method for oneshot searches.
- The **Service.Export** method for export searches.

Code examples

This section provides examples of how to use the job APIs, assuming you first connect to a Splunk instance:

- To list search jobs for the current user
- To run a normal search and poll for completion
- To run a blocking search and display properties of the job
- To run a basic oneshot search and display results
- To run a real-time search
- To run an export search

> ***Note:*** *This topic touches on displaying search results, but for more in-depth information, see How to display search results.*

The following parameters are available for search jobs:

- Collection parameters
- Search job parameters (properties to set)
- Search job parameters (properties to retrieve)

To list search jobs for the current user

This example gets the collection of jobs available to the current user:

```
// Retrieve the collection
JobCollection jobs = service.GetJobs();
System.Console.WriteLine("There are " + jobs.Size + " jobs available to 'admin'\n");

// List the job SIDs
foreach (Job job in jobs.Values) {
    System.Console.WriteLine(job.Name);
}
```

To run a normal search and poll for completion

Running a normal search creates a search job and immediately returns the search ID, so you need to poll the job to find out when the search has finished.

When you create a search job, you need to set the parameters of the job as an argument map of key-value pairs. For a list of all the possible parameters, see Search job parameters.

This example runs a normal search, waits for the job to finish, and then displays the results along with some statistics:

```
// Run a normal search
String searchQuery_normal = "search * | head 100";
JobArgs jobArgs = new JobArgs();
jobArgs.ExecutionMode = JobArgs.ExecutionModeEnum.Normal;
Job job = service.GetJobs().Create(searchQuery_normal, jobArgs);

// Wait for the search to finish
while (!job.IsDone) {
    try {
    Thread.Sleep(500);
    } catch (ThreadInterruptedException e) {
    // TODO Auto-generated catch block
    System.Console.WriteLine(e.StackTrace);
    }
}

// Get the search results and use the built-in XML parser to display them
var outArgs = new JobResultsArgs
{
    OutputMode = JobResultsArgs.OutputModeEnum.Xml,
    Count = 0 // Return all entries.
};

using (var stream = job.Results(outArgs))
{
    using (var rr = new ResultsReaderXml(stream))
    {
    foreach (var @event in rr)
    {
        System.Console.WriteLine("EVENT:");
        foreach (string key in @event.Keys)
        {
        System.Console.WriteLine("   " + key + " -> " + @event[key]);
```

```
        }
    }
    }
}

// Get properties of the job
System.Console.WriteLine("Search job properties:\n---------------------");
System.Console.WriteLine("Search job ID:     " + job.Sid);
System.Console.WriteLine("The number of events:  " + job.EventCount);
System.Console.WriteLine("The number of results: " + job.ResultCount);
System.Console.WriteLine("Search duration:       " + job.RunDuration + " seconds");
System.Console.WriteLine("This job expires in:   " + job.Ttl + " seconds");
```

To run a blocking search and display properties of the job

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 results are in.

When you create a search job, you need to set the parameters of the job as an argument dictionary of key-value pairs. For a list of all the possible parameters, see Search job parameters.

This example runs a blocking search, waits for the job to finish, and then displays some statistics:

```
// Run a blocking search
String searchQuery_blocking = "search * | head 100"; // Return the first 100 events
JobArgs jobArgs = new JobArgs();
jobArgs.ExecutionMode = JobArgs.ExecutionModeEnum.Normal;

// A blocking search returns the job when the search is done
System.Console.WriteLine("Wait for the search to finish...");
Job job = service.GetJobs().Create(searchQuery_blocking, jobArgs);
System.Console.WriteLine("...done!\n");

// Get properties of the job
System.Console.WriteLine("Search job properties:\n---------------------");
System.Console.WriteLine("Search job ID:     " + job.Sid);
System.Console.WriteLine("The number of events:  " + job.EventCount);
System.Console.WriteLine("The number of results: " + job.ResultCount);
System.Console.WriteLine("Search duration:       " + job.RunDuration + " seconds");
System.Console.WriteLine("This job expires in:   " + job.Ttl + " seconds");
```

To run a basic oneshot search and display results

Unlike other searches, the oneshot search does not create a search job, so you can't access it using the Job and JobCollection classes. Instead, use the **Service.Oneshot** method. To set properties for the search (for example, to specify a time range to search), you'll need to create an argument map with the parameter key-value pairs. Some common parameters are:

- **output_mode**: Specifies the output format of the results (XML, JSON, JSON_ROWS, JSON_COLS, Atom, or CSV).
- **earliest_time**: 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.
- **latest_time**: 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 list of Search job parameters, although most of these parameters don't apply to a oneshot search.

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

> ***Note:*** *If you don't see any search results with this example, you might not have anything in the specified time range. Just modify the date and time as needed for your data set.*

```
// Set the parameters for the search:
var oneshotSearchArgs = new JobArgs();
oneshotSearchArgs.EarliestTime = "2012-06-19T12:00:00.000-07:00";
oneshotSearchArgs.LatestTime = "2012-06-20T12:00:00.000-07:00";
String oneshotSearchQuery = "search * | head 10";

var outArgs = new JobResultsArgs
{
    OutputMode = JobResultsArgs.OutputModeEnum.Xml,

    // Return all entries.
    Count = 0,
};

using (var stream = service.Oneshot(
    oneshotSearchQuery, outArgs))
{
    using (var rr = new ResultsReaderXml(stream))
    {
    foreach (var @event in rr)
    {
        System.Console.WriteLine("EVENT:");
        foreach (string key in @event.Keys)
        {
        System.Console.WriteLine(
            "   " + key + " -> " + @event[key]);
        }
    }
    }
}
```

To run a real-time search

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 ("`exec_mode`") to "`normal`".
- Set the search mode ("`search_mode`") to "`realtime`", which also enables previews.
- Set the earliest and latest times to search ("`earliest_time`" and "`latest_time`") to "`rt`". (Setting these to "`rt`" also sets the search mode to "`realtime`".)
  If you want to specify a sliding window for your search (let's say a one-minute window), you can use relative time modifiers (for example, set "`earliest_time`" to "`rt-1m`", "`latest_time`" 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.

Displaying the real-time search results requires displaying preview results (for more information, see **JobResultsPreviewArgs** class):

- By default, the most recent 100 previews are retrieved. To change this number, set a value for "`count`". Use the "`offset`" value to page through large sets of previews.
- Only previews from the time range to search are retrieved.
- Depending on the time range to search, the number of events that are arriving to be indexed, and the count of previews to retrieve, the previews from one set to the next might include duplicates or be incomplete.

The following example shows a real-time search of your internal index. Results are displayed in XML. For more about displaying results, output formats, and results readers, see How to display search results.

```
// a normal real-time search, a window of 5 minutes, with timeline data
var queryArgs = new JobArgs
{
    ExecutionMode = JobArgs.ExecutionModeEnum.Normal,
    SearchMode = JobArgs.SearchModeEnum.Realtime,
    EarliestTime = "rt-5m",
    LatestTime = "rt",
    StatusBuckets = 300
};

// Create the job
var mySearch = "search index=_internal";
var job = service.GetJobs().Create(mySearch, queryArgs);

var outputArgs = new JobResultsPreviewArgs
{
    OutputMode = JobResultsPreviewArgs.OutputModeEnum.Xml,
    Count = 0  // Return all entries
};

for (var i = 0; i < 5; i++)
{
    System.Console.WriteLine();
    System.Console.WriteLine();
    System.Console.WriteLine("Snapshot " + i + ":");

    using (var stream = job.ResultsPreview(outputArgs))
    {
    using (var rr = new ResultsReaderXml(stream))
    {
        foreach (var @event in rr)
        {
        System.Console.WriteLine("EVENT:");
        foreach (string key in @event.Keys)
        {
            System.Console.WriteLine(
            "    " + key + " -> " + @event[key]);
        }
        }
    }
    }

    Thread.Sleep(500);
}

job.Cancel();
```

To run an export search

An export search is the most reliable way to return a large set of results because exporting returns results in a stream, rather than as a search job that is saved on the server. So the server-side limitations to the number of results that can be returned don't apply to export searches.

You can run an export search in normal and real-time modes, and running the search is similar to running a regular search. However, displaying the results of an export search is a little tricky because they require different parsing; these issues are covered in To work with results from an export search.

To show you how to run a simple export search, this example runs a normal-mode export search of your internal index over the last hour and uses the SDK's **MultiResultsReaderXml** class to display the output. For more about displaying results, see How to display search results.

```
// a normal search, a window of 5 minutes
var queryArgs = new JobExportArgs
{
    SearchMode = JobExportArgs.SearchModeEnum.Normal,
    EarliestTime = "rt-5m",
    LatestTime = "rt",
};

// Create the job
var mySearch = "search index=_internal";

var outputArgs = new JobResultsPreviewArgs
{
    OutputMode = JobResultsPreviewArgs.OutputModeEnum.Xml ,
    Count = 0  // Return all entries
};

for (var i = 0; i < 5; i++)
{
    System.Console.WriteLine();
    System.Console.WriteLine();
    System.Console.WriteLine("Snapshot " + i + ":");

    using (var stream = service.Export(mySearch, queryArgs))
    {
    using (var rr = new ResultsReaderXml(stream))
    {
        foreach (var @event in rr)
        {
        System.Console.WriteLine("EVENT:");
        foreach (string key in @event.Keys)
        {
            System.Console.WriteLine(
            "    " + key + " -> " + @event[key]);
        }
        }
    }
    }

    Thread.Sleep(500);
}
```

Collection parameters

By default, all entities are returned when you retrieve a collection. Using the parameters below, you can specify the number of entities to return, how to sort them, and so on. Set these parameters by creating a generic **Args** dictionary:

Parameter	Description
`count`	A number that indicates the maximum number of entities to return.
`offset`	A number that specifies the index of the first entity to return.
`search`	A string that specifies a search expression to filter the response with, matching field values against the search expression. For example, "search=foo" matches any object that has "foo" as a substring in a field, and "search=field_name%3Dfield_value" restricts the match to a single field.
`sort_dir`	An enum value that specifies how to sort entities. Valid values are "asc" (ascending order) and "desc" (descending order).
`sort_key`	A string that specifies the field to sort by.
`sort_mode`	An enum value that specifies how to sort entities. Valid values are "auto", "alpha" (alphabetically), "alpha_case" (alphabetically, case sensitive), or "num" (numerically).

Search job parameters

Properties to set

The parameters you can use for search jobs correspond to the parameters for the search/jobs endpoint in the REST API.

This list summarizes the properties you can set for a search job (click here for properties you can retrieve). There are different ways to set these properties--you can create an argument dictionary with these parameters as key-value pairs using the generic **Args** class, or you can use the setters for these specialized classes with different Job methods:

- **JobArgs** class for jobs
- **JobEventsArgs** class for events
- **JobExportArgs** class for export
- **JobResultsArgs** class for results
- **JobResultsPreviewArgs** class for preview results

Parameter	Description
`search`	Required. A string that contains the search query.
`auto_cancel`	The number of seconds of inactivity after which to automatically cancel a job. 0 means never auto-cancel.
`auto_finalize_ec`	The number of events to process after which to auto-finalize the search. 0 means no limit.
`auto_pause`	The number of seconds of inactivity after which to automatically pause a job. 0 means never auto-pause.
`earliest_time`	A time string that 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. For a real-time search, specify "rt".

enable_lookups	A Boolean that indicates whether to apply lookups to events.
exec_mode	An enum value that indicates the search mode ("blocking", "oneshot", or "normal").
force_bundle_replication	A Boolean that indicates whether this search should cause (and wait depending on the value of "sync_bundle_replication") bundle synchronization with all search peers.
id	A string that contains a search ID. If unspecified, a random ID is generated.
index_earliest	A string that specifies the time for the earliest (inclusive) time bounds for the search, based on the index time bounds. The time string can be a UTC time (with fractional seconds), a relative time specifier (to now), or a formatted time string.
index_latest	A string that specifies the time for the latest (inclusive) time bounds for the search, based on the index time bounds. The time string can be a UTC time (with fractional seconds), a relative time specifier (to now), or a formatted time string.
latest_time	A time string that 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. For a real-time search, specify "rt".
max_count	The number of events that can be accessible in any given status bucket.
max_time	The number of seconds to run this search before finalizing. Specify 0 to never finalize.
namespace	A string that contains the application namespace in which to restrict searches.
now	A time string that sets the absolute time used for any relative time specifier in the search.
reduce_freq	The number of seconds (frequency) to run the MapReduce reduce phase on accumulated map values.
reload_macros	A Boolean that indicates whether to reload macro definitions from the macros.conf configuration file.
remote_server_list	A string that contains a comma-separated list of (possibly wildcarded) servers from which to pull raw events. This same server list is used in subsearches.
rf	A string that adds one or more required fields to the search.
rt_blocking	A Boolean that indicates whether the indexer blocks if the queue for this search is full. For real-time searches.
rt_indexfilter	A Boolean that indicates whether the indexer pre-filters events. For real-time searches.
rt_maxblocksecs	The number of seconds indicating the maximum time to block. 0 means no limit. For real-time searches with "rt_blocking" set to "true".

rt_queue_size	The number indicating the queue size (in events) that the indexer should use for this search. For real-time searches.
search_listener	A string that registers a search state listener with the search. Use the format: search_state;results_condition;http_method;uri;
search_mode	An enum value that indicates the search mode ("normal" or "realtime"). If set to "realtime", searches live data. A real-time search is also specified by setting "earliest_time" and "latest_time" parameters to "rt", even if the search_mode is normal or is not set.
spawn_process	A Boolean that indicates whether to run the search in a separate spawned process. Searches against indexes must run in a separate process.
status_buckets	The maximum number of status buckets to generate, which corresponds to the size of the data structure used to store timeline information. This value is also used for summaries. A value of 0 means to not generate timeline or summary information.
sync_bundle_replication	A Boolean that indicates whether this search should wait for bundle replication to complete.
time_format	A string that specifies the format to use to convert a formatted time string from {start,end}_time into UTC seconds.
timeout	The number of seconds to keep this search after processing has stopped.

Properties to retrieve

This list summarizes the properties that are available for an existing search job:

Property	Description
cursorTime	The earliest time from which no events are later scanned.
delegate	For saved searches, specifies jobs that were started by the user.
diskUsage	The total amount of disk space used, in bytes.
dispatchState	The state of the search. Can be any of QUEUED, PARSING, RUNNING, PAUSED, FINALIZING, FAILED, DONE.
doneProgress	A number between 0 and 1.0 that indicates the approximate progress of the search.
dropCount	For real-time searches, the number of possible events that were dropped due to the "rt_queue_size".
eai:acl	The access control list for this job.
eventAvailableCount	The number of events that are available for export.
eventCount	The number of events returned by the search.
eventFieldCount	The number of fields found in the search results.
eventIsStreaming	A Boolean that indicates whether the events of this search are being streamed.

`eventIsTruncated`	A Boolean that indicates whether events of the search have not been stored.
`eventSearch`	Subset of the entire search before any transforming commands.
`eventSorting`	A Boolean that indicates whether the events of this search are sorted, and in which order ("asc" for ascending, "desc" for descending, and "none" for not sorted).
`isDone`	A Boolean that indicates whether the search has finished.
`isFailed`	A Boolean that indicates whether there was a fatal error executing the search (for example, if the search string syntax was invalid).
`isFinalized`	A Boolean that indicates whether the search was finalized (stopped before completion).
`isPaused`	A Boolean that indicates whether the search has been paused.
`isPreviewEnabled`	A Boolean that indicates whether previews are enabled.
`isRealTimeSearch`	A Boolean that indicates whether the search is a real time search.
`isRemoteTimeline`	A Boolean that indicates whether the remote timeline feature is enabled.
`isSaved`	A Boolean that indicates whether the search is saved indefinitely.
`isSavedSearch`	A Boolean that indicates whether this is a saved search run using the scheduler.
`isZombie`	A Boolean that indicates whether the process running the search is dead, but with the search not finished.
`keywords`	All positive keywords used by this search. A positive keyword is a keyword that is not in a NOT clause.
`label`	A custom name created for this search.
`messages`	Errors and debug messages.
`numPreviews`	Number of previews that have been generated so far for this search job.
`performance`	A representation of the execution costs.
`priority`	An integer between 0-10 that indicates the search's priority.
`remoteSearch`	The search string that is sent to every search peer.
`reportSearch`	If reporting commands are used, the reporting search.
`request`	GET arguments that the search sends to splunkd.
`resultCount`	The total number of results returned by the search, after any transforming commands have been applied (such as `stats` or `top`).
`resultIsStreaming`	A Boolean that indicates whether the final results of the search are available using streaming (for example, no transforming operations).
`resultPreviewCount`	The number of result rows in the latest preview results.
`runDuration`	A number specifying the time, in seconds, that the search took to complete.
`scanCount`	The number of events that are scanned or read off disk.

`searchEarliestTime`	The earliest time for a search, as specified in the search command rather than the "earliestTime" parameter. It does not snap to the indexed data time bounds for all-time searches (as "earliestTime" and "latestTime" do).
`searchLatestTime`	The latest time for a search, as specified in the search command rather than the "latestTime" parameter. It does not snap to the indexed data time bounds for all-time searches (as "earliestTime" and "latestTime" do).
`searchProviders`	A list of all the search peers that were contacted.
`sid`	The search ID number.
`ttl`	The time to live, or time before the search job expires after it has finished.

How to create modular inputs

Modular inputs were introduced in Splunk® 5.0. They enable you to extend the Splunk framework to define a custom input capability. Modular inputs are useful when you need to prepare data from a non-standard source. Splunk treats your custom input definitions as if they were part of Splunk's native inputs. You can use the Splunk SDK for C# to programmatically create modular inputs.

This topic contains the following sections:

Modular input SDK example

We recommend taking a look at the **modular_input** example in the SDK's **examples** directory. That example is a full, top-to-bottom sample modular input application that you can modify to suit your needs.

The sample application adds the ability for Splunk to monitor changes to one or more of your system's environment variables. You can try it out now. Install the app by doing the following:

1. Copy the **inputs.conf.spec** file from the **modular_input** directory into the following path, where **$SPLUNK_HOME** is the directory where Splunk was installed, and **app_folder_name** corresponds to any app installed in Splunk (you can use an existing app, such as search, or you can create a new app for the modular input; for more information, see Developing Views and Apps for Splunk Web):

   `$SPLUNK_HOME\etc\apps\app_folder_name\README\`

2. Build the project, and then copy the **modular_input.exe** and **SplunkSDK.dll** files from

   `\splunk-sdk-csharp\examples\modular_input\bin\Debug`

   into

   `$SPLUNK_HOME\etc\apps\app_folder_name\bin\`

3. Restart Splunk.
4. In Splunk, go to **Manager > Data Inputs**. If you see "C# SDK Example: System Environment Variable Monitor" listed, you've installed the example correctly.

5. Click **C# SDK Example: System Environment Variable Monitor**. You can now experiment with the example by clicking New, and then adding an environment variable and polling interval.

You can add a per-modular input default configuration—such as a default host name and default index—by adding an **inputs.conf** file to:

```
$SPLUNK_HOME\etc\apps\app_folder_name\default\
```

To debug the example, on the command line, go to `$SPLUNK_HOME\bin`, and then enter the following command, where *input_name* represents the **name** parameter that you specified when creating the modular input instance in the UI:

```
splunk cmd splunkd print-modinput-config modular_input modular_input://input_name |
..\etc\apps\app_folder_name\bin\modular_input
```

Create a modular input from scratch

If, instead, you prefer to create your modular input from scratch, there are some things to keep in mind. The basic steps for creating a modular input, as described in "Implement modular inputs," are as follows:

- Create a modular input script (in our case, an executable)
- Define a scheme for introspection
    - Introspection scheme details
    - Set up streaming (simple or XML)
    - Specify single or multiple instances of a script
- Set up logging
- Set up external validation (optional)
- Create a modular input spec file

Most of these steps can be accomplished programmatically using the Splunk SDK for C#.

Create a modular input script

The *modular input script* that you build using the Splunk SDK for C# is actually an executable. The modular input script defines the modular input's functionality. Your application derives from the **Script** class to define the modular input.

Define a scheme for introspection

You define an introspection scheme by overriding the **Scheme** property of the **Script** class, which represents the XML output when a modular input is called. The XML output is returned through **stdout** to Splunk. The **Scheme** class enables you to define such parameters as:

- the modular input's title: **Title** property
- its description: **Description** property
- its streaming mode (Simple or XML): **StreamingMode** property
- whether external validation is enabled: **UseExternalValidation** property
- whether to launch a single instance or multiple instances of the script: **UseSingleInstance** property
- the introspection scheme's **endpoint** element: **Endpoint** property, which is a container for the **Argument** properties

You'll need to create a way for events to be streamed into **stdout**, using an **InputDefinition** object as input that determines what events are streamed to Splunk. For an example, see the **StreamEvents** method in the modular input example's **Program.cs** file.

Set up logging

It's best practice for your modular input script to log diagnostic data to **splunkd.log**. You use the **SystemLogger** class to write log messages, which include both a standard splunkd.log levels (such as "DEBUG" or "ERROR") and a descriptive message (**Write** method).

In this SDK's modular input example, the logger is inside the method that streams the events. To retrieve the logged events, run the following search:

```
index=_internal sourcetype="splunkd"
```

The output will contain diagnostic events logged by the modular input, progress messages and unhandled exceptions from the SDK, and other Splunk internal events.

Set up external validation

Your executable should be able to validate the configuration of your input. Create a validation method that uses a **ValidationItems** object as input. (For an example, see the **Validate** method in the modular input example's **Program.cs** file.)

Create a modular input spec file

You will also need to define the configuration for your modular input. You do this by editing the **inputs.conf.spec** file manually. See "Create a modular input spec file" in the main Splunk documentation for instructions, or take a look at the SDK sample's **inputs.conf.spec** file, which is directly inside the **modular_input** directory.

Troubleshooting

Events are being produced by the modular input, but are not being consumed by Splunk.

The Splunk SDK for C# requires that the **Time** property of the **EventElement** object must be fully qualified with a time zone. If the timestamp of the event that is being received by the modular input is, for instance, in Coordinated Universal Time (UTC), but its *kind* is not specified as such, Splunk may not consume the event. For example, if you're consuming events generated by SharePoint 2010, the **SPAuditEntry.Occurred** property represents the date and time of an event expressed in UTC, but the **Kind** field of that timestamp is null. Adding an explicit Kind (in this case, setting it to UTC) fixes the problem.

How to display search results using the Splunk SDK for C#

After you run a search, you can retrieve different output from the search job:

- **Events:** The untransformed events of the search.
- **Results:** The transformed results of the search after processing has been completed. If the search does not have transforming commands, the results are the same as the events. The result count will be less than the event count if there are transforming commands.
- **Results preview:** A preview of a search that is still in progress, or results from a real-time search. When the search is complete, the preview results are the same as the results. You must enable previews for non-real-time searches (previews are enabled automatically for real-time searches).
- **Summary:** Summary information about the fields of a search from the results that have been read thus far. Set "`status_buckets`" on the search job to a positive value to access this data.
- **Timeline:** The event distribution over time of the untransformed events that have been read thus far. Set "`status_buckets`" on the search job to a positive value to access this data.

This output is returned as a stream in XML, JSON, JSON_COLS, JSON_ROWS, CSV, Atom, or raw format. For examples, see Sample output in different formats below.

You can display the direct results using standard C# classes or make your own parser. For convenience, the SDK includes results readers for XML and JSON that properly parse and format the results for you, and handle the idiosyncrasies of each output type for each Splunk® version. For a comparison of XML output displayed using standard C# classes versus the SDK's XML results reader, see Results reader comparison, below.

The search results APIs

Retrieve a search job using the **Job** class (or for streaming searches, retrieve the stream using the Stream class). From the search job, you can retrieve events, results, preview results, the summary, and timeline information:

- The **Job.Events** method retrieves events from a search job. Use the **JobEventsArgs** class to specify additional arguments to the method.
- The **Job.Results** method retrieves results from a search job. Use the **JobResultsArgs** class to specify additional arguments to the method.
- The **Job.ResultsPreview** method retrieves result previews from a search job. Use the **JobResultsPreviewArgs** class to specify additional arguments to the method.
- The **Job.Summary** method retrieves summary data from a search job. Use the **Args** class to specify additional arguments to the method.
- The **Job.Timeline** method retrieves timeline data from a search job. Use the **Args** class to specify additional arguments to the method.

Use the following classes to display results with the results readers. For results from an export search, use the multi-results reader to parse the multiple results sets that are returned.

- The **ResultsReader** class is the base class for the typed results readers.
  - The **ResultsReaderJson** class displays JSON results.
  - The **ResultsReaderXml** class displays XML results.
- The **MultiResultsReader** class is the base class for the typed multi-results readers.
  - The **MultiResultsReaderJson** class displays multiple sets of JSON results.
  - The **MultiResultsReaderXml** class displays multiple sets of XML results.

Code examples

This section provides examples of how to use the job APIs, assuming you first connect to a Splunk instance:

- To display results without a reader
- To display results using a results reader
- To paginate through a large set of results
- To display preview results
- To work with results from an export search

The following parameters are available:

- Event, results, and results preview parameters

To display results without a reader

You can use the basic built-in capabilities of C# to read results streams in different output formats, or you can create your own reader.

This example shows how to set the output mode and display a simple XML results stream using standard C# classes. No need to set XML as the output mode explicitly—XML is the default.

```
// ...
// Create a simple search job
var mySearch = "search * | head 5";
```

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

// Wait for the job to finish
while (!job.IsDone) {
    Thread.Sleep(500);
}

// Create a UTF-8 encoding
UTF8Encoding utf8 = new UTF8Encoding();

// Display results
var results = job.Results();
String line = null;
System.Console.WriteLine("Results from the search job as XML:\n");
StreamReader sr = new StreamReader(results, utf8);
while ((line = sr.ReadLine()) != null) {
    System.Console.WriteLine(line);
}

sr.Close();
```

To display results using a results reader

For convenience, this SDK includes results readers for XML and JSON that parse and format results for you, and handle the idiosyncrasies of each output type:

- Use the **ResultsReaderXml** class for XML, which is the default format.
- Use the **ResultsReaderJson** class for JSON.

This example shows how to set the output mode to JSON and display the results stream using the **ResultsReaderJson** class:

```
// Create a simple search job
var mySearch = "search * | head 5";
var job = service.GetJobs().Create(mySearch);

// Wait for the job to finish
while (!job.IsDone) {
    try {
    Thread.Sleep(500);
    } catch (ThreadInterruptedException e) {
    // TODO Auto-generated catch block
    System.Console.WriteLine(e.StackTrace);
    }
}

// Specify JSON as the output mode for results
var resultsArgs = new JobResultsArgs {
    OutputMode = JobResultsArgs.OutputModeEnum.Json,
    Count = 0 // Return all entries.
};

System.Console.WriteLine("\nFormatted results from the search job as JSON\n");

// Display results in JSON using ResultsReaderJson
using (var stream = job.Results(resultsArgs))
{
    using (var rr = new ResultsReaderJson(stream))
    {
```

```
    foreach (var @event in rr)
    {
        foreach (string key in @event.Keys)
        {
        System.Console.WriteLine("   " + key + " -> " + @event[key]);
        }
    }
    }
}
```

To paginate through a large set of results

The maximum number of results you can retrieve at a time from your search results is determined by the `maxresultrows` field, which is specified in a Splunk configuration file. Here's a quick way to find out what your system setting is:

```
// Find out how many results your system is configured to return
Entity restApi = service.GetConfs().Get("limits").Get("restapi");
var maxResults = restApi.Get("maxresultrows");
System.Console.WriteLine("Your system is configured to return a maximum of " +
maxResults + " results");
```

However, we don't recommend changing the default value of 50,000. If your job has more results than this limit, you can retrieve your results in sets (0-49999, then 50000-99999, and so on), using the "count" and "offset" parameters to define how many results to retrieve at a time:

1. Set the count to a value up to the size of `maxresultrows` to define the number of results in a set.
2. Retrieve this set of results.
3. Increment the offset by the count to retrieve the next set.
4. Repeat until you've retrieved all of your results.

The example below shows how to retrieve 50 search results in sets of 10, and uses the **ResultsReaderXml** class to parse and format the results.

```
// Create a simple job that returns 50 results
String mySearch = "search * | head 50";
Job job = service.GetJobs().Create(mySearch);

// Wait for the job to finish
while (!job.IsDone) {
    try {
    Thread.Sleep(500);
    } catch (ThreadInterruptedException e) {
    // TODO Auto-generated catch block
    System.Console.WriteLine(e.StackTrace);
    }
}

// Page through results by looping through sets of results
int resultCount = job.ResultCount; // Number of results this job returned
int x = 0;      // Result counter
int offset = 0;     // Start at result 0
int count = 10;     // Get sets of 10 results at a time

// Loop through each set of results
while (offset < resultCount) {
    var resultsArgs = new JobResultsArgs {
    OutputMode = JobResultsArgs.OutputModeEnum.Xml,
    Count = count,
```

```
    Offset = offset
    };

    // Display results in XML using ResultsReaderXml
    using (var stream = job.Results(resultsArgs))
    {
    using (var rr = new ResultsReaderXml(stream))
    {
        foreach (var @event in rr)
        {
        System.Console.WriteLine("\n***** RESULT " + x++ + " *****\n");
        foreach (string key in @event.Keys)
        {
            System.Console.WriteLine("   " + key + " -> " + @event[key]);
        }
        }
    }
    }

    // Increase the offset to get the next set of results
    offset = offset + count;
};
```

Another option when working with a very large data set is to use an export search, where data is streamed directly from a search rather than saved to the server as a search job. For more, see To run an export search and To work with results from an export search.

To display preview results

You can display a preview of the results of a search that is in progress as long as a couple conditions are met:

- The search must be run in normal execution mode ("`exec_mode`" is "`normal`"). Previews aren't available for blocking searches (the search job ID is not available until the search is done) or streaming searches (results are returned as they become available anyway.)
- Previews must be enabled ("`preview`" is "`1`"). By default, previews are only enabled for real-time searches, and searches with "`status_buckets`" set to a positive value. Use the **Job.enablePreview** method to enable previews for an existing search job.

To display the previews, run the search, enable previews for the search job, then retrieve the preview results from it. By default, the most recent 100 previews are retrieved. To change this number, set a value for "`count`". Use the "`offset`" value to page through large sets of previews.

The following example runs a normal search, enables preview for the search job, and then displays preview results while the search runs. For an example of a real-time search, see To run a real-time search.

```
String mySearch = "search * | head 50000";

// Create an argument dictionary for the job arguments:
JobArgs jobArgs = new JobArgs {
    ExecutionMode = JobArgs.ExecutionModeEnum.Normal
};

// Create the job
Job job = service.GetJobs().Create(mySearch, jobArgs);
job.EnablePreview();
job.Update();

// Wait for the job to be ready
while (!job.IsReady) {
```

```
    Thread.Sleep(500);
}

// Display previews using the built-in XML parser
int countPreview=0;  // count the number of previews displayed
int countBatch=0;    // count the number of times previews are retrieve
while (!job.IsDone) {
    JobResultsPreviewArgs previewargs = new JobResultsPreviewArgs {
    Count = 500,  // Get 500 previews at a time
    OutputMode = JobResultsPreviewArgs.OutputModeEnum.Xml
    };

    // Display results in XML using ResultsReaderXml
    using (var stream = job.ResultsPreview(previewargs))
    {
    using (var rr = new ResultsReaderXml(stream))
    {
        foreach (var @event in rr)
        {
        System.Console.WriteLine("BATCH " + countBatch + "\nPREVIEW " + countPreview++
+ " *****");
        foreach (string key in @event.Keys)
        {
            System.Console.WriteLine("   " + key + " -> " + @event[key]);
        }
        }
    }
    }
    countBatch++;
}
System.Console.WriteLine("Job is done with " + job.ResultCount + " results");
```

To work with results from an export search

Working with search results from export searches is a little different than that of regular searches:

- A reporting (transforming) search returns a set of previews followed by the final events, each as separate elements.
- A non-reporting (non-transforming) search returns events as they are read from the index, each as separate elements.
- A real-time search returns multiple sets of previews, each preview as a separate element.
- For JSON output, each result set is not returned as a single JSON object, but rather each row is an individual object, where rows are separated by a new line and the last row of the set is indicated by `"lastrow":true`.

Here's sample JSON output that shows two results sets, each with five rows:

```
{"preview":true,"offset":0,"result":{"sourcetype":"eventgen-2","count":"58509"}}
{"preview":true,"offset":1,"result":{"sourcetype":"splunk_web_service","count":"119"}}
{"preview":true,"offset":2,"result":{"sourcetype":"splunkd","count":"4153"}}
{"preview":true,"offset":3,"result":{"sourcetype":"splunkd_access","count":"12"}}
{"preview":true,"offset":4,"lastrow":true,"result":{"sourcetype":"splunkd_stderr","cou
nt":"2"}}
{"preview":true,"offset":0,"result":{"sourcetype":"eventgen-2","count":"60886"}}
{"preview":true,"offset":1,"result":{"sourcetype":"splunk_web_service","count":"119"}}
{"preview":true,"offset":2,"result":{"sourcetype":"splunkd","count":"4280"}}
{"preview":true,"offset":3,"result":{"sourcetype":"splunkd_access","count":"12"}}
{"preview":true,"offset":4,"lastrow":true,"result":{"sourcetype":"splunkd_stderr","cou
nt":"2"}}
```

This format allows results to be sent as a continuous stream of JSON data that is still easy to parse.

So, we recommend using the SDK's multi-results readers to parse the output—we've already done some of the heavy lifting here, and these results readers handle the output appropriately. Display the results stream in XML or JSON, and use one of the **MultiResultsReader** classes to parse and format the results.

This example below shows how to display a real-time export search with XML output, using a 30-second time range:

```
// Set up the job properties
var mySearch = "search index=_internal | head 10";

// Set up a real-time export with a 30-second window
JobExportArgs jobArgs = new JobExportArgs() {
    SearchMode = JobExportArgs.SearchModeEnum.Realtime,
    EarliestTime = "rt-30s",
    LatestTime = "rt",
    OutputMode= JobExportArgs.OutputModeEnum.Xml
};

// Create the job
var exportStream = service.Export(mySearch, jobArgs);

// Display previews
var multiResultsReader = new MultiResultsReaderXml(exportStream);

int counterSet = 0;  // count the number of results sets
foreach (ISearchResults searchResults in multiResultsReader)
{
    System.Console.WriteLine("Result set " + counterSet++ + " ********");
    int counterEvent = 0;
    foreach (var @event in searchResults)
    {
    System.Console.WriteLine("Event " + counterEvent++ + " --------");
    foreach (string key in @event.Keys)
    {
        System.Console.WriteLine("   " + key + " -> " + @event[key]);
    }
    }
}
```

This next example shows how to display previews from a normal export search. This search is useful when you are performing a reporting (transforming) search on a large data set and you want to view previews while the search runs.

```
String mySearch = "search * | stats count by host";

JobExportArgs jobArgs = new JobExportArgs {
    OutputMode = JobExportArgs.OutputModeEnum.Xml
};

// Create the job
var stream = service.Export(mySearch, jobArgs);

// Display previews
MultiResultsReaderXml multiResultsReader = new MultiResultsReaderXml(stream);

int counterSet = 0;
foreach (ISearchResults searchResults in multiResultsReader)
{
    // Display whether the results is a preview (search in progress) or
```

```
    // final (search is finished)
    String resultSetType = searchResults.IsPreview ? "Preview":"Final";
    System.Console.WriteLine(resultSetType + " result set " + counterSet++ + "
********");
    int counterEvent = 0;
    foreach (var @event in searchResults)
    {
    System.Console.WriteLine("Event " + counterEvent++ + " --------");
    foreach (string key in @event.Keys)
    {
        System.Console.WriteLine("    " + key + " -> " + @event[key]);
    }
    }
}
```

For an example showing a simple export search, see To run an export search.

Sample output in different formats

The following is sample output in different formats for the search "`search sourcetype=access_combined_wcookie | head 1`":

```
***** ATOM *****

<?xml version='1.0' encoding='UTF-8'?>
<results preview='0'>
<meta>
<fieldOrder>
<field>_bkt</field>
<field>_cd</field>
<field>_indextime</field>
<field>_raw</field>
<field>_serial</field>
<field>_si</field>
<field>_sourcetype</field>
<field>_time</field>
<field>host</field>
<field>index</field>
<field>linecount</field>
<field>source</field>
<field>sourcetype</field>
<field>splunk_server</field>
</fieldOrder>
</meta>
    <result offset='0'>
        <field k='_bkt'>
            <value><text>main~22~016AA522-9069-446D-BF3A-79DC1B5A458
4</text></value>
        </field>
        <field k='_cd'>
            <value><text>22:25717531</text></value>
        </field>
        <field k='_indextime'>
            <value><text>1370383278</text></value>
        </field>
        <field k='_raw'><v xml:space='preserve' trunc='0'>233.77.49.53 -
 - [02/Jun/2013:16:14:43] &amp;quot;GET /flower_sto</v></field>
        <field k='_serial'>
            <value><text>0</text></value>
        </field>
```

```
        <field k='_si'>
            <value><text>TESTBOX</text></value>
            <value><text>main</text></value>
        </field>
        <field k='_sourcetype'>
            <value><text>access_combined_wcookie</text></value>
        </field>
        <field k='_time'>
            <value><text>2013-06-02T16:14:43.000-07:00</text></value
>
        </field>
        <field k='host'>
            <value><text>apache2.splunk.com</text></value>
        </field>
        <field k='index'>
            <value><text>main</text></value>
        </field>
        <field k='linecount'>
            <value><text>1</text></value>
        </field>
        <field k='source'>
            <value><text>Sampledata.zip:.\apache2.splunk.com/access_
combined.log</text></value>
        </field>
        <field k='sourcetype'>
            <value><text>access_combined_wcookie</text></value>
        </field>
        <field k='splunk_server'>
            <value><text>TESTBOX</text></value>
        </field>
    </result>
</results>

***** CSV *****

"_bkt","_cd","_indextime","_raw","_serial","_si","_sourcetype","_time",host,inde
x,linecount,source,sourcetype,"splunk_server"
"main~22~016AA522-9069-446D-BF3A-79DC1B5A4584","22:25717531",1370383278,"233.77.
49.53 - - [02/Jun/2013:16:14:43] ""GET /flower_sto",0,"TESTBOX
main","access_combined_wcookie","2013-06-02T16:14:43.000-07:00","apache2.splunk.
com",main,1,"Sampledata.zip:.\apache2.splunk.com/access_combined.log","access_co
mbined_wcookie","TESTBOX"

***** JSON *****

{"preview":false,"init_offset":0,"messages":[{"type":"DEBUG","text":"base lispy:
 [ AND sourcetype::access_combined_wcookie ]"},{"type":"DEBUG","text":"search co
ntext: user=\"admin\", app=\"search\", bs-pathname=\"C:\\Program Files\\Splunk\\
etc\""},{"type":"WARN","text":"Unable to distribute to peer named 10.80.1.59:808
9 at uri https://10.80.1.59:8089 because peer has status = \"Down\". "}],"result
s":[{"_bkt":"main~22~016AA522-9069-446D-BF3A-79DC1B5A4584","_cd":"22:25717531","
_indextime":"1370383278","_raw":"233.77.49.53 - - [02/Jun/2013:16:14:43] \"GET /
flower_sto","_serial":"0","_si":"TESTBOX\nmain","_sourcetype":"access_combin
ed_wcookie","_time":"2013-06-02T16:14:43.000-07:00","host":"apache2.splunk.com",
"index":"main","linecount":"1","source":"Sampledata.zip:.\\apache2.splunk.com/ac
cess_combined.log","sourcetype":"access_combined_wcookie","splunk_server":"MTEVE
NAN-VM"}]}

***** JSON_COLS *****
```

{"preview":false,"init_offset":0,"messages":[{"type":"DEBUG","text":"base lispy:
 [AND sourcetype::access_combined_wcookie]"},{"type":"DEBUG","text":"search co
ntext: user=\"admin\", app=\"search\", bs-pathname=\"C:\\Program Files\\Splunk\\
etc\""},{"type":"WARN","text":"Unable to distribute to peer named 10.80.1.59:808
9 at uri https://10.80.1.59:8089 because peer has status = \"Down\". "}],"fields
":["_bkt","_cd","_indextime","_raw","_serial","_si","_sourcetype","_time","host"
,"index","linecount","source","sourcetype","splunk_server"],"columns":[["main~22
~016AA522-9069-446D-BF3A-79DC1B5A4584"],["22:25717531"],["1370383278"],["233.77.
49.53 - - [02/Jun/2013:16:14:43] \"GET /flower_sto"],["0"],[["TESTBOX","main
"]],["access_combined_wcookie"],["2013-06-02T16:14:43.000-07:00"],["apache2.splu
nk.com"],["main"],["1"],["Sampledata.zip:.\\apache2.splunk.com/access_combined.l
og"],["access_combined_wcookie"],["TESTBOX"]]}

***** JSON_ROWS *****

{"preview":false,"init_offset":0,"messages":[{"type":"DEBUG","text":"base lispy:
 [AND sourcetype::access_combined_wcookie]"},{"type":"DEBUG","text":"search co
ntext: user=\"admin\", app=\"search\", bs-pathname=\"C:\\Program Files\\Splunk\\
etc\""},{"type":"WARN","text":"Unable to distribute to peer named 10.80.1.59:808
9 at uri https://10.80.1.59:8089 because peer has status = \"Down\". "}],"fields
":["_bkt","_cd","_indextime","_raw","_serial","_si","_sourcetype","_time","host"
,"index","linecount","source","sourcetype","splunk_server"],"rows":[["main~22~01
6AA522-9069-446D-BF3A-79DC1B5A4584","22:25717531","1370383278","233.77.49.53 - -
 [02/Jun/2013:16:14:43] \"GET /flower_sto","0",["TESTBOX","main"],"access_co
mbined_wcookie","2013-06-02T16:14:43.000-07:00","apache2.splunk.com","main","1",
"Sampledata.zip:.\\apache2.splunk.com/access_combined.log","access_combined_wcoo
kie","TESTBOX"]]}

***** RAW *****

233.77.49.53 - - [02/Jun/2013:16:14:43] "GET /flower_sto

***** XML *****

<?xml version='1.0' encoding='UTF-8'?>
<results preview='0'>
<meta>
<fieldOrder>
<field>_bkt</field>
<field>_cd</field>
<field>_indextime</field>
<field>_raw</field>
<field>_serial</field>
<field>_si</field>
<field>_sourcetype</field>
<field>_time</field>
<field>host</field>
<field>index</field>
<field>linecount</field>
<field>source</field>
<field>sourcetype</field>
<field>splunk_server</field>
</fieldOrder>
</meta>
    <result offset='0'>
        <field k='_bkt'>
            <value><text>main~22~016AA522-9069-446D-BF3A-79DC1B5A458
4</text></value>

```
        </field>
        <field k='_cd'>
            <value><text>22:25717531</text></value>
        </field>
        <field k='_indextime'>
            <value><text>1370383278</text></value>
        </field>
        <field k='_raw'><v xml:space='preserve' trunc='0'>233.77.49.53 -
 - [02/Jun/2013:16:14:43] &quot;GET /flower_sto</v></field>
        <field k='_serial'>
            <value><text>0</text></value>
        </field>
        <field k='_si'>
            <value><text>TESTBOX</text></value>
            <value><text>main</text></value>
        </field>
        <field k='_sourcetype'>
            <value><text>access_combined_wcookie</text></value>
        </field>
        <field k='_time'>
            <value><text>2013-06-02T16:14:43.000-07:00</text></value
>
        </field>
        <field k='host'>
            <value><text>apache2.splunk.com</text></value>
        </field>
        <field k='index'>
            <value><text>main</text></value>
        </field>
        <field k='linecount'>
            <value><text>1</text></value>
        </field>
        <field k='source'>
            <value><text>Sampledata.zip:.\apache2.splunk.com/access_
combined.log</text></value>
        </field>
        <field k='sourcetype'>
            <value><text>access_combined_wcookie</text></value>
        </field>
        <field k='splunk_server'>
            <value><text>TESTBOX</text></value>
        </field>
    </result>
</results>
```

Results reader comparison

Here's a single search result. For comparison, the output is displayed using the standard C# classes and the SDK's XML results reader:

```
***** Output from standard C# classes *****

<?xml version='1.0' encoding='UTF-8'?>
<results preview='0'>
<meta>
<fieldOrder>
<field>_bkt</field>
<field>_cd</field>
<field>_indextime</field>
```

```
<field>_raw</field>
<field>_serial</field>
<field>_si</field>
<field>_sourcetype</field>
<field>_time</field>
<field>host</field>
<field>index</field>
<field>linecount</field>
<field>source</field>
<field>sourcetype</field>
<field>splunk_server</field>
</fieldOrder>
</meta>
    <result offset='0'>
        <field k='_bkt'>
            <value><text>main~4~016AA522-9069-446D-BF3A-79DC1B5A4584
</text></value>
        </field>
        <field k='_cd'>
            <value><text>4:276188</text></value>
        </field>
        <field k='_indextime'>
            <value><text>1366746411</text></value>
        </field>
        <field k='_raw'><v xml:space='preserve' trunc='0'>10.192.1.43 -
- [21/Apr/2013:14:13:0</v></field>
        <field k='_serial'>
            <value><text>0</text></value>
        </field>
        <field k='_si'>
            <value><text>TESTBOX-VM</text></value>
            <value><text>main</text></value>
        </field>
        <field k='_sourcetype'>
            <value><text>access_combined_wcookie</text></value>
        </field>
        <field k='_time'>
            <value><text>2013-04-21T14:13:00.000-07:00</text></value
>
        </field>
        <field k='host'>
            <value><text>apache2.splunk.com</text></value>
        </field>
        <field k='index'>
            <value><text>main</text></value>
        </field>
        <field k='linecount'>
            <value><text>1</text></value>
        </field>
        <field k='source'>
            <value><text>Sampledata.zip:.\apache2.splunk.com/access_
combined.log</text></value>
        </field>
        <field k='sourcetype'>
            <value><text>access_combined_wcookie</text></value>
        </field>
        <field k='splunk_server'>
            <value><text>TESTBOX-VM</text></value>
        </field>
```

```
    </result>
</results>

***** Output from the XML results reader *****

   _bkt -> main~4~016AA522-9069-446D-BF3A-79DC1B5A4584
   _cd -> 4:276188
   _indextime -> 1366746411
   _raw -> 10.192.1.43 - - [21/Apr/2013:14:13:0
   _serial -> 0
   _si -> TESTBOX-VM,main
   _sourcetype -> access_combined_wcookie
   _time -> 2013-04-21T14:13:00.000-07:00
   host -> apache2.splunk.com
   index -> main
   linecount -> 1
   source -> Sampledata.zip:.\apache2.splunk.com/access_combined.log
   sourcetype -> access_combined_wcookie
   splunk_server -> TESTBOX-VM
```

Event, results, and results preview parameters

Set these parameters using setters for the following classes:

- **JobEventsArgs** class
- **JobResultsArgs** class
- **JobResultsPreviewArgs** class

For more, see the POST search/jobs endpoint.

Parameter	Description	Applies to
`count`	A number that indicates the maximum number of results to return.	events, results, results preview
`earliest_time`	A time string that 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. For a real-time search, specify "rt".	events
`f`	A string that contains the field to return for the event set.	events, results, results preview
`field_list`	A string that contains a comma-separated list of fields to return for the event set.	events, results, results preview
`latest_time`	A time string that specifies the earliest time in the time range to search. The time string can be a UTC time (with fractional seconds), a relative	events

	time specifier (to now), or a formatted time string. For a real-time search, specify "rt".	
max_lines	The maximum number of lines that any single event's "_raw" field should contain.	events
offset	A number of the index of the first result (inclusive) from which to begin returning data. This value is 0-indexed.	events, results, results preview
output_mode	Specifies the output format of the results (XML, JSON, JSON_COLS, JSON_ROWS, CSV, ATOM, or RAW).	events, results, results preview
output_time_format	A string that contains a UTC time format.	events
search	A string that contains the post-processing search to apply to results.	events, results, results preview
segmentation	A string that contains the type of segmentation to perform on the data.	events
time_format	A string that contains the expression to convert a formatted time string from {start,end}_time into UTC seconds.	events
truncation_mode	A string that specifies how "max_lines" should be achieved ("abstract" or "truncate").	events

Troubleshooting

This topic describes how to troubleshoot problems when coding with the Splunk® SDK for C#. It contains the following sections:

If you still have questions after reading this topic, see the **Questions?** sidebar on the right side of this page for additional help.

Events are being produced by the modular input, but are not being consumed by Splunk.

The Splunk SDK for C# requires that the **Time** property of the **EventElement** object must be fully qualified with a time zone. If the timestamp of the event that is being received by the modular input is, for instance, in Coordinated Universal Time (UTC), but its *kind* is not specified as such, Splunk may not consume the event. For example, if you're consuming events generated by SharePoint 2010, the **SPAuditEntry.Occurred** property represents the date and time of an event expressed in UTC, but the **Kind** field of that timestamp is null. Adding an explicit **Kind** (in this case, setting it to UTC) fixes the problem.

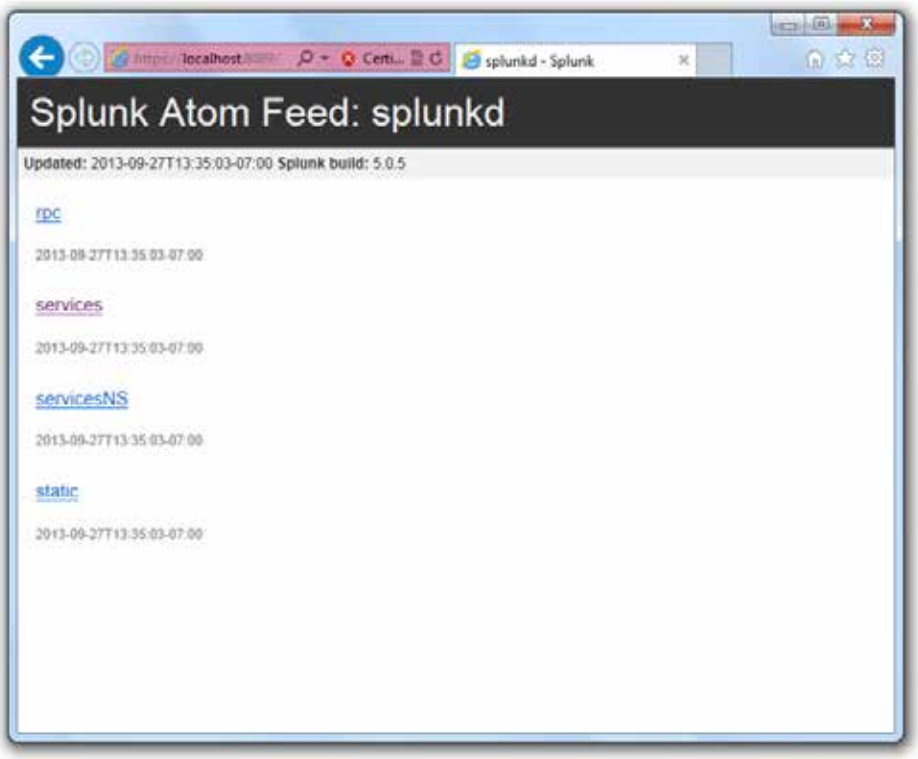
My C# app is not able to successfully connect to my Splunk Enterprise instance.

or

I'm not seeing the correct information (for instance, search reports (saved searches in Splunk Enterprise 5) are wrong or aren't appearing) when I connect.

There are several possibilities for what's wrong:

- *Double-check the host and port in* **.splunkrc**. Is the hostname correct? Is the port number correct? By default, the port for splunkd is 8089, and splunkd is what the Splunk SDK for C# is connecting to.
- *Your permissions are incorrect or insufficient.* If your permissions are incorrect or insufficient, you might be preventing the Splunk SDK for C# from seeing the search reports you're expecting. Try the following:
  1. Using the values for **scheme**, **host**, and **port** that you entered into the **.splunkrc** file, construct a URL as follows in the address bar of a new browser window: *scheme://host:port*
  2. Press Enter. If you see a window warning you about a problem with a security certificate, choose the option to continue. Assuming the URL is correct, you are presented with a window that looks like the following:



3. Click the link for "services".
4. When prompted, enter the same credentials that you entered into the **.splunkrc** file.

   If, after clicking **OK**, you see an expanded list of endpoints, then the credentials you entered are incorrect. Re-enter your credentials, or check with your Splunk Enterprise administrator.

5. In the list of services, click the link for "saved".
6. Click "searches". If the "savedsearch" page that appears is empty, that means there are either no global search reports within your Splunk Enterprise instance, or no search reports in the default app of the current user.

- *The **.splunkrc** file may not be in the correct Windows user's home folder.* Double-check that it is directly inside the current Windows user's home folder.

If you don't see the search reports you expected, you'll need to change their permissions when logged in as someone with sufficient permissions, or create new ones.